

Kristian Lynghjem Vegsund  
Modestas Sukarevicius

## Intergalactic Machine Vision

Detecting dark matter through gravitational  
lensing with machine learning

Bachelor's thesis in Automation and Robotics

Supervisor: Hans Georg Schaathun

Co-supervisor: Ben David Normann

May 2023





Kristian Lynghjem Vegsund  
Modestas Sukarevicius

# Intergalactic Machine Vision

Detecting dark matter through gravitational lensing  
with machine learning

Bachelor's thesis in Automation and Robotics  
Supervisor: Hans Georg Schaathun  
Co-supervisor: Ben David Normann  
May 2023

Norwegian University of Science and Technology  
Faculty of Information Technology and Electrical Engineering  
Department of ICT and Natural Sciences



Norwegian University of  
Science and Technology





Norwegian University of  
Science and Technology

# **Intergalactic Machine Vision**

**Detecting dark matter through gravitational lensing  
with machine learning**

**Bachelor Thesis**

**Faculty of Information Technology and Electrical Engineering  
Department of ICT and Natural Sciences**

Kristian Lynghjem Vegsund; Modestas Sukarevicius

22nd May 2023



# Foreword

Kristian Lynghjem Vegsund

Student at NTNU in Ålesund,  
B.Sc. Automation and Robotics  
Ålesund, 22.05.23

Modestas Sukarevicius

Student at NTNU in Ålesund,  
B.Sc. Automation and Robotics  
Ålesund, 22.05.23

This bachelor thesis is written by two students from NTNUs Electrical Engineering - Automation and Robotics programme. The project is part of a broader scientific project by NTNU, building on what other students did last year. The motivation for us choosing this task was twofold. We both have an interest in the cosmos, and find machine learning to be a very interesting field. This project gave us a unique chance to combine the two, and help a bigger project along the way. The project was open, and we were given lenient autonomy to choose what we wanted to contribute with.

We express our gratitude to our supervisors, Hans Georg Schaathun and Ben David Normann. Their insight and help has been crucial to not only this project, but to our learning as well. Without their help and motivation this project would have no doubt proved too challenging.

## Terminology

**Bias** A term added to the weighted sum of previous layer before feeding into next neuron.

**Ground truth** Correct result that is hidden until a prediction has been made

**High Performance Computing Cluster** A cluster of computers specialised at data-intensive tasks

**Hyperparameter** Parameters that directly controls the learning process

**Idun** A computer cluster (High Performance Computing Group) owned by NTNU

**Loss** Indication of performance of the neural network, where less loss is preferable

**RGB** A colour model that displays colours by mixing red, green, and blue

**Weight** Multiplies the activation from one neuron before feeding into next neuron.

## Notation

$R_E$  Einstein radius indicating lens strength

$\chi_L$  Distance from the observer to the lens plane

$\chi_S$  Distance from the observer to the source plane

$\chi$   $\chi_L/\chi_S$ , a ratio between the two distances

$(x', y')$  Cartesian coordinates in source plane

$(r, \phi)$  Polar coordinates in lens plane

$(\sigma_1, \sigma_2)$  Major and minor standard deviation along axis

$\theta$  Rotation of the source

## **Abbreviations**

**CNN** Convolutional neural network

**CPU** Central Processing Unit

**GL** Gravitational lensing

**GPU** Graphics Processing Unit

**HPCC** High Performance Computing Cluster

**ML** Machine learning

**NN** Neural network

**SIS** Singular Isothermal Sphere

**ViT** Vision Transformer

## Summary

Gravitational lensing gives us an indirect probe on dark matter, thus allowing us to map out its distribution in the Universe. In this project, we further the development of an open-source project aiming at mapping dark matter through the aid of machine learning. Building on previous work, we employ the newly developed framework of the roulette formalism, which is notable in combining both weak and strong lensing. Within this paradigm, we successfully demonstrate that undistorted images can be recovered from distorted ones, and by such making a first preparatory step towards the implementation of this formalism with real data. We set out to find the best machine learning network possible to detect the parameters of these lenses. Using this, it should be further possible to map dark matter automatically on real images. Out of all the networks tested, Inception-v3 and VGG-19\_BN are found to be particularly well suited, and are able to predict lens parameters with very high accuracy.

Furthermore, we discover a surprising result when comparing the point mass lens formula and the images produced, and argue that this is due to an error in previous work that needs to be further investigated. Finally, we recommend multiple steps for the continued development of the project.



# Contents

Foreword . . . . .	1
Terminology . . . . .	2
Notation . . . . .	2
Abbreviations . . . . .	3
Summary . . . . .	4
List of Figures . . . . .	10
List of Tables . . . . .	11
<b>1. Introduction</b>	<b>12</b>
1.1. Background . . . . .	12
1.2. Problem . . . . .	12
1.3. Scope of the project . . . . .	13
1.4. Report structure . . . . .	13
<b>2. Theory</b>	<b>15</b>
2.1. Cosmology . . . . .	15
2.2. Machine learning . . . . .	18
<b>3. Development</b>	<b>23</b>
3.1. Testing of previous work . . . . .	23
3.2. AlexNet estimating same output for different inputs . . . . .	29
3.3. Creating a reference network . . . . .	36
3.4. Hyperparameter optimisation . . . . .	38
Choosing the optimiser . . . . .	38
Loss function choice . . . . .	38
Choosing which parameters to predict . . . . .	40
Creating a data set . . . . .	41
3.5. Achieving the best results . . . . .	44
<b>4. Results</b>	<b>48</b>
4.1. Results deemed interesting . . . . .	48
4.2. The best performing network . . . . .	58
4.3. How to run the system . . . . .	61
<b>5. Discussion</b>	<b>62</b>
5.1. Blunders to learn from . . . . .	62

*Contents*

5.2. AlexNet predicting same output for different inputs . . . . .	63
5.3. Achieving the best results . . . . .	63
5.4. Discussing best network results . . . . .	65
5.5. Interesting tidbits . . . . .	67
5.6. Suggestion for future work . . . . .	68
<b>6. Retrospective</b>	<b>70</b>
6.1. Project management . . . . .	70
6.2. Prerequisite knowledge . . . . .	70
6.3. Learning outcomes . . . . .	71
<b>7. Conclusion</b>	<b>73</b>
<b>A. Appendix Cosmo-ML code</b>	<b>A1</b>
<b>B. Appendix CosmoSim code</b>	<b>B1</b>
<b>C. Other results</b>	<b>C1</b>
<b>D. Pre-project report</b>	<b>D1</b>
<b>E. Supervisor meeting notes</b>	<b>E1</b>
<b>F. Time lists</b>	<b>F1</b>
<b>G. Code tutorial</b>	<b>G1</b>
<b>H. Code for comparing images</b>	<b>H1</b>

# List of Figures

2.1.	Different types of lensing visualised. . . . .	16
2.2.	Figures showing LeNet-5(2.2a), AlexNet with default amount of layers (2.2b) and with extra added regression layers (2.2c) (Anwar 2022). . . . .	19
3.1.	Total loss per epoch of initial code from last year. . . . .	24
3.2.	Standard deviation along major (blue) and minor (green) axis of elliptical source, shown in simulator software. . . . .	26
3.3.	Subfigures showing the difference between two images with an absolute difference of 1.003. . . . .	27
3.4.	Figures showing the supposed same image, but with figure 3.4b having three barely visible dots not supposed to be there. . . . .	30
3.5.	Figures showing total loss with different hyperparameters. . . . .	33
3.6.	Figures showing total loss with different learning rates. . . . .	34
3.7.	Histogram showing deviation from ground truth in different settings. . . . .	37
3.8.	Total loss over 50 epochs using Adam and RMSprop. . . . .	39
3.9.	Images showing the lack of visual disparity between large numerical disparity. . . . .	40
3.10.	Simulator showing how difference in radius can impact visual artifacts. . . . .	42
3.11.	Histogram showing the deviation from ground truth with Inception-v3 architecture with different modifications. . . . .	46
3.12.	Graphs showing the total MAE loss per epoch on test and training data set. . . . .	47
4.1.	Figures showing total loss (logarithmic) with different network architectures. Subfigures only show the best result achieved with that network model. The blue line represent training data set and yellow line represents testing data set. . . . .	49
4.2.	Histograms from all the best performances for each network when limited to 100 epochs. . . . .	50
4.3.	Histograms showing the lowest deviations from ground truth using VGG-19_BN and Inception-v3. . . . .	51
4.4.	Histograms showing different configurations of Inception-v3 and VGG-19_BN run for either 100 epochs or 200 epochs. . . . .	52
4.5.	Histograms showing the best results from Inception-v3 and VGG-19_BN zoomed in to +/-5. . . . .	53

*List of Figures*

4.6.	Histograms showing the best results from Inception-v3 and VGG-19_BN zoomed in to +/-5. . . . .	54
4.7.	Histograms showing the performance of different vision transformers. . .	55
4.8.	Histograms showing the deviations from ground truth on different Inception-v3 configurations. . . . .	56
4.9.	Histograms showing the deviations from ground truth on different VGG-19_BN. . . . .	57
4.10.	Histogram showing the deviation from ground truth on all estimated parameters. . . . .	59
4.11.	Comparing best result to reference network. . . . .	60
5.1.	Histograms separating the results from coordinates and the other parameters estimated by AlexNet reference network. . . . .	64
C.1.	AlexNet total loss performance with logarithmic graph . . . . .	C2
C.2.	ConvNeXt total loss performance with logarithmic graph . . . . .	C3
C.3.	DenseNet total loss performance with logarithmic graph . . . . .	C4
C.4.	EfficientNet-B7 total loss performance with logarithmic graph . . . . .	C5
C.5.	EfficientNet-v2_1 total loss performance with logarithmic graph . . . . .	C6
C.6.	Inception-v3 total loss performance with logarithmic graph . . . . .	C7
C.7.	MnasNet total loss performance with logarithmic graph . . . . .	C8
C.8.	ResNet-152 total loss performance with logarithmic graph . . . . .	C9
C.9.	SqueezeNet-v1.1 total loss performance with logarithmic graph . . . . .	C10
C.10.	Swin-v2-b total loss performance with logarithmic graph . . . . .	C11
C.11.	VGG-19_bn total loss performance with logarithmic graph . . . . .	C12
C.12.	ViT-16-b total loss performance with logarithmic graph . . . . .	C13
C.13.	AlexNet total loss performance with linear graph . . . . .	C14
C.14.	ConvNeXt total loss performance with linear graph . . . . .	C15
C.15.	DenseNet-201 total loss performance with linear graph . . . . .	C16
C.16.	EfficientNet-B7 total loss performance with linear graph . . . . .	C17
C.17.	EfficientNet-v2_1 total loss performance with linear graph . . . . .	C18
C.18.	Inception-V3 total loss performance with linear graph . . . . .	C19
C.19.	MnasNet-6.0 total loss performance with linear graph . . . . .	C20
C.20.	ResNet-152 total loss performance with linear graph . . . . .	C21
C.21.	SqueezeNet-V1.1 total loss performance with linear graph . . . . .	C22
C.22.	Swin-v2-b total loss performance with linear graph . . . . .	C23
C.23.	VGG-19_bn total loss performance with linear graph . . . . .	C24
C.24.	ViT-16-b total loss performance with linear graph . . . . .	C25
C.25.	AlexNet histogram pre-trained with 0.0001 learning rate . . . . .	C26
C.26.	ConvNeXt histogram with 0.0001 learning rate . . . . .	C27
C.27.	DenseNet histogram with 0.0001 learning rate . . . . .	C28
C.28.	EfficientNet histogram with 0.0001 learning rate . . . . .	C29
C.29.	EfficientNet-v2_1 histogram with 0.0001 learning rate . . . . .	C30
C.30.	Inception-v3 histogram pre-trained with 0.0001 learning rate . . . . .	C31

*List of Figures*

C.31.MnasNet histogram with 0.0001 learning rate . . . . .	C32
C.32.ResNet histogram with 0.0001 learning rate . . . . .	C33
C.33.SqueezeNet histogram with 0.0001 learning rate and extra layers . . . . .	C34
C.34.Swin-v2_b histogram with 0.0001 learning rate . . . . .	C35
C.35.VGG-19_BN histogram with 0.0001 learning rate . . . . .	C36
C.36.ViT histogram with 0.0001 learning rate . . . . .	C37
C.37.AlexNet histogram with 0.001 learning rate . . . . .	C38
C.38.AlexNet histogram with 0.0001 learning rate . . . . .	C39
C.39.AlexNet histogram with 0.001 learning rate and extra layers . . . . .	C40
C.40.AlexNet histogram with 0.0001 learning rate and extra layers . . . . .	C41
C.41.EfficientNet-B7 histogram with 0.0001 learning rate . . . . .	C42
C.42.EfficientNet-v2_1 histogram with 0.0001 learning rate . . . . .	C43
C.43.EfficientNet histogram with 0.0001 learning rate . . . . .	C44
C.44.Inception-v3 histogram with 0.001 learning rate . . . . .	C45
C.45.Inception-v3 histogram with 0.0001 learning rate. . . . .	C46
C.46.Inception-v3 histogram pre-trained with 0.001 learning rate. . . . .	C47
C.47.Inception-v3 histogram with 0.001 learning rate and extra layers. . . . .	C48
C.48.Inception-v3 histogram pre-trained with 0.0001 learning rate. . . . .	C49
C.49.Inception-v3 histogram with 0.0001 learning rate and extra layers. . . . .	C50
C.50.Inception-v3 histogram pre-trained with 0.00001 learning rate. . . . .	C51
C.51.Inception-v3 histogram pre-trained with 0.0001 learning rate attempt 2. . . . .	C52
C.52.Inception-v3 histogram pre-trained with 0.001 learning rate and extra layers. . . . .	C53
C.53.Inception-v3 histogram pre-trained with 0.0001 learning rate and extra layers. . . . .	C54
C.54.Inception-v3 histogram pre-trained with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epochs. . . . .	C55
C.55.Inception-v3 histogram with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epochs, all with extra layers. . . . .	C56
C.56.SqueezeNet histogram with 0.0001 learning rate. . . . .	C57
C.57.SqueezeNet histogram pre-trained with 0.0001 learning rate. . . . .	C58
C.58.SqueezeNet histogram with 0.0001 learning rate and extra layers. . . . .	C59
C.59.SqueezeNet histogram pre-trained with 0.0001 learning rate and extra layers. . . . .	C60
C.60.VGG-19_BN histogram with 0.0001 learning rate. . . . .	C61
C.61.VGG-19_BN histogram with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epoch. . . . .	C62
C.62.AlexNet histogram pre-trained with 0.0001 learning rate. . . . .	C63
C.63.ConvNeXt histogram with 0.0001 learning rate. . . . .	C64
C.64.DenseNet histogram with 0.0001 learning rate. . . . .	C65
C.65.EfficientNet-B7 histogram with 0.0001 learning rate. . . . .	C66
C.66.EfficientNet-v2_1 histogram with 0.0001 learning rate. . . . .	C67
C.67.Inception-v3 histogram pre-trained with 0.0001 learning rate. . . . .	C68
C.68.MnasNet histogram with 0.0001 learning rate. . . . .	C69
C.69.ResNet-152 histogram with 0.0001 learning rate. . . . .	C70

*List of Figures*

C.70.SqueezeNet histogram with 0.0001 learning rate and extra layers. . . . . C71  
C.71.Swin histogram with 0.0001 learning rate. . . . . C72  
C.72.VGG-19\_BN histogram with 0.0001 learning rate. . . . . C73  
C.73.ViT histogram with 0.0001 learning rate. . . . . C74

# List of Tables

3.1.	Table showing number of networks performing worse than guessing averages.	32
3.2.	Hyperparameters chosen for reference network. . . . .	36
4.1.	Table showing best performing networks on separated parameters with MSE and MAE loss. . . . .	53
4.2.	Table comparing results in ImageNet competition and results achieved in this project. . . . .	58
4.3.	Hyperparameters chosen for best performing neural network. . . . .	58

# 1. Introduction

## 1.1. Background

Out of the incredible amount of energy in the universe, only around 5% of it is directly observable to us with modern technology (Aghanim et al. 2020). The rest is divided into *dark energy* (68%) and *dark matter* (27%). While dark energy remains a complete mystery, dark matter has a few clues despite not being directly observable. The key component to mapping dark matter is that it has mass, and as such exerts gravity. This allows us to see its gravitational effect on visible celestial objects. Through a process called *gravitational lensing* (GL) we can see light from distant galaxies distorted around dark matter. This creates a distorted image of the galaxy. This lensing is grouped into two different arrangements, being either *weak* or *strong* gravitational lensing. In 2016 a mathematical framework was developed (Clarkson 2016) to remove the previous unnatural gap between these. With this, it is now possible to solve strong lensing with geodesic deviation equations instead of ray-tracing/time delay. The Roulette formalism makes it possible to consider both weak and strong lensing together, which is perceived as advantageous in scenarios where both effects appear, such as cluster lens-mass reconstruction. This paved the way for machine learning to help in the field, by making the problem a simple mathematical one for computers. While time-consuming for humans, the calculations for these gravitational lenses can be done by machines in a fraction of a second.

## 1.2. Problem

Manually detecting lensed galaxies and trying to reconstruct the original image is very time consuming. Only a few thousand *strongly lensed* galaxies have been discovered and documented (Huang et al. 2021). The number of *weakly lensed* galaxies however, is much higher, albeit much harder to detect due to their weak nature. If roulette formalism could be used to detect the lensing effects from lens clusters, machine learning could do the rest of the work finding lens parameters. Mapping all of these lenses would directly help to map the dark matter of the universe.



## 1. Introduction

Through machine learning it should theoretically be possible to detect such lensing, and reconstruct an undistorted source image of the galaxy. This project aims to find a stronger neural network driven machine learning network to determine parameters of the lensing effect. This will aid in the continued mission to automatically reconstruct the lens of an image of a real galaxy. Previous research on this topic is sparse, and this project aims to be part of the first fully open source project using roulette formalism to map dark matter, by using machine learning to detect the lens parameters.

### 1.3. Scope of the project

This project is part of a larger set of multidisciplinary projects, and will aim to complete the machine learning part. Our contributions will consist of creating a solid machine learning foundation that can be taken further if necessary. As an interdisciplinary project, our contributions in machine learning and programming will be crucial to the outcome. The goal is to produce a base that can be used for the rest of the project and can be used by everyone from the other disciplines. One of the other goals of this project is being an open source project. This means the code and documentation has to be understandable and clear.

### 1.4. Report structure

The report does not follow the normal structure blueprint given by the university. The typical blueprint report includes chapters for materials and method. Because of the nature of our task it makes more sense to replace these with a solid chapter called Development. This chapter will guide the reader through the processes that took place in a semi-chronological manner. Doing this will make rational behind decisions more apparent, and jumping between sections less necessary. The report also separates the typical Discussion chapter into Discussion and Retrospective. This is done to separate the discussion into two parts, the academical discussion around the development and results in Discussion, and the discussions related to the groups management, learning outcome and prerequisites in Retrospective.

The rest of the report is structured in the following way:

**Chapter 2 - Theory:** Contains an introductory explanation to the concepts relevant to the task. Separated into the cosmological concepts (physics) and the machine learning concepts.

## 1. Introduction

**Chapter 3 - Development:** Contains the experiments done, and some of the results necessary for further experimentation to avoid having to jump between chapters.

**Chapter 4 - Results:** Contains most of the raw results that was not needed to be expanded upon in chapter 3 for further work.

**Chapter 5 - Discussion:** Contains a summary of the results, and the groups opinions on the results and what it means. Also contains suggestions for further work.

**Chapter 6 - Retrospective:** Contains meta-evaluation of the group's structure, prerequisites, and learning outcomes.

**Chapter 7 - Conclusion:** Contains a conclusion/summary of the entire project.

**Appendix:** Contains additional material not necessary to read the report, such as source code and other things used for thesis assessment.

## 2. Theory

This sections will describe the theory behind the two main parts, cosmology and machine learning. It builds on the same mathematical and theoretical framework as the previous project (Ingebritsen et al. 2022). The cosmology theory section brings no new theory from last year, but the machine learning part does.

### 2.1. Cosmology

The following sections on cosmological theory serves as a beginners introduction to the terms and phenomena relevant to this project. The section is only meant for the reader to understand the content of the report, and thus does not go into great detail. If the reader wants a deeper understanding, it is recommended to read up on these topics.

#### **Gravitational lensing**

Gravitational lensing is a phenomena where light traveling past massive objects gets bent around said massive objects. This is due to photons being affected by gravity. When this happens, a distorted image is created for the observer. The object is also observed at a false position due to this(Congdon and Keeton 2018, p. 5). Figure 2.1 below shows different types of lensings.

Gravitational lensing is split into three main classes. We are only concerned with strong lensing and weak lensing in this project. These classes are artificial in nature, due to them being human constructs to make it easier to calculate and categorise, and not natural barriers.

#### **Strong lensing**

Strong lensing is where there is a clear lensing effect present. Examples include a visible Einstein ring or multiple images of the object. The lensing can show a magnifying effect,

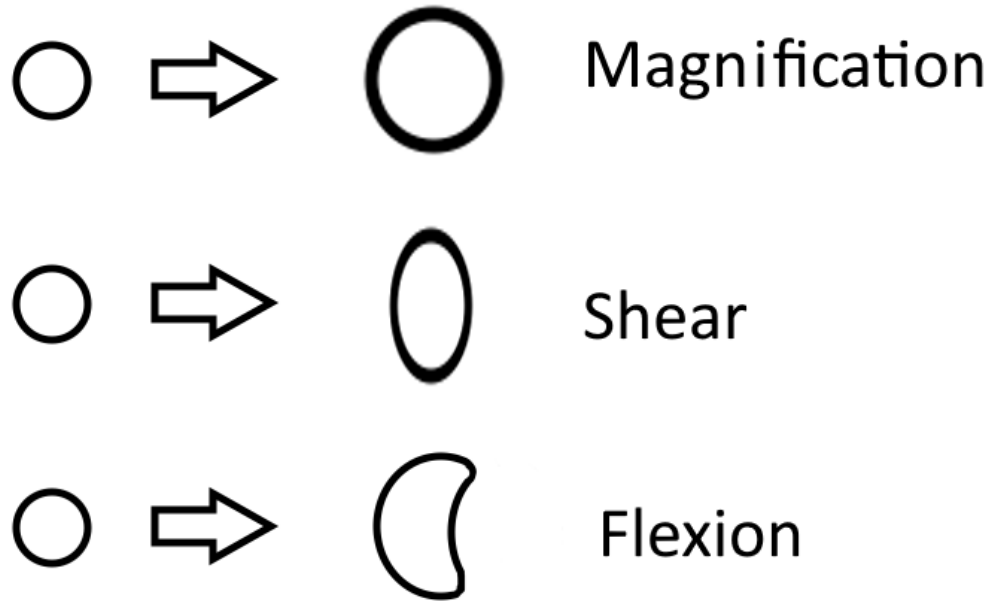


Figure 2.1.: Different types of lensing visualised.

shearing (stretching), and flexion (bending)(Congdon and Keeton 2018, p. 5).

### **Weak lensing**

Weak lensing is a lot weaker than strong lensing. This means that it's hardly detectable. Like strong lensing it can produce a magnifying and shearing effect, but does not produce any flexion(Congdon and Keeton 2018, p. 8). It is instead usually found by analysing multiple sources in a statistical manner.

### **Roulette formalism**

In 2016, a mathematical framework was developed to simulate strong lensing using a weak-lensing approach (Clarkson 2016). This allows weakly lensed objects to appear as strongly lensed, leading to more features being distinguishable. While normally strongly

## 2. Theory

lensed objects are found using ray-tracing and time delay, this new method can simulate this using the geodesic deviation equations of weak lensing.

### **Einstein radius**

An Einstein ring is the result when the observer, lens, and source light all align. This creates a gravitational lensing around the entire lens, forming a visible halo around it (Congdon and Keeton 2018, p. 31). Note that the ring doesn't have to be a full circle, and can be one or more arcs around the lens.

### **Cosmological redshift**

Redshift is a process where photons gradually shift towards longer wave lengths. This can happen for a few reasons. When the wavelength increases, the light moves toward the upper end of the visible light spectrum, therefore becoming more red. This is a known phenomena and can be used to measure distances at cosmological scales (Congdon and Keeton 2018, p. 59).

### **Lenses**

A lens in cosmology is anything with great enough mass to noticeably bend light. While stars and planets have a mass and therefore bends light, its effect is too tiny for us to reliably notice. More often light will be bent noticeably by galaxy clusters. It can also be caused by dark matter. There are different models for gravitational lenses.

The simplest model is point mass. In this model, all the mass of the lens is situated in an infinitely small point, called a singularity. Such singularities are not believed to be possible outside of black holes, so they are merely models for convenience sake (Congdon and Keeton 2018, p. 20).

Another model is the Singular Isothermal Sphere (SIS). In this model, all the mass is evenly distributed all over the lens. Like black holes, the SIS has a fixed mass-to-radius ratio (Remmen 2021), making the mathematics very simple. Below are the formulas for point mass and SIS provided by our supervisor, showing their simplistic nature. Equation 2.1 shows the mass distribution  $\Sigma_{PM}$  of a point mass. Equation 2.2 shows the mass distribution  $\Sigma_{SIS}$  of a SIS. Note that these equations are for two dimensions, as that is essentially the angle we are viewing them from since the celestial sphere is 2-dimensional.

## 2. Theory

$$\Sigma_{PM} \sim R_E \delta_{(x=0)}^1 \quad (2.1)$$

$$\Sigma_{SIS} \sim \frac{R_E}{R} \quad (2.2)$$

## 2.2. Machine learning

This section will explain some of the theoretical terms behind the machine learning part relevant to the project.

### Neural networks

A neural network is a subset of machine learning crucial to deep learning. It has a visible input and output layer which are connected by multiple hidden layers. Each node in the system is connected to another node, and each node has a threshold and weight (Goodfellow, Bengio, and Courville 2016, p. 13).

### Convolutional neural networks

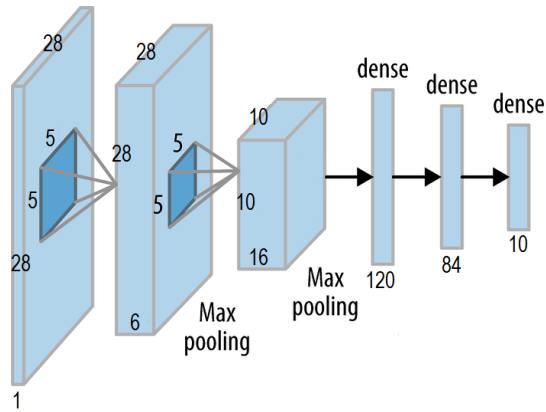
A neural network can have many different qualities used for different purposes. The convolutional neural network excels in dealing with inputs from the real world, be it images or audio signals like speech. While our problem is one of regression, these neural networks deal with classification problems. Our solution to this problem is to attempt training a classification model to deal with the regression problem. This is due to dealing with images with hidden parameters, and classification convolutional networks are specialised to work on images. Some recent attempts at image regression have been made (Ranganathan et al. 2020).

As an example AlexNet is one of multiple CNNs used in this project. The network architecture is similar to LeNet-5 figure 2.2a. LeNet was one of the first convolutional neural networks (Lecun et al. 1998), and AlexNet is an evolution of LeNet. The main difference is that AlexNet is larger and deeper. It consists of eight layers, including five convolutional layers, with three followed by max-pooling layers, three fully connected layers, and a softmax output layer figure 2.2b (Krizhevsky, Sutskever, and Hinton 2012). Figure 2.2c shows network with two extra fully connected layers made for this project.

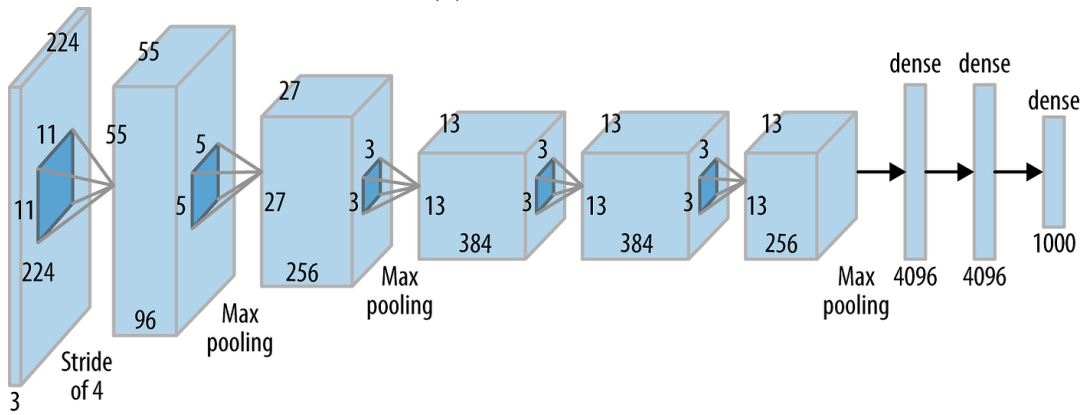
---

<sup>1</sup>This ( $\delta$ ) is the Dirac-Delta function

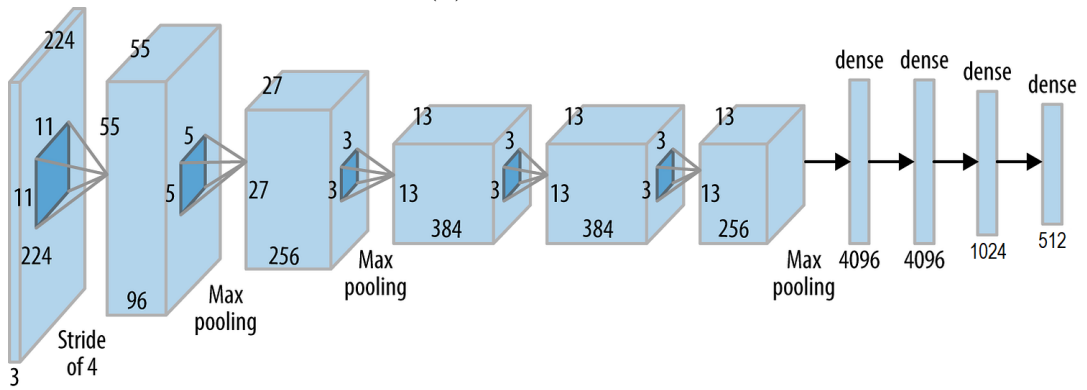
## 2. Theory



(a) LeNet-5 layers



(b) AlexNet layers



(c) AlexNet with added layers

Figure 2.2.: Figures showing LeNet-5(2.2a), AlexNet with default amount of layers (2.2b) and with extra added regression layers (2.2c) (Anwar 2022).

### **Neural architecture search**

Regrettably, the absence of comprehensive guidelines or tutorials for selecting an appropriate network architecture tailored to a specific task poses a significant challenge. In response to this, neural architecture search (NAS) techniques have emerged. This technique automates the process of choosing or designing network architecture by searching for the most optimal architecture that yields the best performance for a given task (Russell and Norvig 2022, p. 821). In this project only one NAS network called MnasNet was tested. It's designed for mobile devices to be efficient, use as little resources as possible and to be easily scalable (Tan et al. 2019). For testing it was scaled up to 6 times it's base size.

### **Neural networks using transformers**

Transformer architecture were originally designed for natural language processing. Instead of using convolutional layers, networks using transformers divide an input image into fixed-size patches and linearly project them into sequence embeddings. These embeddings are then processed by transformer layers, enabling interactions between patches and capturing long-range dependencies (Russell and Norvig 2022, p. 920).

### **Hyperparameter**

In machine learning, a hyperparameter is one of the parameters that directly affects the learning process. While normal parameters are derived by training, hyperparameters are set beforehand and play an important role in how well the machine learning performs. There are different branches of hyperparameters as well, such as model hyperparameters and algorithm hyperparameters. For the purposes of this report, all hyperparameters are grouped together as one group (Goodfellow, Bengio, and Courville 2016, p. 120).

The hyperparameters used in this project are model network (including topology and size), batch size, learning rate, epochs, optimiser, and loss function. These will all impact the training results in various ways, and is the main way to tune the machine learning.

Batch size defines the number of processed samples in a single update or iteration during update. It is an important hyperparameter that affects both the computational efficiency and the performance of the trained model (Goodfellow, Bengio, and Courville 2016, p. 278).

Learning rate represents the step size or the rate at which the model parameters are updated during the learning process. It's essential for achieving optimal convergence



## 2. Theory

and preventing instabilities in the training process (Goodfellow, Bengio, and Courville 2016, p. 82).

Epoch refers to a single pass or iteration through the entire training data set during the training phase of a model. By completing one epoch, the model has seen and learned from all the available training samples. The number of epochs determines the total number of times the learning algorithm will work through the entire data set. An appropriate number of epochs is crucial to ensure the model has sufficient exposure to the data for learning without overfitting or underfitting (Goodfellow, Bengio, and Courville 2016, p. 244).

Optimizer is a component of training process that aims to minimize the objective function or loss by adjusting the model's parameters. It determines how the model's parameters are updated during the iterative optimization process (Goodfellow, Bengio, and Courville 2016, p. 82).

Loss or cost function quantifies the discrepancy between the predicted output of a model and the true output. It represents the measure of how well the model is performing on a given task (Goodfellow, Bengio, and Courville 2016, p. 82). For regression task two loss functions are most popular. Mean square error (equation 2.4), and mean absolute error (equation 2.3), where  $l$  is loss for one image,  $n$  is number of parameters,  $x$  is ground truth and  $y$  is estimate value by network. To get total loss all image losses are summed together.

$$l_{MAE} = \frac{1}{n} \sum_{i=1}^n |x_i - y_i| \quad (2.3)$$

$$l_{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - y_i)^2 \quad (2.4)$$

### Transfer learning

Transfer learning is a method where network parameters(weights and biases) that were saved after training on one task are used as a starting point for training on another task. It is an idea that trained parameters of the network on one task will translate to better learning performance on another task.(Russell and Norvig 2022, p. 832)

### **Previous work with machine learning on gravitational lensing**

In the combined field of machine learning and gravitational lensing, there has been a multitude of recent studies and articles. There seem to be great leaps forward, especially when it comes to using CNNs to detect strong lensing. Multiple projects have created their own CNNs to detect strong lensing (Rezaei et al. 2022)(Wilde et al. 2022), while others have attempted to use machine learning to detect quasars (Khramtsov, Vladislav et al. 2019). Magro et al. 2021 has also compiled a list and comparison for similar projects. (Morgan et al. 2021) has created a similar open source simulation software package for strong gravitational lensing.

## 3. Development

This chapter will contain most of the work done in this project. It will contain most events done in chronological order. This will allow the reader to understand the reasoning and intentions behind what work was done. Each subsection will build on the experiences learned and talked about in previous subsections.

### 3.1. Testing of previous work

Since this project is a continuation of a bachelor project from last year, code and tools already existed. A working simulator to visualise gravitational lensing, as well as image generation software for generating images of lensing was already in place. Before any further work on the preexisting code was done, a test was performed. This test would not modify anything, and was only to see how well the current code performed. This could then later serve as a point of comparison, to determine the effectiveness of future networks. This section also includes problems found during testing, as well as the solutions to the problems.

#### Generating images and training

Generating images could be done by a preexisting program, and only required changing some variables to fit our needs. The group had access to the Idun HPC cluster, but a check was made to determine if this was necessary for either image creation or training. With a Ryzen 7 5800H GPU, generating 1000 images with elliptical sources and SIS lenses took around 3 minutes. Changing it to point mass lenses and spherical sources made this only take 13 seconds. Since spherical and elliptical took roughly the same time, this showed that making images with point mass was much faster, and would be the base of testing for now. With these parameters, 100 000 images could be generated in 28 minutes, making it viable to generate images without the help of Idun.

The old code was using a slightly modified Inception-v3 architecture. The network had modified input layer channels and output layers. The input layer was changed from 3 to

### 3. Development

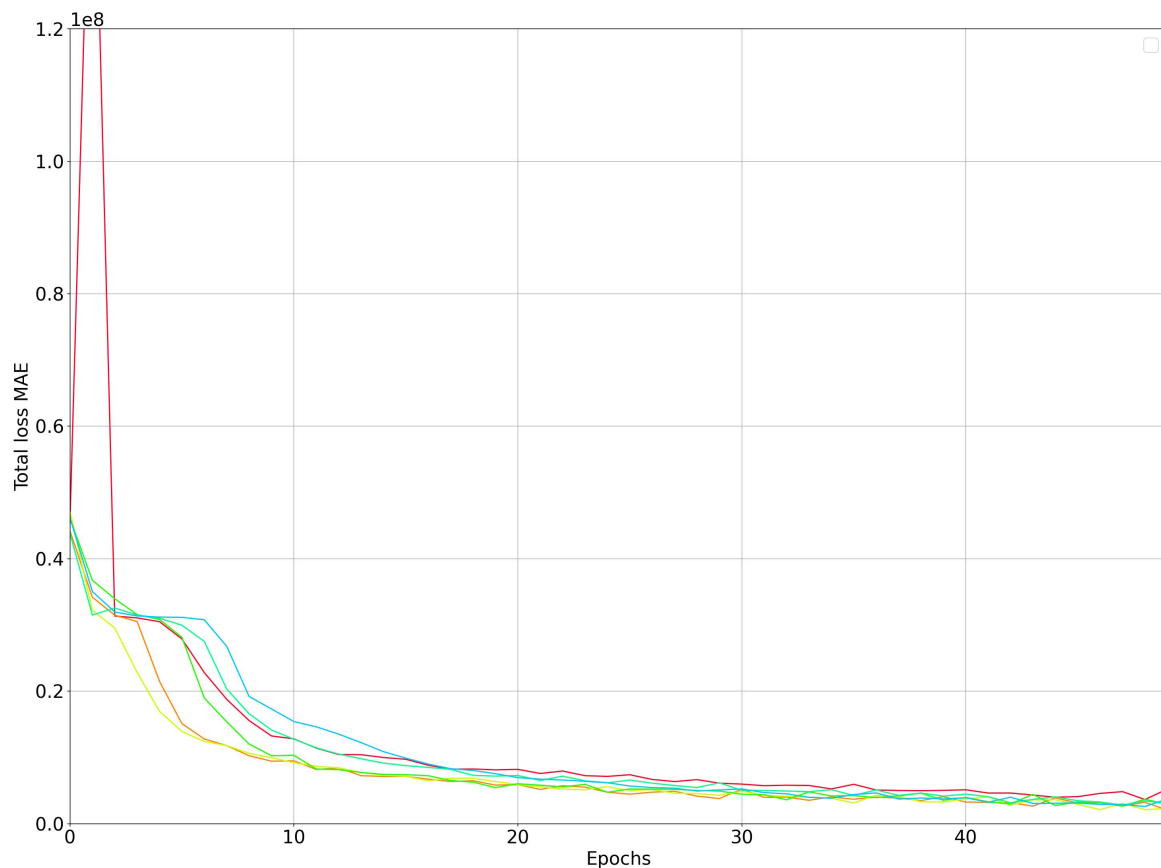


Figure 3.1.: Total loss per epoch of initial code from last year.

1 due to the images not needing RGB colours. The output layers were changed based on how many parameters were being estimated, in this case to 4. Using a desktop computer, a test run on the old machine learning code was run on 10 000 images for 50 epochs with a batch size of 32 and learning rate of 0.001. From the existing codes hyperparameters, only the amount of epochs were changed from 12 to 50. The parameters guessed were  $r$ ,  $\phi$ ,  $R_E$  and  $\sigma_1$ . This took around 2 hours of training. Afterwards, the test was repeated 9 more times to check for inconsistencies and patterns. Figure 3.1 shows the result of all 10 tests. The y-axis shows total loss for each entire epoch. This result was saved for later, to make it was possible to compare the performance at the start and the end of the project. After the test it was decided that using Idun was not necessary for now, as personal desktop computers were fast enough.

#### Problems with the old image generation

The existing code for image generation had a couple of pitfalls necessary to avoid. These were problems that if not taken into account, a good machine learning model could not

### 3. Development

be generated. These were all found either through what made sense theoretically, or accidentally through normal use. They were then checked manually, to see if the problem occurred, documented, and then fixed.

#### **Different parameters generating same images**

The generator creates images with different parameters. If some of the parameters are certain values, the images can appear to look exactly the same as an image created with a different set of parameters. For example, if the angle is 0 or 360 degrees, the images should look exactly the same. If  $\chi$  and  $R_E$  have the same ratio, the images should also look exactly the same, given all other parameters are identical. If an ellipsoid source is rotated 90 degrees one way and source size and secondary size have opposite values, this can also generate seemingly identical images. A script to randomly flip around half the images were made. The script made 55 with normal parameters, and 45 images with theta 90 degrees more and  $\sigma_1$  and  $\sigma_2$  flipped. Out of these 100 images, 41 were identical, while all the rest were the smallest error possible (up to one pixel shifted to the side). This was considered good enough to show the images were the same. The error was not visible with the naked eye.

Ellipsoid sources have sizes measured with standard deviation along major and minor axis ("Source size" and "Secondary size" in the simulator). Figure 3.2 visualises these standard deviations. In the figure, the standard deviation for x-axis is 60, and 30 for y-axis. This equates to the source being twice as wide as tall from our perspective. Since the major and minor standard deviation of an ellipsoid in essence are its magnitude in x and y direction, it stands to reason that by interchanging the parameters, the image has visually rotated 90 degrees to one side. By applying a 90 degree counterclockwise rotation to the source ( $\varphi$ ), the ellipsoid will appear to have returned to its original position. It could also have rotated by 180 degrees if  $\varphi$  is 90 degrees clockwise. However, this would run the risk of looking identical as well if the radius is small.

To prove that this is the case, a python script that analyses differences between images was created. Two test sets of images were created. One test set contained images where  $\chi$  and  $R_E$  was the same ratio in all of them ( $\chi$  was randomized, then  $R_E$  was set with a ratio based on  $\chi$ ). For both the sets, the rest of the parameters were set to the same values every time. To account for rounding errors,  $R_E$  was set to be double of  $\chi$ .

The python script showed they were not exact replicas. Another function showed the difference between the images to be 0.03 absolute value pixel-by-pixel difference. Image difference analysis showed some images appeared to be shifted to the side despite all parameters except  $\chi$  and  $R_E$  being the same. This was unexpected. Comparing a set of 100 images with these parameters showed only 14 were identical, with the others having a small difference up to 2.12 absolute value.

### 3. Development

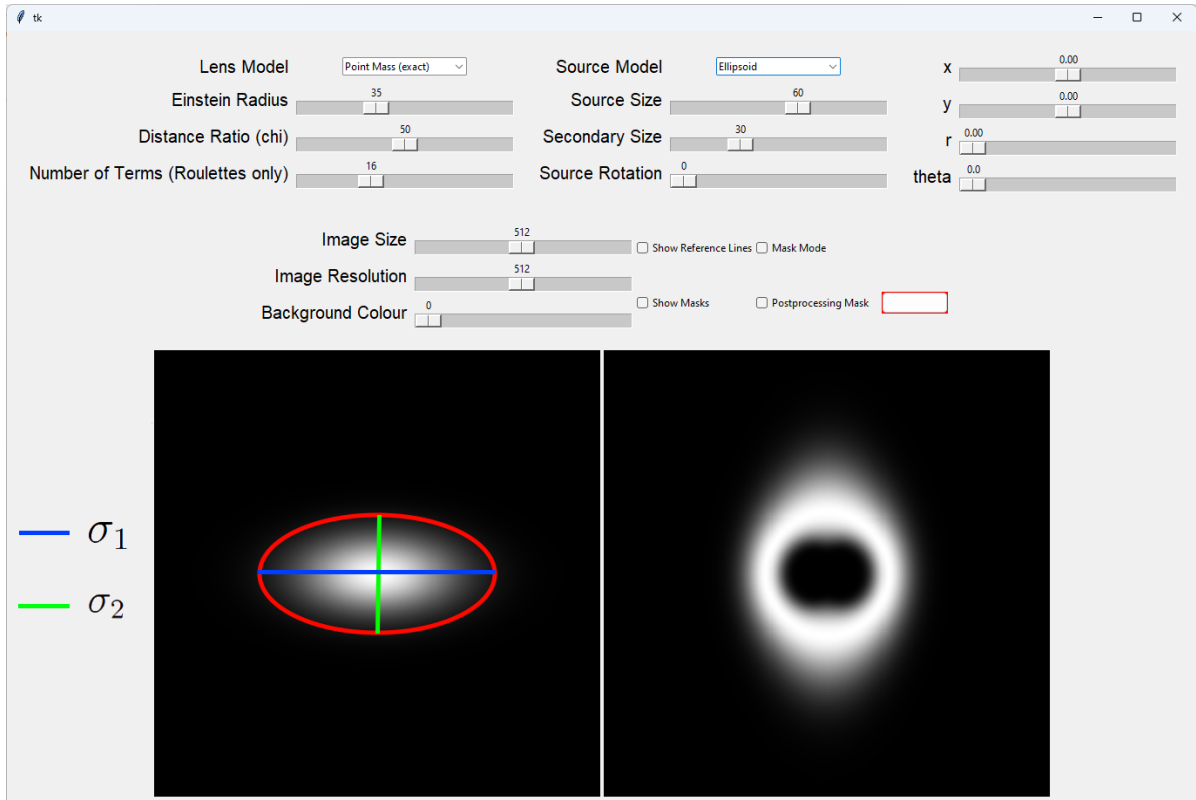


Figure 3.2.: Standard deviation along major (blue) and minor (green) axis of elliptical source, shown in simulator software.

### 3. Development

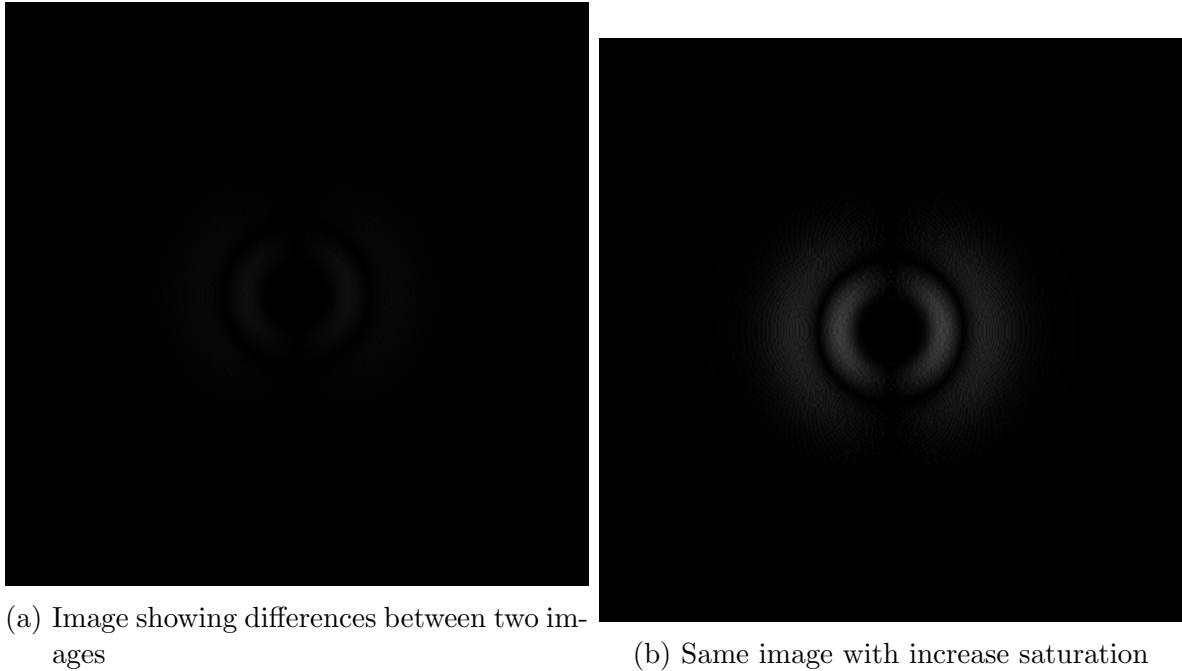


Figure 3.3.: Subfigures showing the difference between two images with an absolute difference of 1.003.

When tested on 100 images created with all parameters being the same, the images were all deemed identical by the same program. This led us to believe that the images created by the simulator are inaccurate. Seeing manually how small the difference was, the error was deemed as irrelevant to the machine learning, as it was at most 2 pixels to the side ( $\pm 1$  pixel from center). For reference, two images where one is shifted one pixel to the side produced an absolute difference of 1.0003. The python script can be found in Appendix H.

Figure 3.3 above shows the difference in the images when absolute difference is 1.0003. Figure 3.3a shows the difference as it is. Figure 3.3b is the same but with exposure set to 200%. This shows that it is difficult for humans to see the difference. No pattern was found in the set of different images. The difference seemed random and independent of parameters. The difference would not be the same every time. Given the small absolute value difference, along with the even smaller visual difference, this issue was deemed not problematic enough to spend more time on.

### 3. Development

#### Verifying if a fixed ratio between $\chi$ and $R_E$ produce the same images

Equation 3.1 shows the light distribution provided by our cosmology supervisor. It is a mapping from the source-plane coordinates  $(x', y')$  to the lens-plane coordinates  $(r, \phi)$ , and shows how a light point from the source plane is distorted in the lens plane. From the formula, it is evident that different values of  $\chi$  and  $R_E$  should not give the same results, even if the ratio remains the same. As clearly visible in the equation,  $\frac{R_E}{\chi} \cdot R_E$  and  $\frac{r}{\chi}$  does not remain the same if both values are changed even if the ratio remains.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{r}{\chi} \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} + \frac{\frac{R_E}{\chi} \cdot R_E}{(\vec{r} + \vec{R})^2} \begin{pmatrix} \frac{r}{R} \cos\phi \\ -\sin\phi \end{pmatrix} \quad (3.1)$$

Another experiment was set up. The hypothesis was that when  $\chi$  approaches 0 or 100, these difference would show themselves. Since all image generation previously had  $\chi$  limited between 30 and 70 (with it being a fixed number most of the time), perhaps the difference was not noticeable for these values. Three different tests would be performed to check this hypothesis. All the tests would randomise all values for every other parameter once, then use those values for all images.  $\chi$  and  $R_E$  would remain the same fixed ratio, but the numeric values would chance between images. The difference this time was that the experiment was divided into three groups. One where  $\chi$  is between 6 and 30, another where  $\chi$  is between 30 and 70 (like the original test), and a last test where  $\chi$  is between 70 and 96. This would test both extremes more thoroughly. Should the values of  $\chi$  come too close to the source or observer plane (too close to 0 or 100), the lensing effect would become too weak.

These tests were all performed with the same software as the test earlier, which checks for absolute value pixel-by-pixel image disparity (Root mean square (RMS) difference). After creating 100 images for each test, they were all checked against all other images. For 100 images this means a total of 4950 comparisons.

The test results show the identical pairings out of the 4950 pairings. It also shows the largest RMS difference achieved between the images. The following amount of pairings were completely identical:

- $\chi$  from 6 to 30: 683 out of 4950 (13.8%) (largest RMS difference: 0.04)
- $\chi$  from 30 to 70: 548 out of 4950 (11.1%) (largest RMS difference: 0.03)
- $\chi$  from 70 to 96: 92 out of 4950 (1.9%) (largest RMS difference: 1.12)

From this it is clear that when  $\chi$  approaches high values, images are less likely to be identical. Whether or not the higher difference causes a realistic problem to the machine



### 3. Development

learning is still up for debate, but with the largest absolute difference being 1.12 it seems unlikely for now.

Another set of tests were set up to further confirm this hypothesis, as the implications from such a discovery could have further consequences along the way. These tests would further limit the scope of  $\chi$  to between 92 and 98 and between 1 and 5. Even though the previous test showed little difference when  $\chi$  was between 6 and 30, the last test limited this further to between 1 and 5. This is to further prove that when  $\chi$  approaches its limits, the images show a noticeable difference. Testing other limits were made difficult due to  $\chi$  and  $R_E$  always needing to be whole numbers. This makes it impossible to test ranges like 97 to 99, since there is no ratio that gives  $R_E$  different whole numbers. This was fixed when  $\chi$  was between 1 and 5 by multiplying to find  $R_E$  instead of dividing.

After the new test, results showed that out of 4950 pairings, the following amount of pairings were completely identical:

- $\chi$  from 1 to 5: 877 out of 4950 (17.7%) (largest difference: 0.83)
- $\chi$  from 92 to 98: 1201 out of 4950 (24.3%) (largest difference: 0.02)

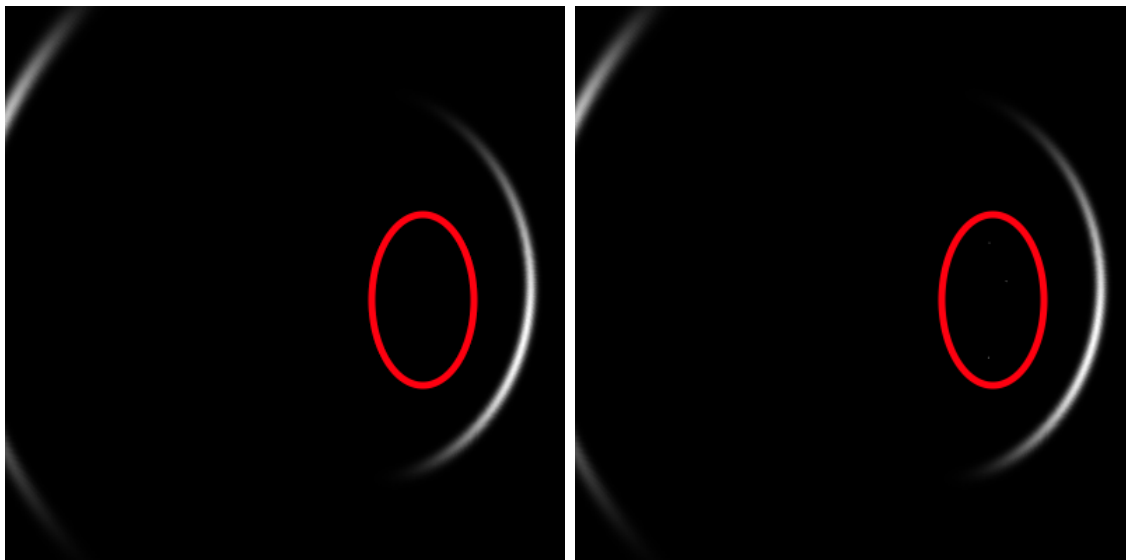
Intrigued by these results, a closer look on the images was done manually. This revealed a critical fault in the image generation. Figure 3.4b shows artifacts in the one of created images where  $\chi$  was between 1 and 5. The image is supposed to look exactly like figure 3.4a since they have all the same parameters, but has three barely visible white dots. This is present in around half the images in different magnitudes. What is interesting is that in every single image with these artifacts, all artifacts are located in the exact same three spots in the image that figure 3.4b shows. These artifacts can also vary from all three spots having artifacts to none being present, even when all the parameters are exact replicas.

It was decided that as long as image generation could be done in such a way that it could practically eliminate the chances of such artifacts, the problem would not be pursued any further. The question of whether or not  $\chi$  and  $R_E$  are directly linked in the way we thought previously is still open. For now we believe the reason the images are not identical to be because of the artifacts when generating images.

## 3.2. AlexNet estimating same output for different inputs

A problem occurred when training on the default AlexNet network. The network topology was unmodified and the problem was found manually during an exploratory test of AlexNet. The network would predict the same result for every instance for an entire

### 3. Development



(a) Figure showing no signs of artifacts      (b) Figure showing signs of artifacts (three white dots) that are not supposed to be present

Figure 3.4.: Figures showing the supposed same image, but with figure 3.4b having three barely visible dots not supposed to be there.

epoch. This number might change for the next epoch, but the pattern of predicting one number for the entire epoch continued. The numbers were converging towards the average ground truth values of training data set for each parameter. A hypothesis was created that this had to do with batch sizes and small data sets. To investigate this further an experiment was set up to find out the cause and a solution. 5 different solutions were tested to see what would help, all consisting of changing hyperparameters.

#### Testing different hyperparameters

A baseline was created by running AlexNet 100 times for 50 epochs with batch sizes of 10 and 100. This would show the unmodified networks result, and could be compared to when assessing a new solution. Adam was chosen as the optimizer with a learning rate of 0.001, and MSE loss function. This test-training was done on 2000 images. As shown in figure 3.5a, it's clear that the problem occurs more often with higher batch size. The problem is clearly evident in both cases.

The next part of the experiment consisted of tweaking different hyperparameters. All further experiments were run for 10 epochs. This was done to cut on run time, since most of the changes between experiments can be extrapolated from the first 10 epochs, and the problem would always occur within 10 epochs. The attempted hyperparameter

### 3. Development

changes were:

- Having a bigger data set (10 000 images)
- Changing optimiser from Adam to RMSprop
- Adding more layers to the network
- Changing the loss function from MSE to MAE
- Changing learning rate (both increase and decrease)
- Running every attempt with a batch size of 10 and 100

Figures in 3.5 and 3.6 below show total loss per epoch of each run. Dotted black line represents the total loss achieved by guessing averages of training data as networks output. The red plotted lines represent batch size of 100 and blue line represent batch size of 10. There are in total 100 test done per batch size for every different setting, for a total of 200 lines per graph. Running 100 times each would show if batch size affects the result on a broad scale. Table 3.1 shows the amount of tests over the dotted line at the end of training. One could also check how many tests were within a certain percentage threshold from the dotted line, but the tests done were enough. This was due to the fact that if any line was above guessing the average it would be considered a problem. Since any test performing worse than guessing the average after 10 epochs is considered a problem, any solution will have to completely remove all such cases.

All subfigures in figure 3.5 and figure 3.6 show a clear distinction between batch sizes. Lower batch size is training faster and have a lower chance of hanging up on a local minimum. However it is important to mention that it's not immune to getting stuck. In the first figure 3.5a after 10 epoch 20% and 72% of 100 models with batch size 10 and 100 respectively are over the dotted line. This goes down to 19% for both batch sizes after 50 epochs. Although even when the same amount of models are over the dotted line for both batch sizes, it seems clear that batch size of 10 is faster at getting out of local minimum. Moving forward it was decided from this to use a low batch size of 10.

Focusing on figure 3.5 four different changes were tried with no satisfactory results. Increasing the amount of images for training by factor of 5 in figure 3.5b increased loss scale 5 times. This was exactly as expected, and meant no improvement. After 10 epochs, 17% and 23% of models were left over the total loss limit with batch sizes of 10 and 100 respectively. This affected models with batch size of 100 the most, decreasing number of models over the limit from 72% to 23%. Models with batch size of 10 saw a small change from 20% to 17%.

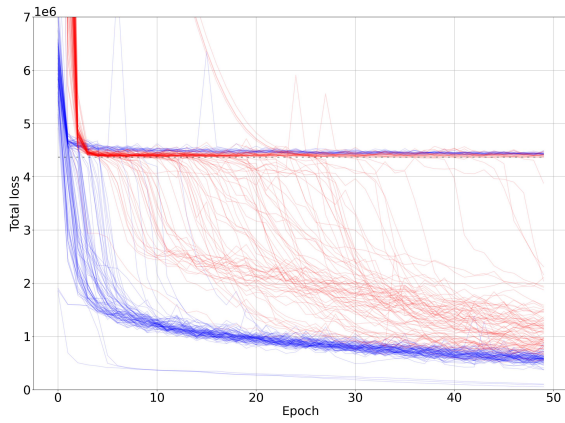
Figure 3.5c gave optimal results using the RMSprop optimiser. Just two out of all

### 3. Development

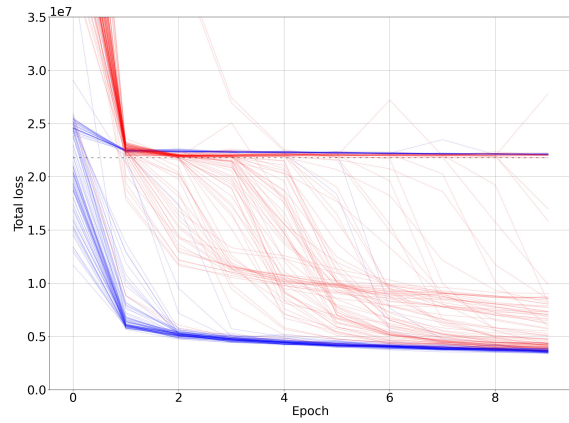
<b>Changes</b>	<b>Batch size</b>	<b># neural networks performing worse than guessing average</b>
Default (10 epochs)	10	20
	100	72
Default (50 epochs)	10	19
	100	19
Bigger data set	10	17
	100	23
Learning rate 0.005	10	30
	100	51
RMSprop	10	0
	100	2
Extra layers	10	15
	100	69
MAE	10	36
	100	49
Learning rate 0.0001	10	0
	100	0
Learning rate 0.00001	10	0
	100	0
Learning rate 0.000001	10	0
	100	0

Table 3.1.: Table showing number of networks performing worse than guessing averages.

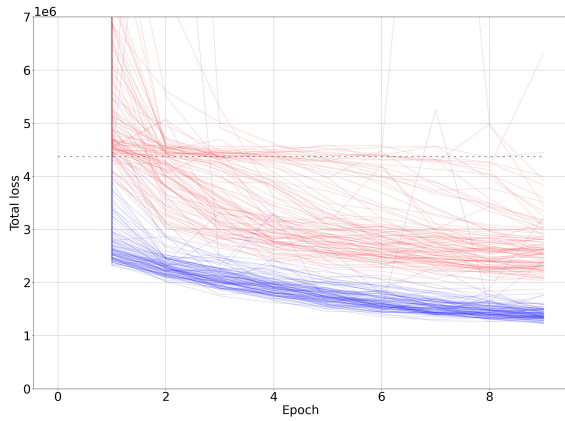
### 3. Development



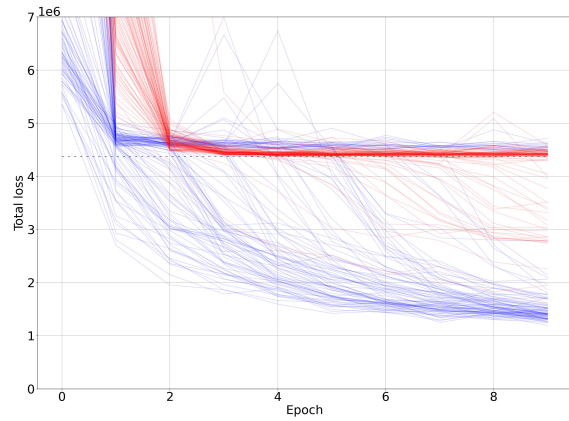
(a) Default parameters with 50 epochs



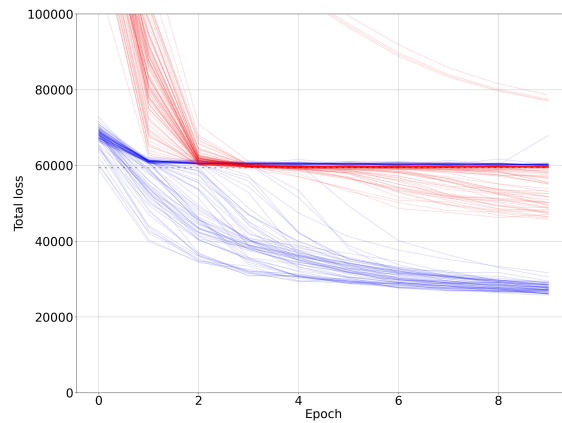
(b) Bigger data set (10000 images)



(c) RMSprop optimizer



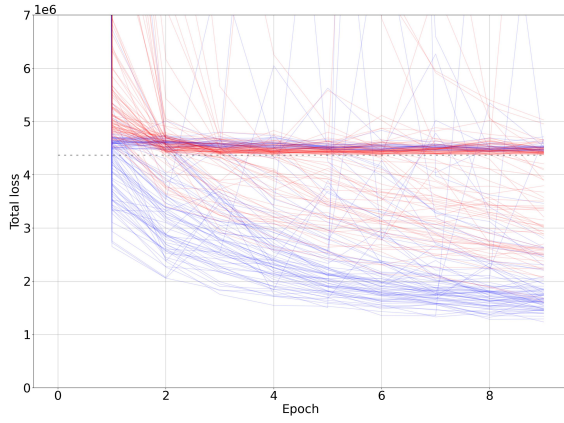
(d) Network with extra layers



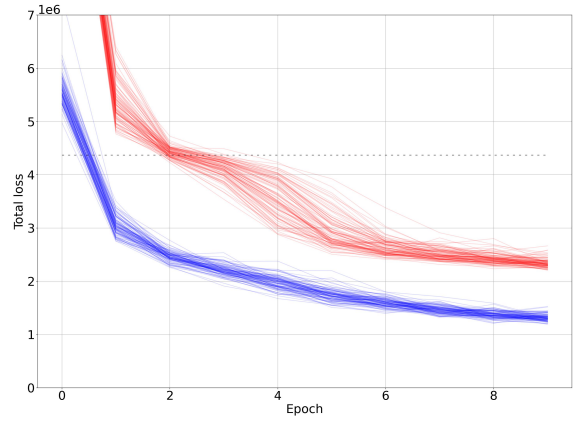
(e) Loss function of MAE

Figure 3.5.: Figures showing total loss with different hyperparameters.

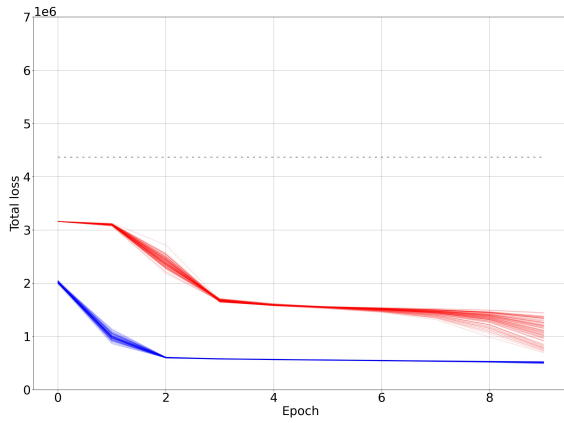
### 3. Development



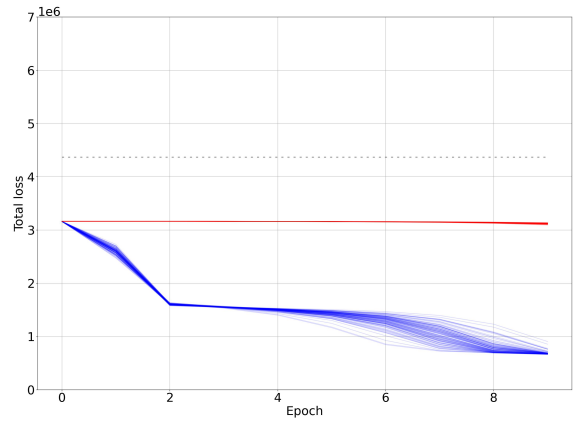
(a) Learning rate of 0.005



(b) Learning rate of 0.0001



(c) Learning rate of 0.00001



(d) Learning rate of 0.000001

Figure 3.6.: Figures showing total loss with different learning rates.

### 3. Development

networks had total loss over dotted line after 10 epochs. But there is still some visible pattern of network getting stuck on local minimum. Although this showed promising results, network is not close to following a satisfactory learning curve.

Addition of extra layers to the network was also attempted. An additional two fully connected layers were added at the end of the sequence, as shown in figure 2.2c. Figure 3.5d shows this did not result in any meaningful improvements either. The performance is close to a network without any additional layers.

Changing loss function in figure 3.5e makes calculated total loss around 10 times smaller. The effect on networks were mixed. Batch size of 100 were affected positively while batch size of 10 were affected negatively. Like all the other tests until now this was not good enough as a solution.

Lastly figure 3.6 shows testing of different learning rates. First test consisted increasing the learning rate to see if that produces a different result. Increasing learning rate by 5 times to 0.005 in figure 3.6a made the network loss unstable, jumping up and down from one epoch to another. This might help for a network to jump out of local minimum, but results do not reflect that consensus.

Since increasing learning rate led to worse results, it was decided to try lowering it. Figure 3.6b shows the effect of lowering learning rate 10 times to  $10^{-4}$ . The result were very positive, and none of the networks got stuck on local minimums. The problem with network estimating the same values for every input disappeared completely. This was unexpected since expectations where that lower learning rate have tendency to getting stuck on local minimum.

After seeing positive results, learning rate was further lowered 10 times more to  $10^{-5}$ . This change in learning rate made networks more consistent as shown in figure 3.6c. After the first epoch total loss for all networks no mater the batch size did not exceed total loss line for guessing averages. Batch size of 10 converge after 3 epochs at the same loss. And as before bigger batch sizes perform worse but seem to start catching up towards the end.

Lastly lowering learning rate to  $10^{-6}$  gave worse results, shown in figure 3.6d. Networks with batch size of 100 are not able to learn anything after first epoch, and the idea of using a learning rate lower than  $10^{-5}$  was scrapped.

A total of 8 different tests were performed. Out of those, 3 were made after the discovery that lower learning rate had a great impact. After this the unmistakable value of a proper learning rate was learned. The test showed that optimal learning rate should probably be somewhere between  $5 * 10^{-4}$  and  $10^{-5}$  as a good rule to follow. It was decided that going forward, the learning rate default should be changed to  $10^{-4}$ . This was due to that learning rate having a decently optimal learning curve, as well as never getting stuck in

any local minimum. If needed, different learning rates around this new reference could be tested.

### 3.3. Creating a reference network

It was decided to create a reference network as a way to determine the performance of new networks. This reference would be a constant to measure the new networks against, to objectively determine which yielded the greatest improvements. As a reference network AlexNet was chosen. Today image classification is a popular field in machine learning. A lot of research has been done on different neural networks just for this task. Our hypothesis is that the better neural networks get at classification the worse it get at performing other tasks like regression. The reasoning behind choosing AlexNet because it's quite a simple network that used to be arguably the best convolutional neural network in 2012, among other things winning ImageNet challenge that same year (Krizhevsky, Sutskever, and Hinton 2012). It was also the network model we had spent some time debugging in section 3.2, so we possessed some knowledge already. Table 3.2 shows the properties and parameters for the reference network.

Network model	Optimiser	Learning rate	Loss function	Batch size	Epochs
AlexNet	Adam	0.001	MSE	10	50

Table 3.2.: Hyperparameters chosen for reference network.

Training was done on data set with 100 000 images with a few different parameters. Trying RMSprop and Adam optimizers, MSE and MAE loss functions and changing estimated coordinate parameters. In total this creates eight experiments. All experiments were run for 50 epochs with a batch size of 10 and learning rate of 0.001. The astute reader will notice this learning rate is not the more optimal one found in chapter 3.2. This was due to simultaneous work the tasks and the solution not having been found at this time. Due to there being around a 20% chance of getting stuck, every test was manually vetted for this. Only one of the eight tests got stuck. That test was promptly restarted and completed without problems.

In figure 3.7 eight histogram plots show distribution of parameter errors made by network estimating test data set of 10 000 images. Top 4 figures are of networks with polar coordinates ( $\phi$  is measured in angles). Bottom figures are with cartesian coordinates. First two columns used MSE loss and last two columns use MAE loss. The first and third columns use Adam as optimiser, while second and fourth columns use RMSprop.

These results show that MAE create a overall more precise network while not sacrificing performance on outliers. Overall networks trained with MAE perform better, when measured with MSE or MAE loss. Networks estimating cartesian coordinates generally



### 3. Development

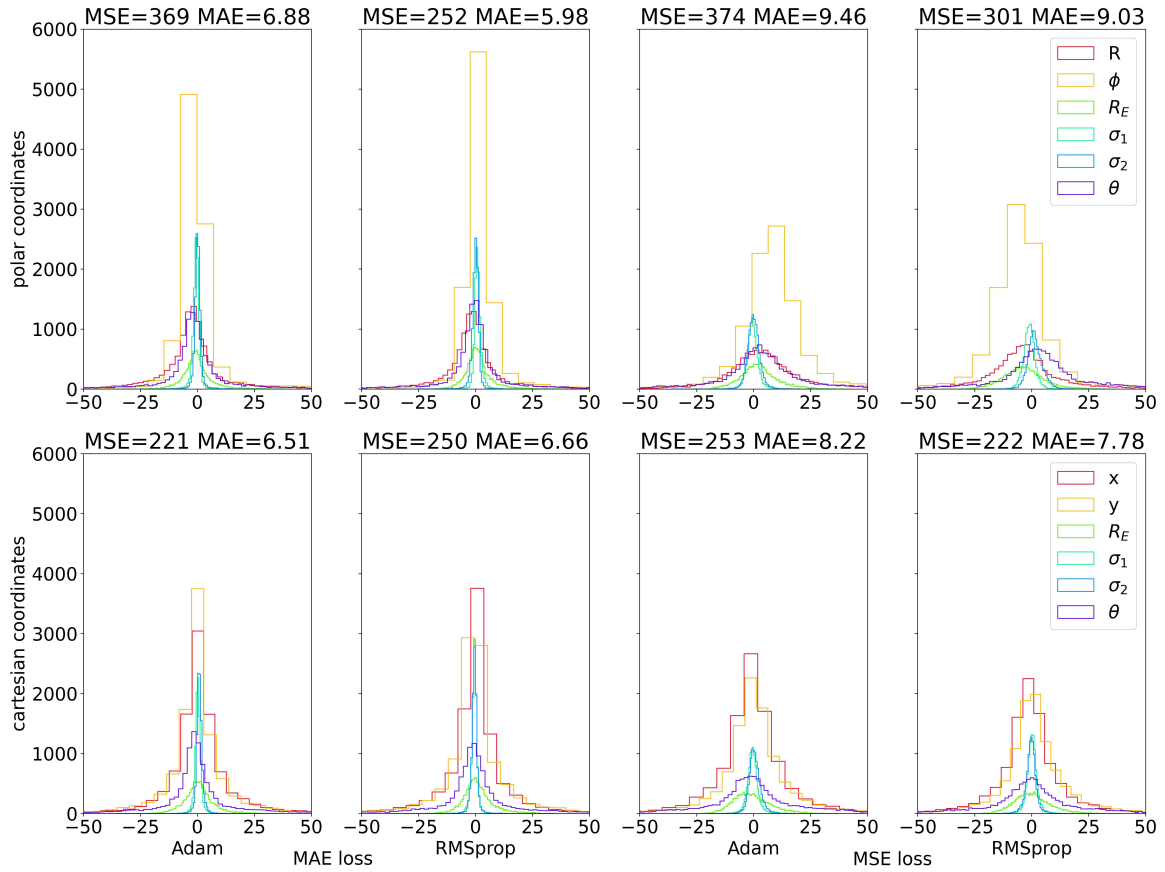


Figure 3.7.: Histogram showing deviation from ground truth in different settings.

perform better, having lower overall MAE and MSE. Trying different optimizer did not show a meaningful differences between there performance.

## 3.4. Hyperparameter optimisation

In this section work was started on optimising the hyperparameters for the machine learning. Using the hyperparameters network model, optimiser, learning rate, loss function, batch size, and epochs, work was started to create the best possible machine learning program. How good the networks performed are judged against the reference network and whichever has the lowest loss.

### Choosing the optimizer

There are a lot of different optimizers for machine learning. But as found in this paper (Li et al. 2022, p 7008) all of them tend to converge at the end of training. The paper notes that although requiring more computation, Adam performed the best. Adam was chosen as the optimizer due to this and being simple to implement. It also boasts a small memory requirement and being computationally efficient (Kingma and Ba 2017). The main strengths of Adam lies in it's ability to converge quickly on data with sparse features. Since this project generates images with very sparse features, Adam should perform well. Adams weakness according to Li et al. is likeliness to converge to a sharp minimum. This problem was already solved in chapter 3.2 by adjusting learning rate. When this was accounted for, Adam performed better across all tests, as shown in figure 3.8 below. Adam was then tested against RMSprop to make sure the hypothesis was correct. Both networks were ran with the same weight and biases on polar and cartesian coordinates. Adam did not require any noticeable increase in computation time. These results were enough to decide that Adam was the best optimiser moving forward.

### Loss function choice

There are two main loss functions for regression to choose from. These are MAE(Mean Absolute Error) and MSE(Mean Square Error). Data set and outliers in it is the biggest factor in deciding what loss function should be used. By calculating absolute error MAE is more robust to outliers in training data set, while MSE put importance on outliers. On the other hand, MSE produces a differentiable result that enables control over the update rate. In contrast, the result obtained from MAE is non-differentiable, making it impossible to determine the update speed during optimization (Li et al. 2022). For our

### 3. Development

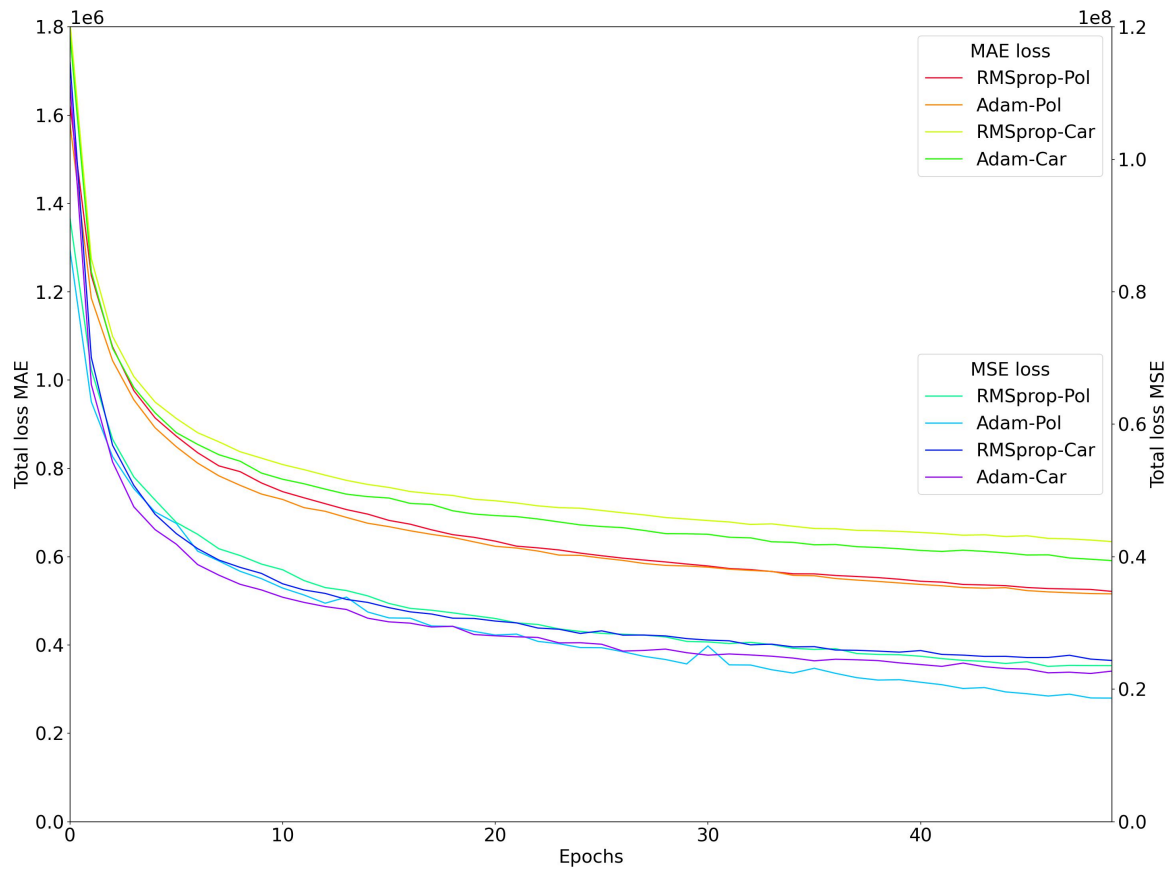
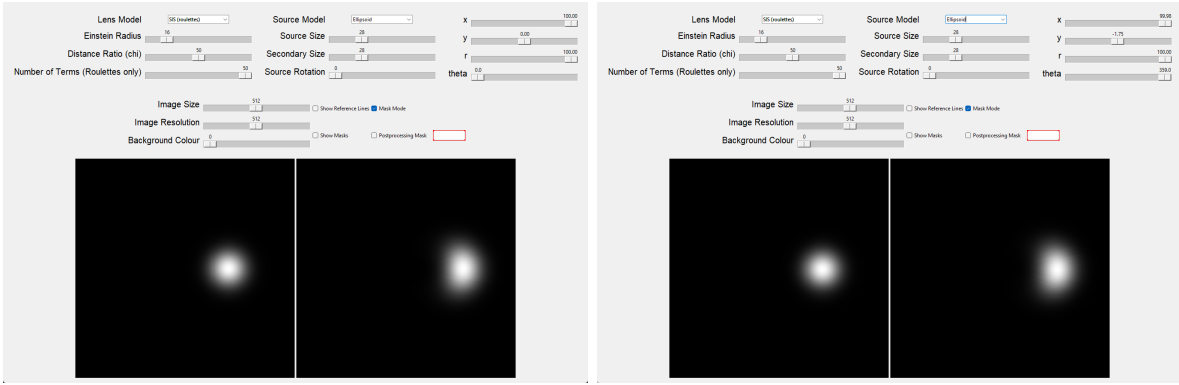


Figure 3.8.: Total loss over 50 epochs using Adam and RMSprop.

### 3. Development



(a) Images showing 0 degrees rotation

(b) Images showing 359 degrees rotation

Figure 3.9.: Images showing the lack of visual disparity between large numerical disparity.

synthetic data set where outliers are few, MAE looks more attractive. And as mentioned previously in chapter 3.3, MAE performed better in figure 3.7.

#### Choosing which parameters to predict

A choice had to be made between predicting cartesian or polar coordinates. Originally it was planned to run the test on polar coordinates, but this caused suboptimal results due to  $\phi$  having a periodic behaviour. This can cause the loss function algorithm to have problems distinguishing between very low and very high values of  $\phi$ . It also creates false negatives when around these periodic checkpoints. For example, if the ground truth is 0 degrees and the network guesses 359 degrees. The network would see this as an error of 359 degrees, while in reality it was only 1 degree. Figure 3.9 shows how similar these two rotations look while being as far apart as possible numerically. Cartesian coordinates avoids this problem but might perform worse. Comparison between the two are difficult to directly measure as they have different ranges which leads to different loss magnitudes. A decision to run two experiments, one with polar and one with cartesian coordinates was done. This was to see if the difference between the coordinate systems caused any noticeable problems.

Figure 3.7 earlier showed that there was not a noticeable decrease in quality using either cartesian or polar coordinates. The MAE losses are roughly the same between the coordinate methods, while the MSE losses varies more. While cartesian coordinates do better in all but 1 of the 16 results, it is not a huge difference. Since a reference network needed to choose one of the two options, the choice was made to use cartesian coordinates in tests going forward. When looking at MSE loss function results of polar coordinates, the resulting loss was as much as 67% higher (369 vs 221) than cartesian. Cartesian coordinates offers additional versatility by being able to perform well with

### 3. Development

both MAE and MSE.

Another problem was predicting  $R_E$  and  $\chi$ . When these have the same ratio between them, they are indistinguishable from one another. Instead, an attempt was made to guess the ratio between the two interconnected parameters. This proved more useful when comparing total loss and was therefore deemed better. Later this was changed to guessing  $R_E$  while  $\chi$  was a constant value. This is essentially the same, but makes it more intuitive and makes data generation easier. It should be possible to estimate  $R_E$  after finding the distance to the lens and source. Finding this distance is not possible on our synthetic data. It is possible to do with real sources manually, and as such could prove useful later.

Other parameters relevant to estimate were  $\sigma_1, \sigma_2$ , and  $\theta$ , all properties of the source. The source model chosen was set to elliptical. Elliptical sources are not necessarily more realistic or common, but were thought to be more interesting than spherical ones. This was merely a matter of preference, and as such we went for the recommended source model.

In the end the parameters were grouped into two sets for testing and training. One test tried to predict polar coordinates  $(r, \phi)$  and the other cartesian  $(x, y)$ . The other parameters predicted for both groups were  $R_E, \sigma_1, \sigma_2, \theta$ .

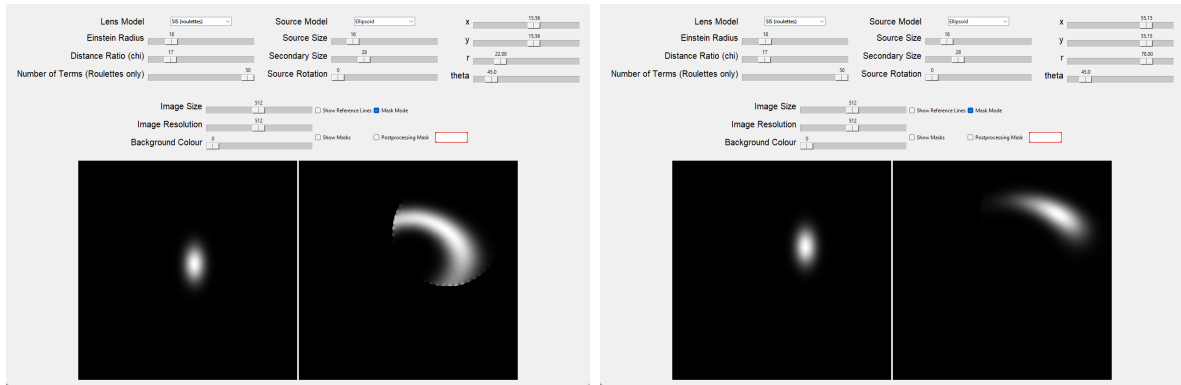
Another thing to consider was whether or not to limit the outputs. This way the neural network would be limited in the range of outputs, allowing only results the image generation could make. This would speed up the start phase of the machine learning where the network is guessing multiple orders of magnitude off. While it would have its uses, it was decided against on the basis of scalability. Fitting a range to our synthetic data would require more work later when changing to real data. Since the end goal of this interdisciplinary project is to use the machine learning on real data we don't know enough about yet, this idea was scrapped.

#### Creating a data set

At the start of this project testing and training was done with a data set of elliptical sources and pointmass lens. This was because elliptical source are more interesting than spherical source which are more researched. Although pointmass lens model is precise mathematically, it is an idealisation that does not occur in nature. As a starting point however it looked good enough. Python code below shows the original settings used to create the images.

```
def getline(idx, chi=0, nterms=16):  
    if 0 == chi:
```

### 3. Development



(a) Artifacts showing around object

(b) Increased radius removing artifact in 3.10a

Figure 3.10.: Simulator showing how difference in radius can impact visual artifacts.

```

chi = randint(30,70)

# Source
sigma = randint(1,60)
sigma2 = randint(1,40)
theta = randint(0,179)

# Lens
einsteinR = randint(10,50)

# Polar Source Co-ordinates
phi = randint(0,359)
R = randint(einsteinR,100)

# Cartesian Co-ordinates
x = R*np.cos(np.pi*phi/180)
y = R*np.sin(np.pi*phi/180)

```

After a while data sets were pivoted towards using SIS roulette as lenses. This was done because SIS, despite not being possible, is more realistic to nature. There is much less work done with SIS compared to point mass. However because the model uses roulette formalism it is limited by what it can produce. The center of the roulette is precise, but the further out from center the less precise the model becomes. This creates artifacts around the model. Figure 3.10 shows how allowing certain combinations of parameters can cause visual artifacts. This was dealt with by fine tuning parameters of data generation. Below is a code snippet showing the modified code to generate images without these artifacts.

```

maxR = int(imgsize/5)
minER = 5
def getline(idx,chi=50,nterms=50):
    if 0 == chi:

```

### 3. Development

```

    chi = randint(30,70)

# Lens
einsteinR = randint(minER, 50)

# Polar Source Co-ordinates
phi = randint(0,359)
R = randint(26, maxR)

si = int(0.4 - einsteinR*0.05 + R*0.282)
if si < minER:
    einsteinR = randint(minER, 50)
    R = randint(28, maxR)
    si = int(0.4 - einsteinR*0.05 + R*0.282)

# Source
sigma = randint(minER, si)
sigma2 = randint(minER, si)
theta = randint(0, 179)

# Cartesian Co-ordinates
x = R*np.cos(np.pi*phi/180)
y = R*np.sin(np.pi*phi/180)

```

Equation 3.4 was found by experimenting in the simulator. Having  $R_E$  as 50 constantly while changing distance from mass to source, maximum  $\sigma_i$  ("si" in python code) was found that wouldn't stretch out into artifacts. After collecting data points, equation 3.3 was calculated that would cover the data. The same method was used to find equation 3.2 for  $R_E$  with a constant distance finding the biggest  $\sigma_i$ . This helped, but made a new problem of sources being possibly too small. Since increasing source size would lead to artifacts, increasing the maximum distance from center of mass to source was the only option. For this to be possible bigger images had to be made. The images were increased to a resolution of 1000x1000. Afterwards images are centered on the light distribution center, and then cropped into 400x400 resolution images. This was the same resolution the data generation made previously. The C++ code had to be edited for all of this to be achieved, see Appendix B. Luckily, the groups limited C++ knowledge was enough. In the end the biggest possible sources could barely fit into the final centered image, which was the goal. This meant that for all the edge cases tested, the data generation held up.

$$f_1 = 2.5 - R_E * 0.05 \quad (3.2)$$

$$f_2 = -2.1 + R * 0.282 \quad (3.3)$$

$$\sigma_i = f_1 + f_2 = 0.4 - E_R * 0.05 + r * 0.282 \quad (3.4)$$

Another criteria is that the galaxy can not be too small either. This is because we need

### 3. Development

enough details after cropping to make out distinguishing details. Setting a minimum size of the source allows for enough details, making sure the machine learning algorithm has a realistic chance. This was done by checking if  $\sigma_i$  was lower than the lowest possible  $R_E$ . If it was, the data would be remade with new parameters that could not generate such a small source.

After deciding all the above, one data set for training and one for testing was created, with 100 000 and 10 000 images respectively. These images are the base for all further testing and results. Running the original test of the old code on this image set generated almost identical loss per sample, proving it was not any worse than the old data generation while generating more complex and unique images.

## 3.5. Achieving the best results

At this point in the project work shifted toward achieving the best possible result. In this case that means minimising loss. Using all the knowledge accumulated so far, tests would be performed on multiple neural networks with different settings and modifications. Since previous research on this topic is very sparse, excessive testing of different networks architectures had to be done. The best results would be decided on which had the lowest MAE and MSE loss.

### Hyperparameters

First the hyperparameters had to be decided upon. After finding the importance of learning rate, a new experiment was set up. To be sure that our finding transfer to other network architectures, a test with three different learning rates were done on the Inception-v3 architecture as well. Figure 3.11a shows a histogram of the deviations from ground truth with MSE and MAE loss with different learning rates. Results show the best performance is achieved with learning rate of  $10^{-4}$ . This learning rate has the smallest amount of spread on all estimated parameters in the histograms and the least loss. This fits what was found earlier when encountering the estimation problem with AlexNet in chapter 3.2. However the perfect learning rate may lie somewhere between  $10^{-4}$  and  $10^{-5}$ . For consistency between tests and due to not having enough time to test different learning rates for all the networks, a decision to use  $10^{-4}$  was made.

Since this chapter is to achieve the best result, the different networks should standardise some things. Henceforth, all tests ran will use the same default hyperparameters unless stated otherwise. All networks will be tested on the same data sets of 100 000 training images and 10 000 test images. Batch size will be set to 10, learning rate to 0.0001, epochs



### 3. Development

to 100, and the network model will be unmodified and if it's possible the pretrained weights will be used. When some hyperparameters are changed, the ones not mentioned can be assumed to be the default. All of the networks have randomised weights and biases, with the exception of the pre-trained ones.

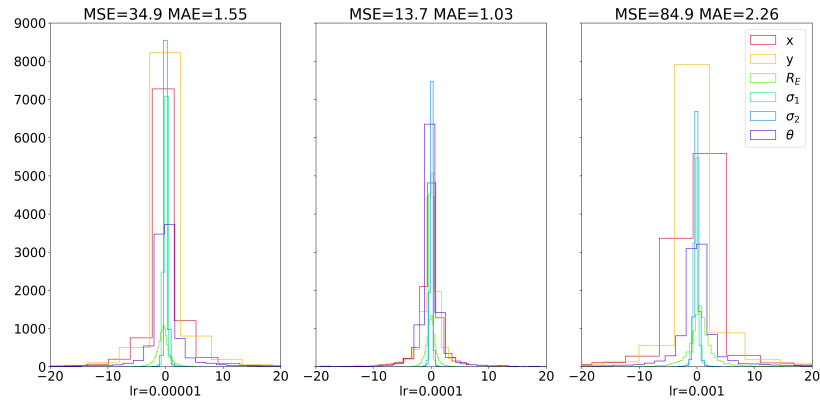
#### Network settings and modifications

The last tests will try transfer learning and changing the last layer with two new fully connected layers. The new layers reduce the size from 1024x1024x1 to 512x512x1, then again to 256x256x1, similar to the added layers to AlexNet in figure 2.2c. Figure 3.11b shows the results of these tests. The testing suggest that networks with pretrained weights or added layers perform better than randomly initiated weights on default network. Adding both changes to the network however made it perform worse than a network with only transfer learning. It still performed better than not being pretrained and with no extra layers. It can be difficult to visually see the difference in the histograms, but the MSE and MAE loss on top of the figures shows there is a difference.

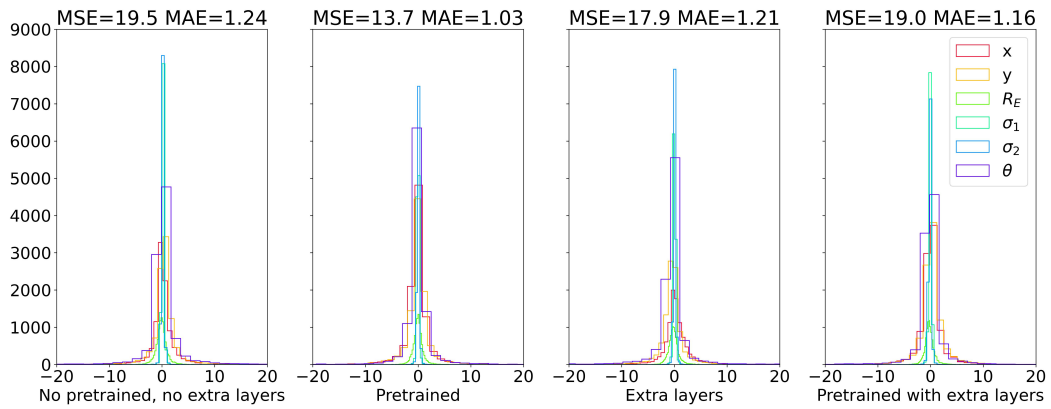
When the search for the best network began, the need for more computing power arose. The group was given access to NTNUs HPC cluster "Idun". This is a HPC cluster that consists of 80 high end GPUs and hundreds of cores. This allowed us to not only run networks with higher requirements, but also to run multiple instances at the same time. Idun runs slurm scripts that had to be learned. This took a couple of days to completely learn, but was not problematic as the time was quickly saved. Idun allowed us to run multiple jobs at once. At most 8 jobs were run simultaneously, allowing massive time saves.

In the end it was decided to take the best performing networks and trained them for 100 epoch more with lower learning rate of  $10^{-5}$ . Lower learning rate was chosen because network is already close to it's best performance and lower learning rate might give it a chance to improve closer to local optimum. Since inception-v3 and VGG-19\_BN where very close to each other in performance both where additionally trained. Figures 3.12a and 3.12c show that both networks performed identically, with VGG being more stable. Extra training of these networks in figures 3.12b and 3.12d gave marginally better results. At the beginning of training the total loss spikes up. This is because of Adam optimisers moving averages witch needs a few epochs to settle down. After this spike the total loss for training data set improved throughout the whole training period. Total loss on test set settles down in the firs half of the graph.

### 3. Development



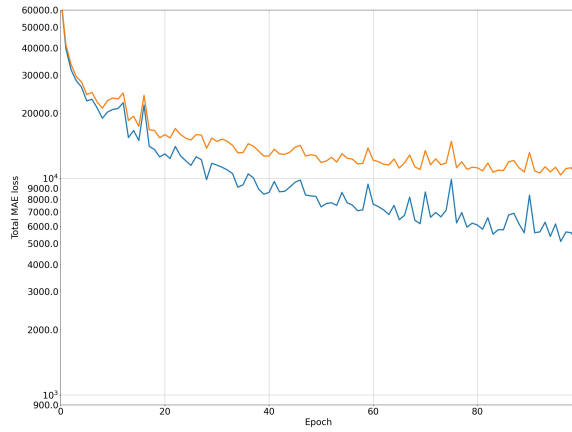
(a) Testing different learning rates



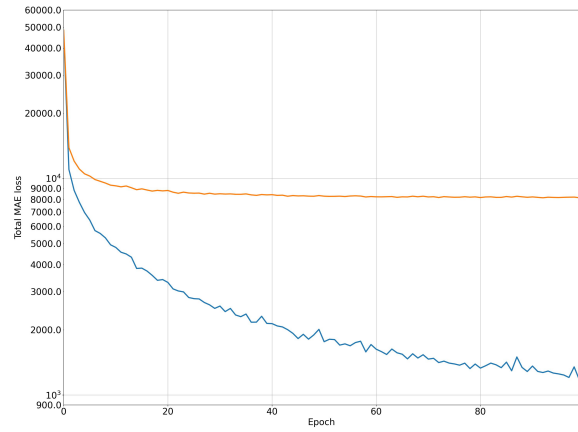
(b) Testing transfer learning and adding extra layers

Figure 3.11.: Histogram showing the deviation from ground truth with Inception-v3 architecture with different modifications.

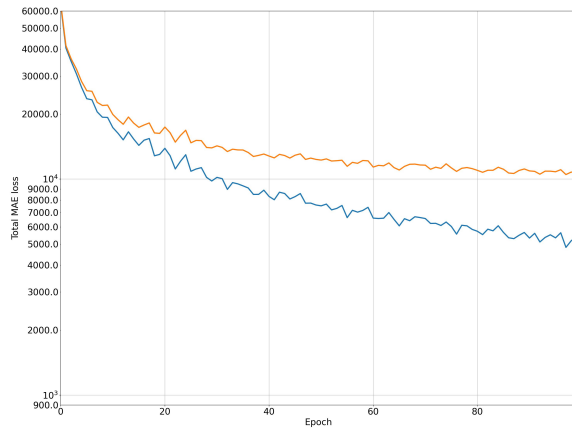
### 3. Development



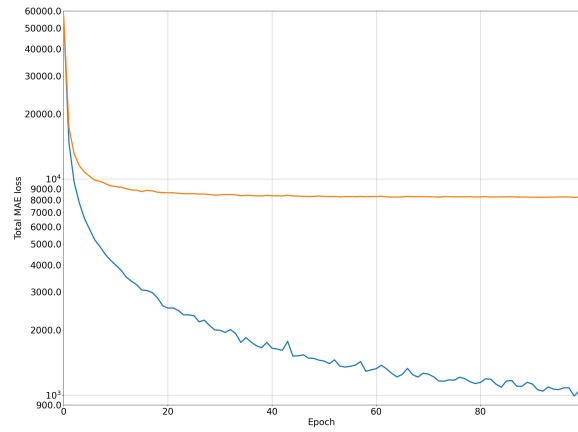
(a) Inception-v3 100 epochs



(b) Inception-v3- 100+100 epochs



(c) VGG-19-bn 100 epochs



(d) VGG-19-bn 100+100 epochs

Figure 3.12.: Graphs showing the total MAE loss per epoch on test and training data set.

## 4. Results

In this chapter the results from chapter 3.5 will be presented alongside comparisons between them, and comparison to the reference network. Results here are considered the final results of the project. Figure 4.1 shows all the best total loss results from each network in one figure. Figure 4.2 shows these same networks as histograms, showing their deviation from the correct values. Due to the large amount of results and data, this chapter will only contain any result that will be discussed further in chapter 5. All results gathered from all tests can be found in Appendix C.

### 4.1. Results deemed interesting

This section includes results from individual network models that will be discussed in chapter 5. It will group together results based on what is being discussed.

Figure 4.3 shows the best results for both VGG-19\_BN and Inception-v3 when limited to 100 epochs.

Figure 4.4 shows the results from different configurations of Inception-v3 and VGG-19. The lowest MSE and MAE loss for both were achieved with transfer learning (pretrained) and no added layers. VGG-19\_BN achieved better MSE loss (10.8 vs 12.6), but the two achieved the same MAE loss (0.82).

Figure 4.5 shows the best results from VGG-19\_BN and Inception-v3 from figure 4.3. In this figure, the x-axis is limited to +/-5.

Figure 4.6 shows the total loss per epoch and histogram performance of MnasNet. This network never escaped local minimum.

Figure 4.7 shows the best results achieved by the vision transformer networks.

Table 4.1 shows the MSE and MAE loss for each parameter for each of the 4 networks with lowest loss overall.  $\theta$  has the highest loss for all networks, while  $\sigma_1$  and  $\sigma_2$  performs the best.

## 4. Results

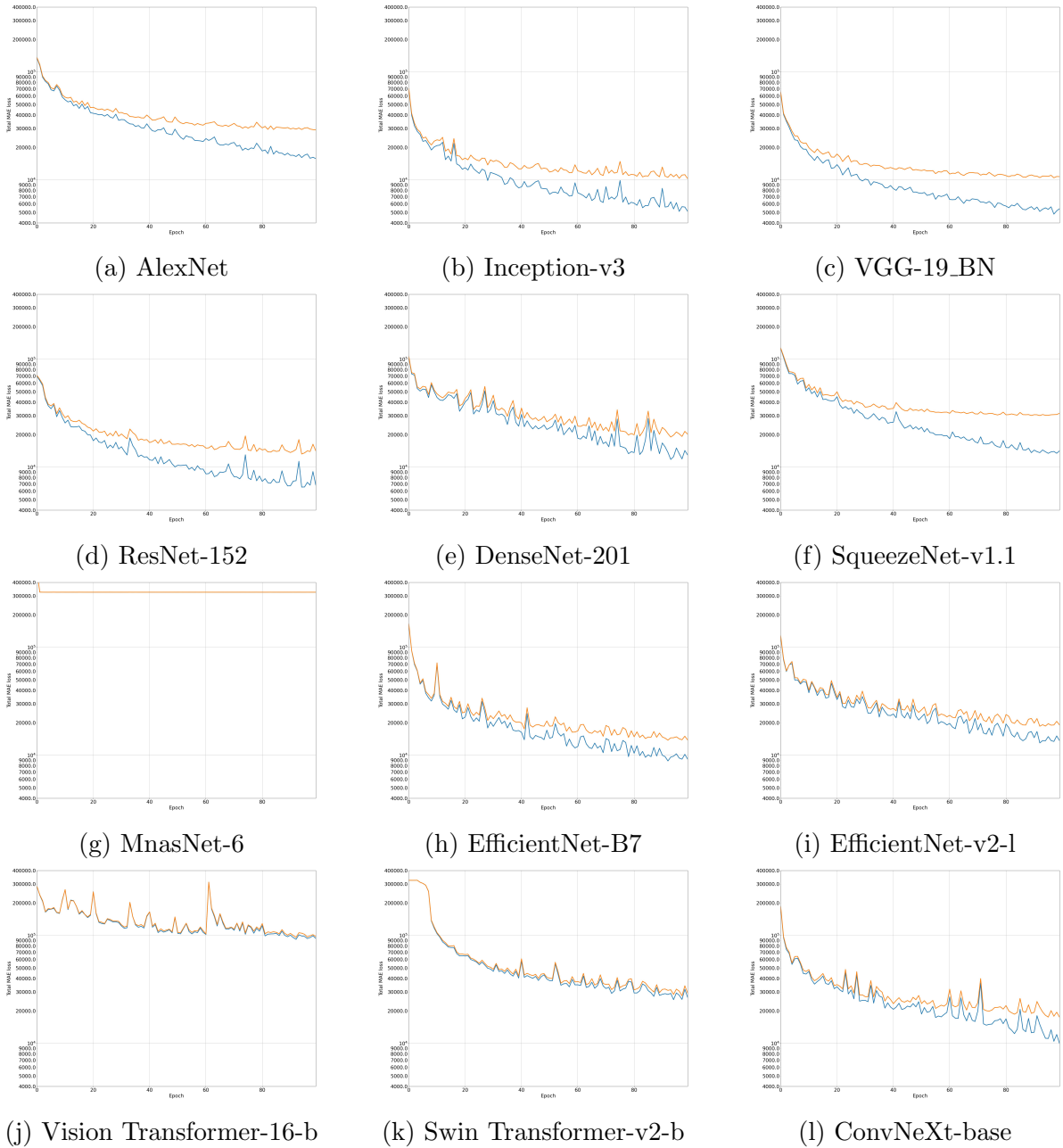


Figure 4.1.: Figures showing total loss (logarithmic) with different network architectures. Subfigures only show the best result achieved with that network model. The blue line represent training data set and yellow line represents testing data set.

## 4. Results

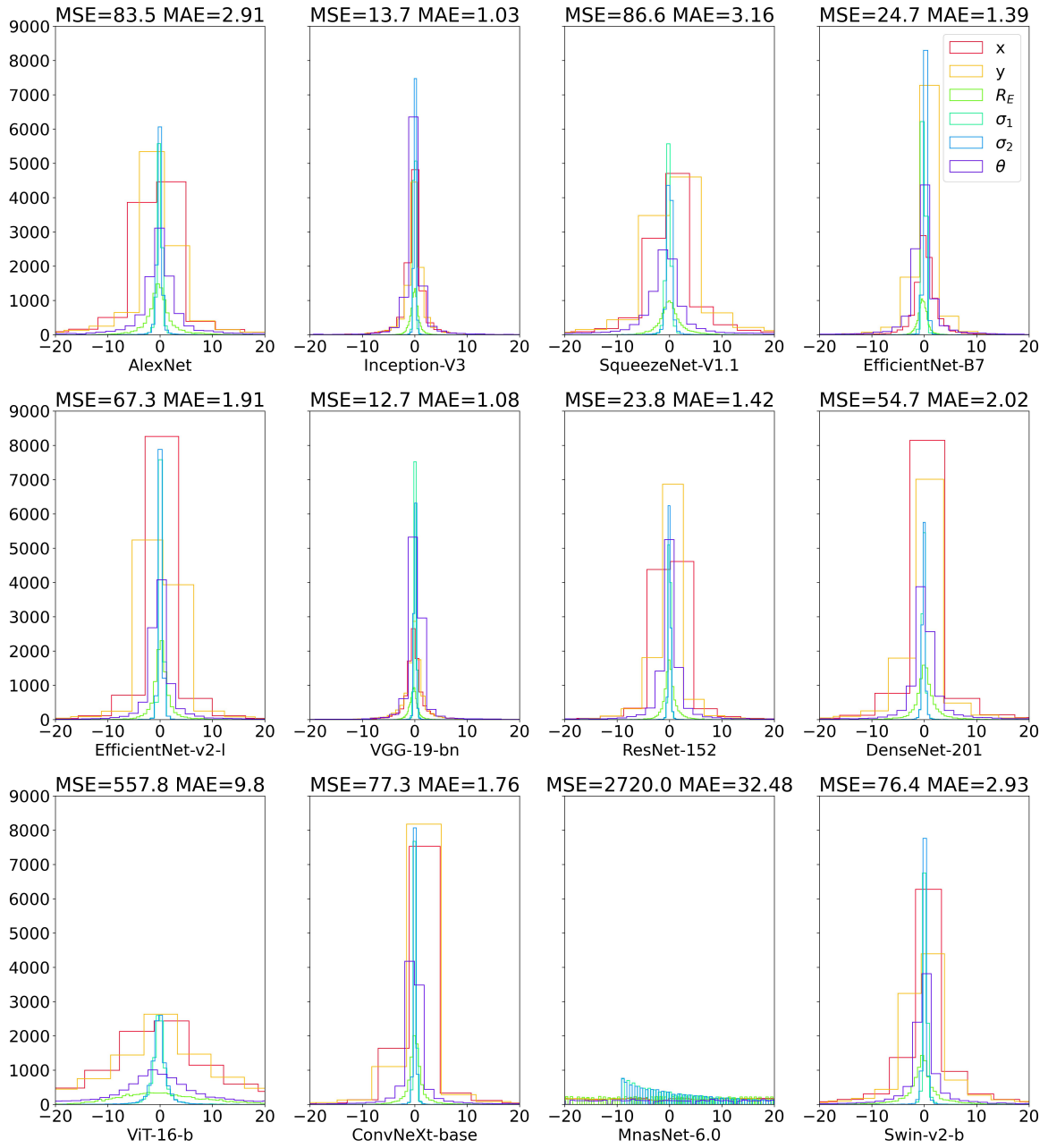
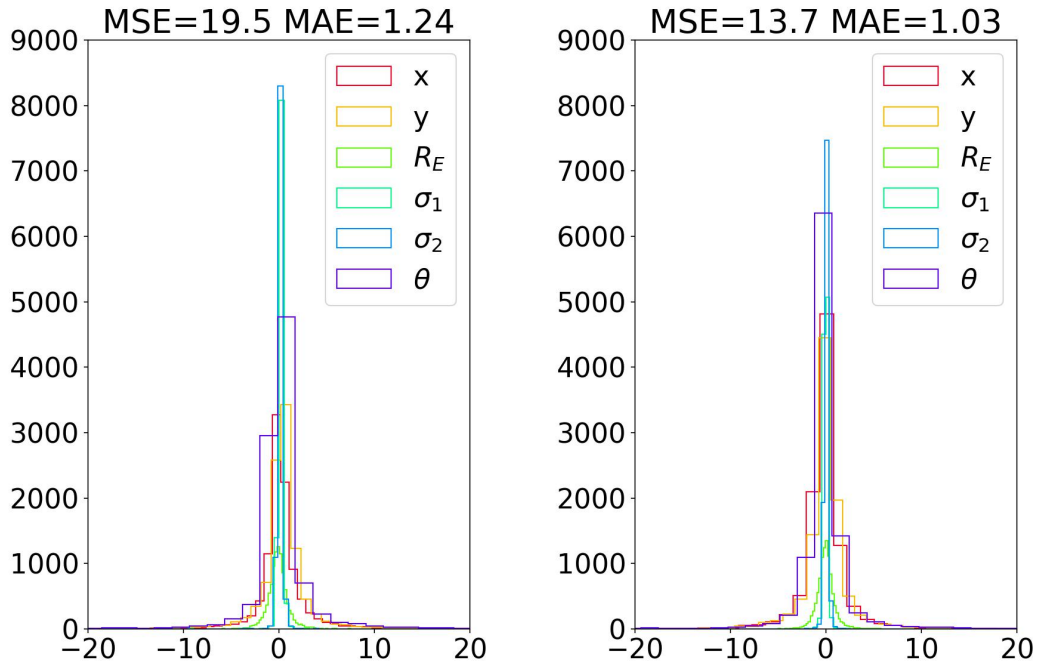
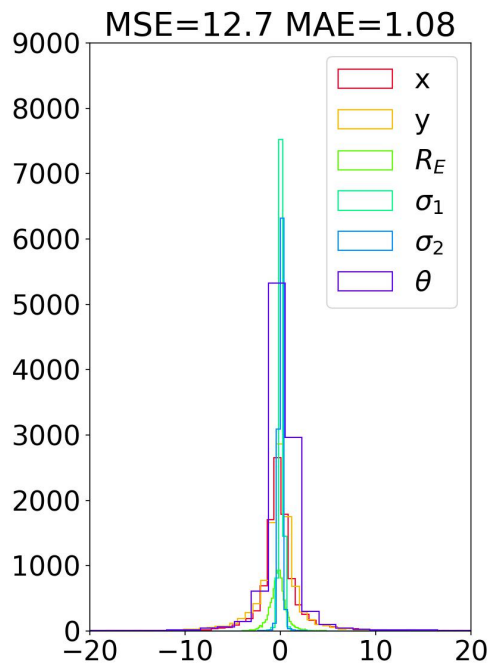


Figure 4.2.: Histograms from all the best performances for each network when limited to 100 epochs.

#### 4. Results



(a) Inception using  $10^{-4}$  learning rate      (b) Inception-v3 using transfer learning and  $10^{-4}$  learning rate



(c) VGG-19\_BN using  $10^{-4}$  learning rate

Figure 4.3.: Histograms showing the lowest deviations from ground truth using VGG-19\_BN and Inception-v3.

#### 4. Results

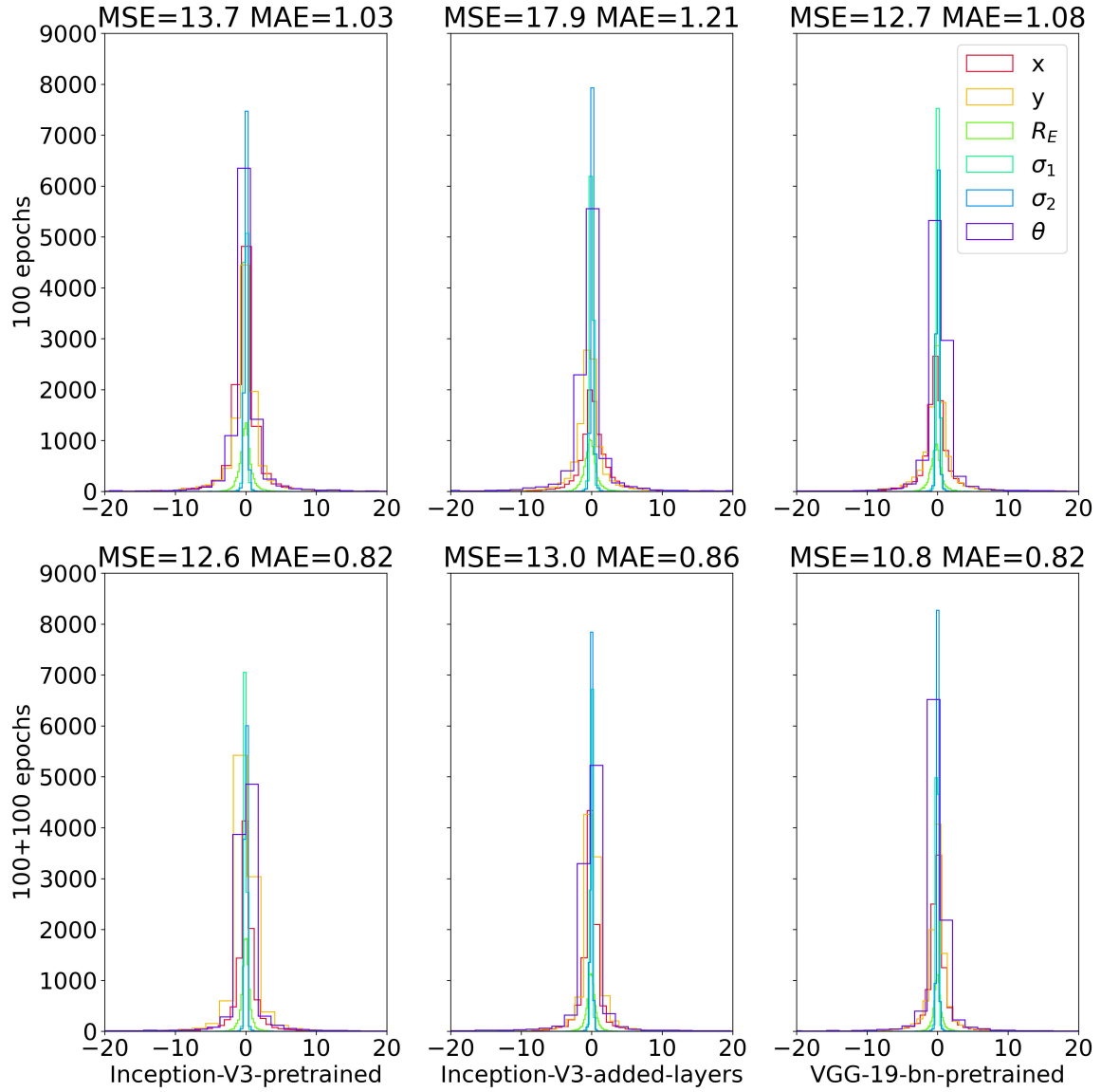
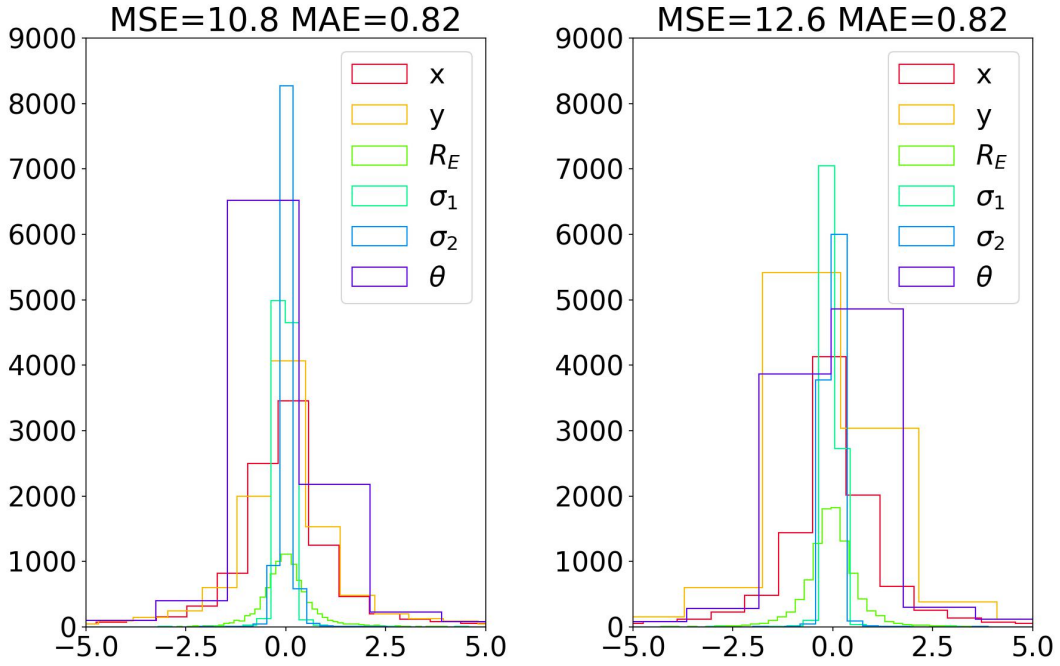


Figure 4.4.: Histograms showing different configurations of Inception-v3 and VGG-19\_BN run for either 100 epochs or 200 epochs.



#### 4. Results



(a) VGG-19\_BN pretrained for 200 epochs (b) Inception-v3 pretrained for 200 epochs

Figure 4.5.: Histograms showing the best results from Inception-v3 and VGG-19\_BN zoomed in to +/-5.

	Network	Inception-v3				VGG-19_BN			
	Epochs	100 epochs		100+100 epochs		100 epochs		100+100 epochs	
Parameter		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
x		10.4	1.50	5.80	1.15	9.01	1.62	6.05	1.89
y		9.66	1.53	9.77	1.18	9.21	1.64	6.14	1.20
$R_E$		1.05	0.69	0.58	0.49	1.04	0.71	0.57	0.49
$\sigma_1$		0.28	0.16	0.33	0.12	0.42	0.20	0.35	0.14
$\sigma_2$		0.29	0.17	0.33	0.12	0.41	0.19	0.35	0.14
$\theta$		60.5	2.13	58.9	1.82	56.0	2.10	51.4	1.79

Table 4.1.: Table showing best performing networks on separated parameters with MSE and MAE loss.

## 4. Results

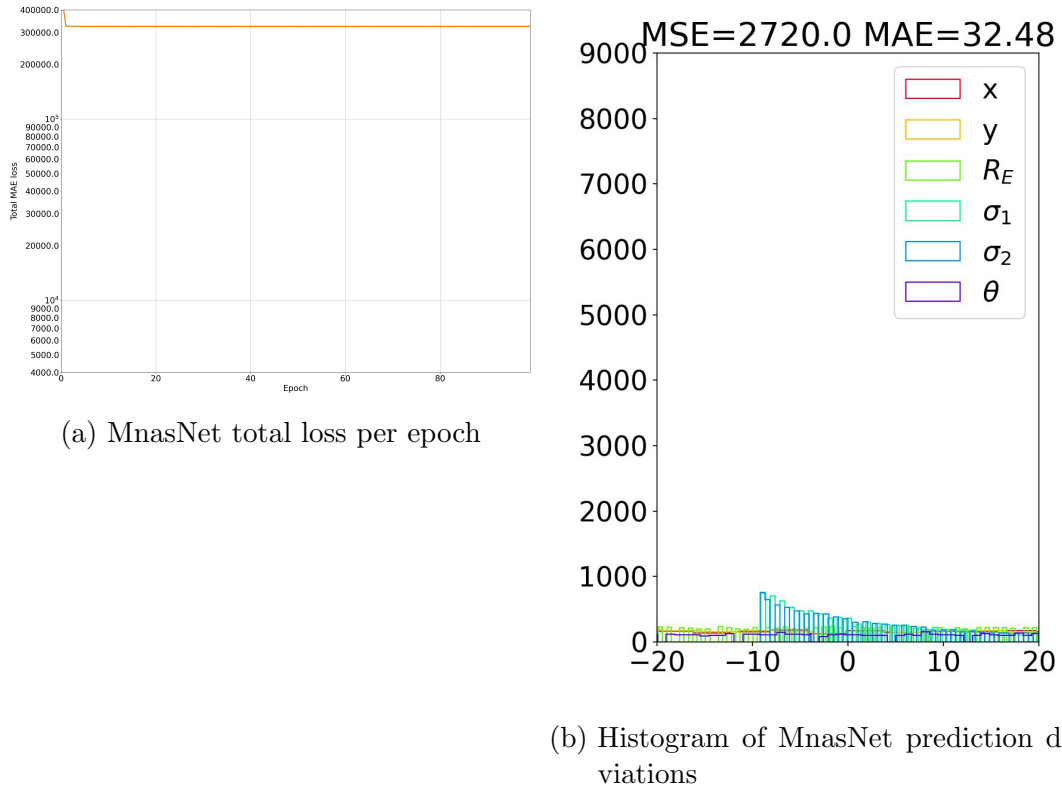
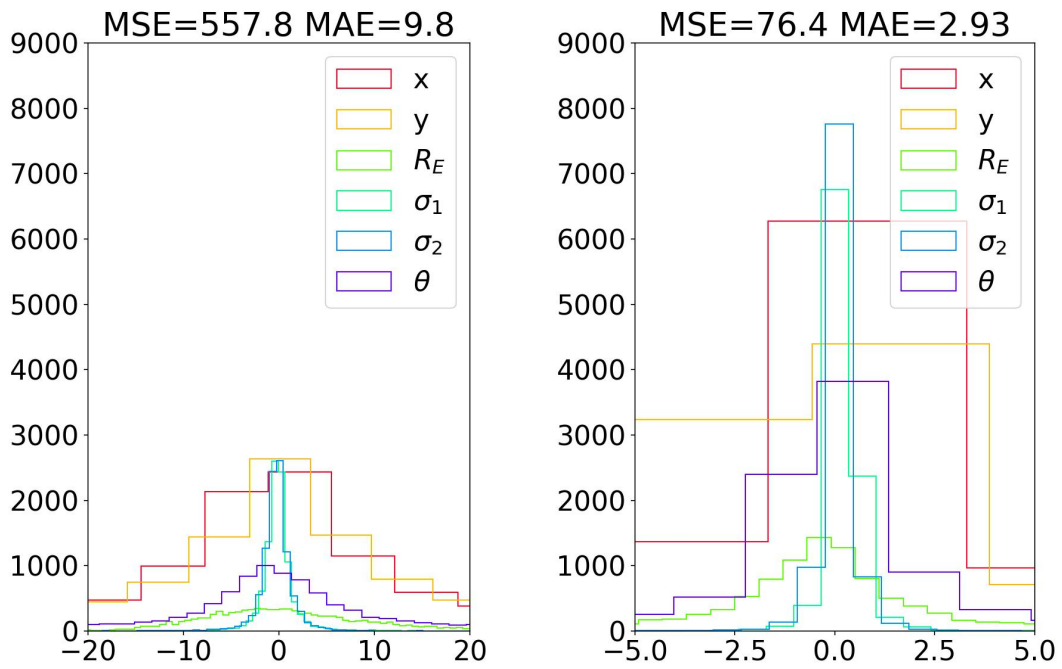
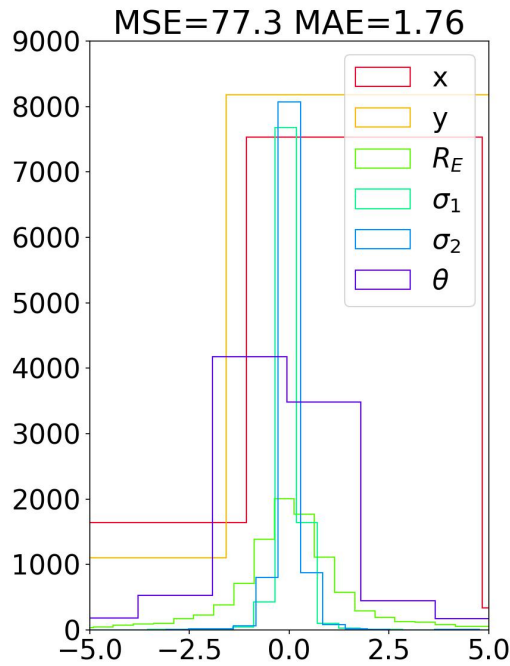


Figure 4.6.: Histograms showing the best results from Inception-v3 and VGG-19\_BN zoomed in to +/-5.

#### 4. Results



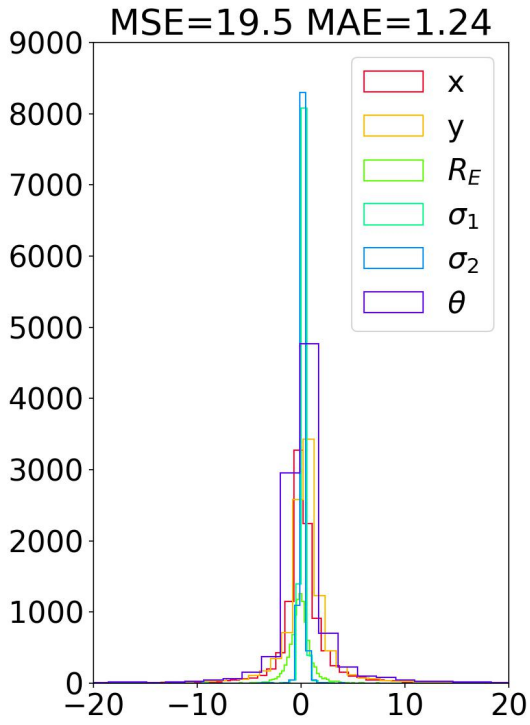
(a) Vision Transformer (ViT-16) with 0.0001 learning rate. (b) Swin with 0.0001 learning rate.



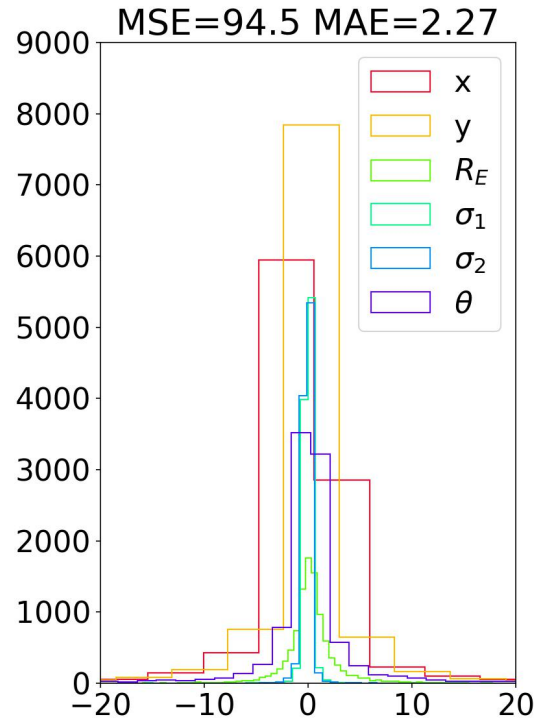
(c) ConvNeXt with 0.0001 learning rate.

Figure 4.7.: Histograms showing the performance of different vision transformers.

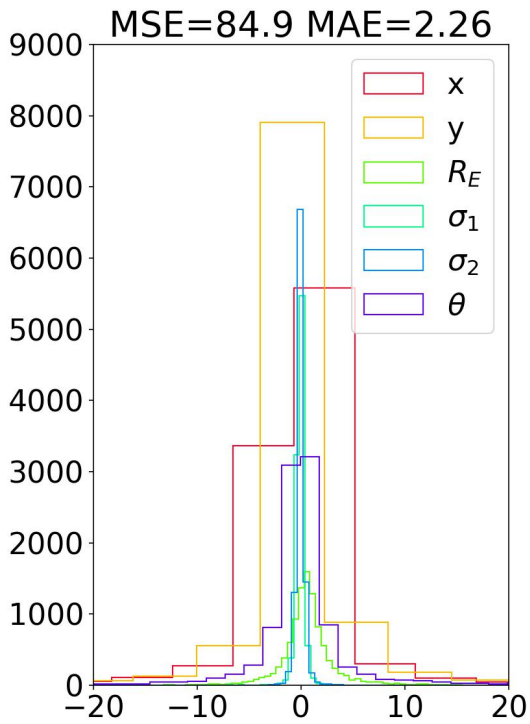
4. Results



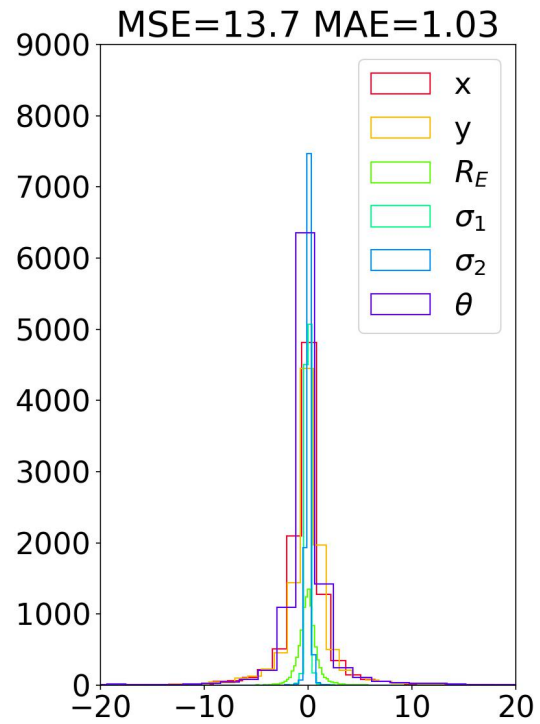
(a) Inception-v3 0.0001 learning rate



(b) Inception-v3 0.001 learning rate



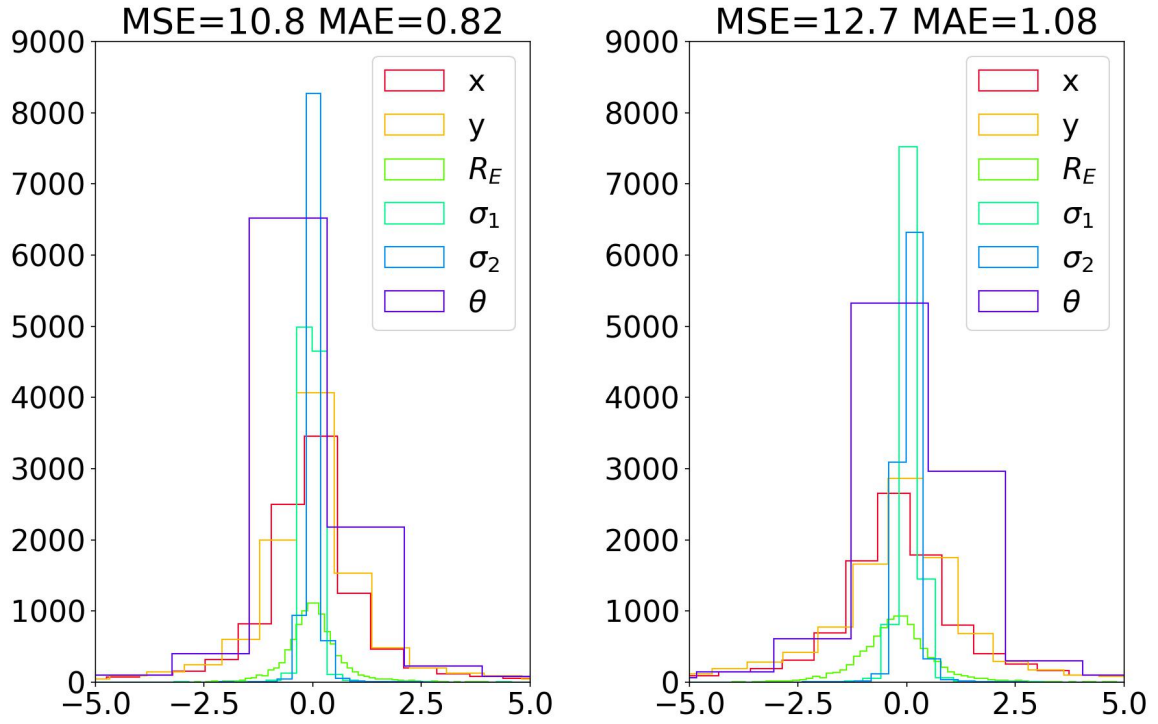
(c) Inception-v3 pre-training 0.001 learning rate



(d) Inception-v3 pre-trained 0.0001 learning rate

Figure 4.8.: Histograms showing the deviations from ground truth on different Inception-v3 configurations.

#### 4. Results



(a) VGG-19\_BN 0.0001 + 0.00001 learning rate

(b) VGG-19\_BN 0.0001 learning rate

Figure 4.9.: Histograms showing the deviations from ground truth on different VGG-19\_BN.

## 4. Results

Network	Top 1% accuracy on ImageNet	Lowest achieved MAE
Swin Transformer-v2-b	87.1%	2.93
ConvNeXt-b	85.8%	1.76
EfficientNet-v2-l	85.7%	1.91
Vision Transformer-b/16	85.22%	9.8
EfficientNet-b7	84.4%	1.39
ResNet-152	82.4%	1.42
MnasNet-6	76.7%(MnasNet-A3)	32.48
DenseNet-201	77.42%	2.02
Inception-v3	77.12%	0.82
VGG-19_BN	74.4%(VGG-16)	0.82
AlexNet	63.3%	2.91
SqueezeNet-v1.1	58.19%	3.16

Table 4.2.: Table comparing results in ImageNet competition and results achieved in this project.

Network model	Optimiser	Learning rate	Loss function	Batch size	Epochs
VGG-19_BN	Adam	0.0001	MSE	10	100+100

Table 4.3.: Hyperparameters chosen for best performing neural network.

Figure 4.10 shows histogram of the best four results. The figure has each parameter separated.

Table 4.2 compares the results from ImageNet challenge for top 1% of accuracy to the the results achieved in this project.

### 4.2. The best performing network

Table 4.3 shows the hyperparameters for the lowest MAE and MSE loss achieved. This was achieved by VGG-19\_BN, using transfer learning and two different learning rates. The network was first trained 100 epochs using  $10^{-4}$  learning rate. Afterwards the network was trained for another 100 epochs with  $10^{-5}$  learning rate. This achieved a MAE loss of 0.82 and MSE loss of 10.8.

## 4. Results

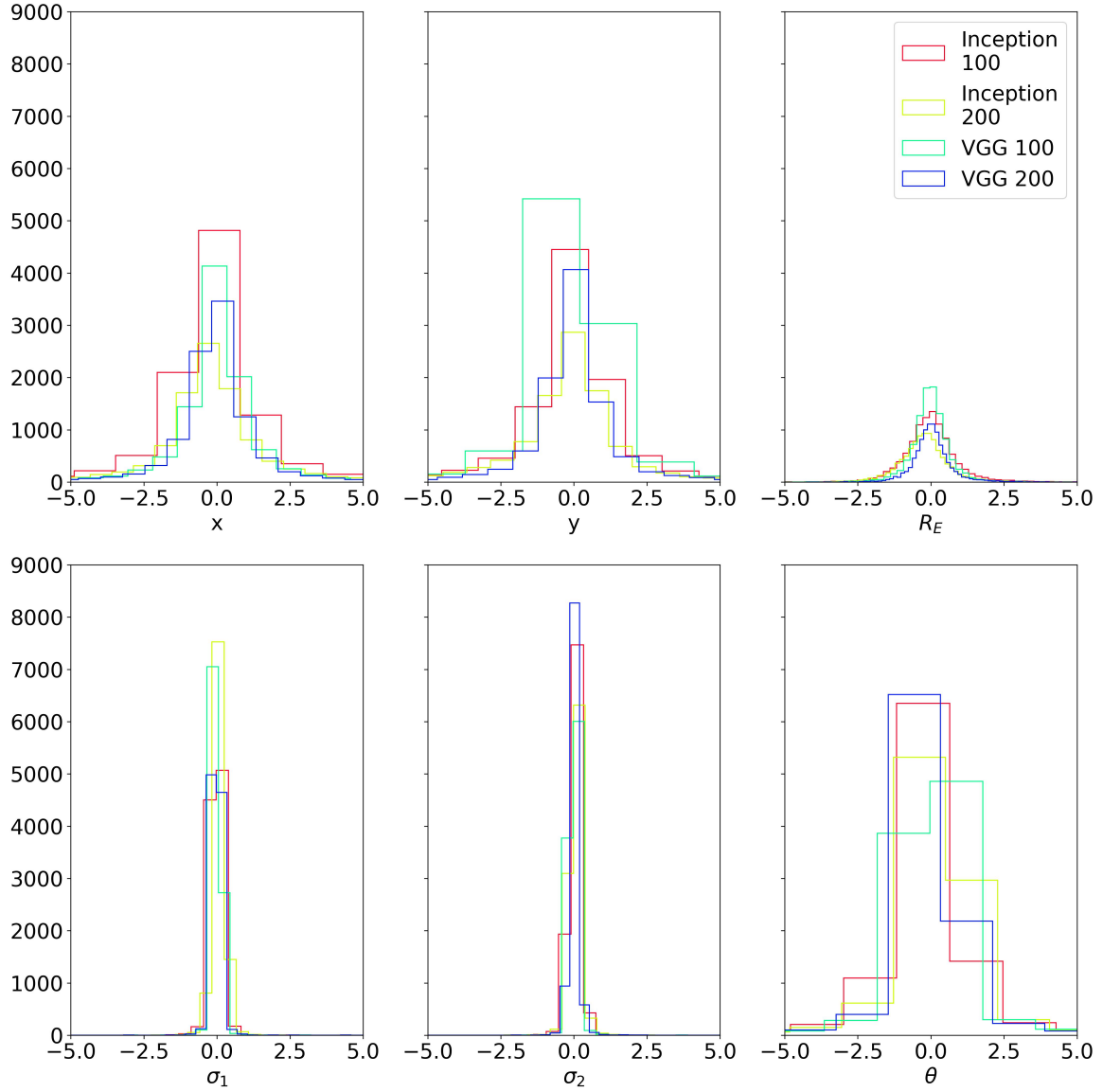
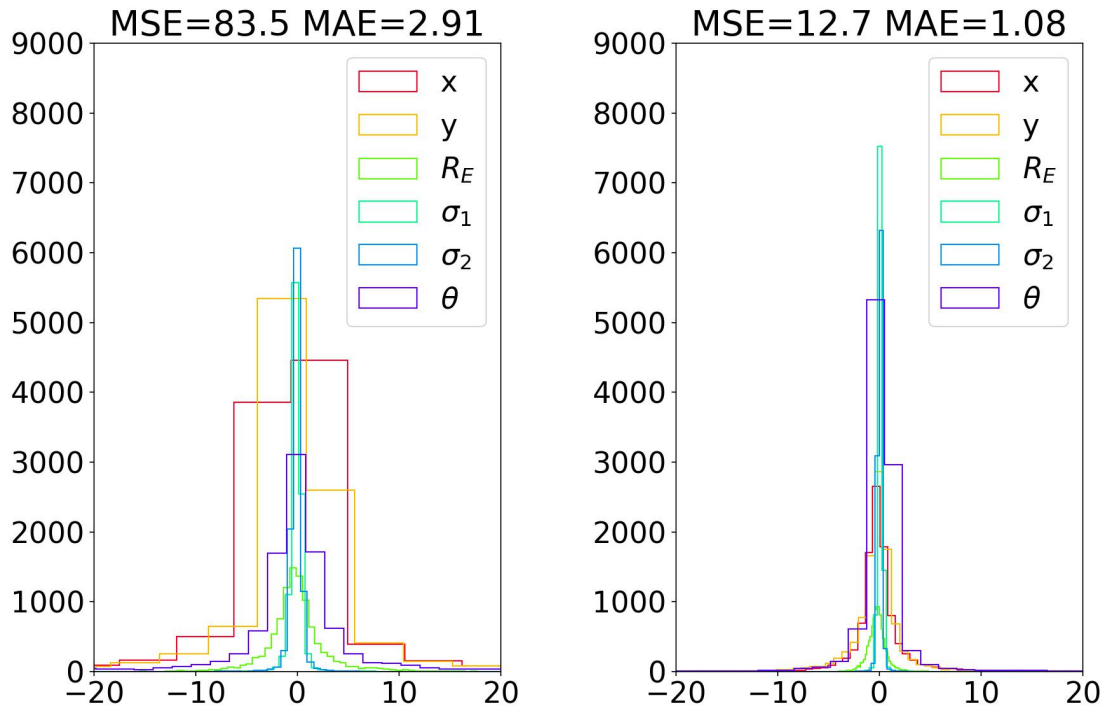


Figure 4.10.: Histogram showing the deviation from ground truth on all estimated parameters.

#### 4. Results



(a) AlexNet(Reference Network) pretrained + trained for 100 epochs with 0.001 learning rate  
(b) VGG architecture pretrained + trained 100 epochs with 0.001 learning rate + 100 epochs with 0.0001 learning rate

Figure 4.11.: Comparing best result to reference network.



### 4.3. How to run the system

A guide showing a possible way to run the systems can be found in Appendix G. All code can be found in Appendix A and B.

## 5. Discussion

This chapter aims to discuss the results and discoveries presented in chapter 3 and chapter 4. It will be contained to the parts relevant to the project itself. Anything about the project management, learning outcomes, or prerequisite knowledge can be found later in chapter 6.

### 5.1. Blunders to learn from

Use of Idun should have been done right from the start. We abstained from using it because of our believe that it might take a long time to setup and get it running. This was not as complicated as it seemed, and it only took a few hours to set up and one day to troubleshoot. The performance and the capacity for network training is much greater than what we used for most of the project. The reason for using Idun is that some of the models require more GPU memory than what we had.

Idun had significantly better hardware than otherwise obtainable with personal computers. Idun can run the code on an A100 GPU, which has significantly more tensor cores than currently possible on a normal personal desktop computer. Idun also allowed us to run multiple instances at once. This saved a good amount of time in the end, as one test of one network could take upwards of 50 hours, even on the better hardware. The time saved by Idun was crucial to being able to test this many network architectures, and without it many networks would have gone untested. Some tests took up to 5 days to run through Idun. The estimated effectiveness of Idun is around 10-12 times faster than what we could do beforehand on bigger models. This combined with being able to run up to 8 of them simultaneously allowed us to run around 80-100 times quicker than previously. While this sounds like a lot, at the start of the project work was spread on different task unrelated to training networks. As such the need for Idun was lower than it may appear, although work with Idun still should have been prioritised earlier.

Polar coordinates in 3.7 showed that estimating  $\phi$  was more difficult and prone to deviations if using MSE loss function. If we found this out earlier we could have saved some time by not running as many tests with polar coordinates, and focusing on cartesian coordinates the entire project. Figure 5.1 shows a histogram of the results from the

reference network with coordinates separated. This highlights that there is little to no difference in any of the other parameters guessed by the network. Only 5.1a shows a difference, and it seems like cartesian coordinates maintains a slimmer histogram and lower MAE/MSE loss. This shows that selecting cartesian coordinates was the correct choice. While the MAE loss can swing either way when trained with MAE loss function, the MSE loss is not close. Cartesian always performs better, sometimes even significantly better. This signals there are many outliers in the polar coordinates data not visible in the histograms.

### 5.2. AlexNet predicting same output for different inputs

The problem of AlexNet getting stuck guessing average values was unexpected to us. The group had never encountered such a problem, and it was difficult to understand why it was happening.

Contrary to our expectations increasing learning rate did not improve performance. We had expected it to be a problem of not having high enough learning rate, therefore not being incentivized to change. Contrary to this expectation, lowering learning rate by 10, or even 100 times prevented the network from getting stuck at the local minima. This lower learning rate even made it perform better.

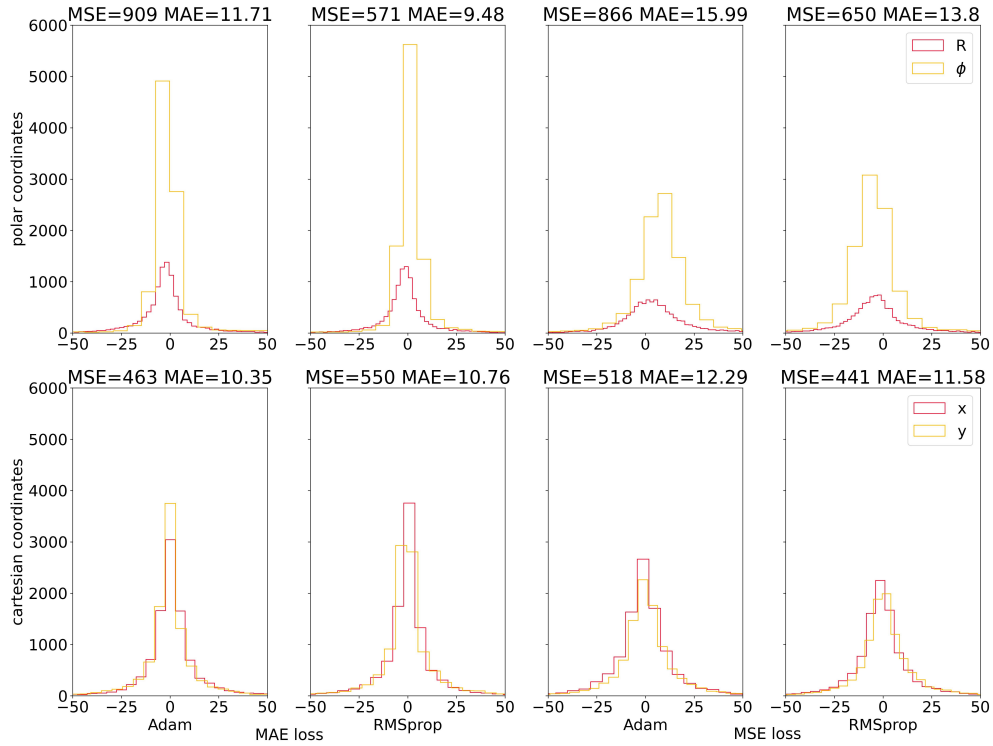
This problem was one we did not fully understand the reason behind on a deeper level. The group understands roughly why lowering learning rate is useful when facing this problem, but because the underlying reasons for this problem is not understood. It is hard for us to understand what lowering learning rate is doing to fix it specifically.

### 5.3. Achieving the best results

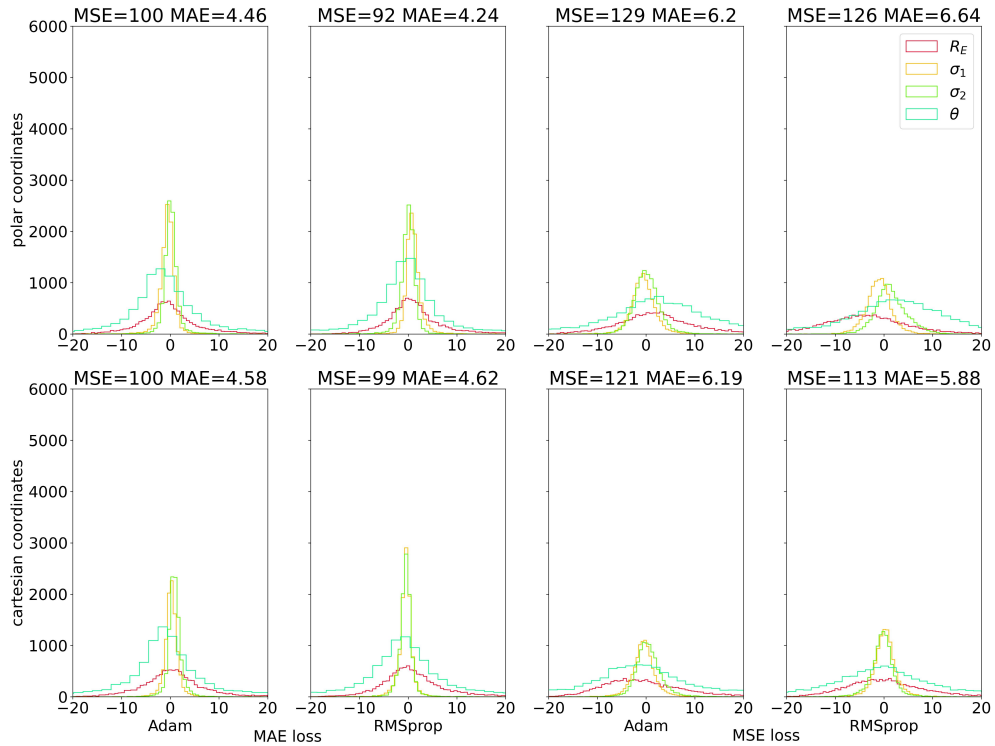
Searching for best performing network, showed us that bigger data sets don't lead to higher accuracy. This was not very surprising, as we had an idea already that 100 000 images was more than enough. While classification networks usually want as many images as possible to classify, the nature of using them for regression makes this different. The images we created were also decently limited in nature, with not much changing between them. This means a lower amount of images in the training data set is required for the network to properly learn. While 100 000 images seemed to be more than necessary, it is unknown what the lower limit necessary for good learning is.

Overall a few conclusions can be draw from these experiments. The most important

## 5. Discussion



(a) AlexNet(Reference Network) result for only coordinates



(b) Histogram showing all other parameters

Figure 5.1.: Histograms separating the results from coordinates and the other parameters estimated by AlexNet reference network.

## 5. Discussion

parameters in machine learning are learning rate, batch size and epoch number, with learning rate seeming like the most important. To the contrary of our expectations, bigger batch size did not lead to better performance of the network. Instead it made performance worse with added tendency to get stuck on local minimum. Our results fall in line with the findings of Master and Luschi (Masters and Luschi 2018). Bigger epoch number gives more time for network to learn and potentially jump out of locally found minimum.

### Training for more than 100 epoch

It's clear that training for more epochs with lower learning rate gives better performance. It is a marginal increase, but the increase in performance is still ever-present. Figure 4.1 show that most networks converge by the end of 100 epochs. Especially looking at the test data set total loss. Figure 3.12 show that networks are capable of getting lower loss on training data set with more training, but that does not convert to better loss results on test data. With this it can be assumed that the performance gained from the following 100 epochs are too small. The networks are likely to be near their global minimum, where they can not get much better. Any decrease in total loss seen after that point will only be the network getting more known with the training data set. This means that it will get better at solving those specific images with no regards to if the network is actually improving. Overall we are happy with this being the case, as it shows the networks got close to the limits after 100 epochs.

## 5.4. Discussing best network results

Inception-v3 and VGG-19\_BN performed the best compared to all other networks in this project, with VGG-19\_BN being the better one. It delivered slightly worse MAE loss but have considerably better MSE loss on test data set. Both networks managed to get a MAE loss close to 1 after 100 epochs and then lowered it under 1 with an extra 100 epochs of training. This was achieved by tuning learning rate to 0.0001, batch size to 10, using transfer learning, MAE loss function and Adam optimizer. All the hyperparameters for both the networks are shown in table 4.3. In the end the best overall performance was from VGG-19\_BN network architecture.

Figure 4.4 shows different configurations of Inception-v3 and VGG-19\_BN architectures. This shows that Inception-v3 can compete with VGG-19\_BN for the best network. Interestingly, Inception-v3 performs worse with added layers. The cause for this is unknown, but added layers proved to go either way in this project so it was not unexpected. Both network model managed to reach the same MAE loss of 0.82, a really solid result. At

## 5. Discussion

MSE loss VGG-19\_BN edged out Inception-v3 with only 10.8 loss against 12.6. These networks were very close in all results.

Figure 4.5 shows the best results from Inception-v3 and VGG-19\_BN, with x-axis limited to +/- 5. Zooming in this way allows us to see what both networks excels at, and to see if one network does something better than the other. When looking at the histograms closer, it appears that there are differences in how well they predict each parameter. Figure 4.5b shows that Inception-v3 struggles to very accurately predict  $y$  and  $\theta$ . While VGG-19-b in figure 4.5a performs overall well, it also struggles with estimating  $\theta$ . The reasons for having difficulty predicting  $y$  is unclear. A hypothesis for the difficulty shown predicting this is that it is related to  $\theta$ . Since both can affect where the object has shifted in vertical direction, it could be very difficult to predict them both accurately. All other parameters estimated follow a really good curve.

As shown by figure 4.10, the networks are really good at estimating  $\sigma_1$  and  $\sigma_2$ .  $R_E$  is worse, but not as bad as the other three. The networks all struggle at roughly the same variables,  $x$ ,  $y$ , and  $\theta$ . This builds on what was just discussed, and shows that neither of the networks can overcome this. It makes intuitive sense to us that the networks are better at estimating  $\sigma_1$  and  $\sigma_2$ . The impact of these two parameters are clearly visible in the images. Higher values of these parameters leads to more white pixels in the image. On the other hand,  $x$ ,  $y$ , and  $\theta$  are sort of intertwined. It can be difficult to know if the object is moved along and  $x$  and  $y$  axes, or if the object is rotated. Circumnavigating this issue can prove to be difficult without further limiting the image generation parameters. These findings are supported by table 4.1 showing the losses for each parameter in numerical values. This table shows that while the histogram might attribute the three parameters equal blame,  $\theta$  has by far the largest deviation, being around 6 times larger than  $x$  and  $y$ .

Determining if the results achieved are satisfactory is difficult. On one hand, we achieved a really low MAE and MSE loss, especially when compared to the work that was already done. Even after increasing the prediction difficulty by increasing image size, and by extension the parameter ranges. We also researched and found which network architectures are well fit for these problems, and how modifying them can benefit us. On the other hand, it is impossible to say if these results are good, since there is no real image reference. We believe the results are really good, and with more advanced methods of calculating the parameters discussed later in subsection 5.6 the results could become so good as to be considered exceptional.

## 5.5. Interesting tidbits

Vision Transformer (ViT), Swin Transformer, and ConvNeXt are all vision transformers. ViT and Swin performed significantly worse than any other type of deep learning networks with the exception of MnasNet-6 as seen in figures 4.1j, 4.1k and 4.1l. While vision transformers undoubtedly are one of the blooming fields in machine vision, it was outperformed by general convolutional networks here. The reason is expected to be because vision transformers are much harder to optimise. It is possible that with proper hyperparameter optimisation and models they can outperform the other convolutional networks. We did not have enough time to tune these networks, as they were among the last networks we tested.

One of the most interesting findings with vision transformers were that ViT and Swin performed practically equally well on the training and testing data sets. Figure 4.1 shows that when compared to all the other networks no other network comes close to this low of a deviation. This can indicate that they do not only get better at prediction the training data, but generally gets better at the task. This is wanted behaviour and good to see, although some discrepancy is expected between the two. Despite this wanted behaviour, the vision transformers did not perform as well as the others.

SqueezeNet also had an interesting quirk where the network would predict 0 for some parameters seemingly randomly. The network would predict normal/good values around half the time, and the other half it would guess 0 no matter what the ground truth was. This only happened with the same two parameters,  $x$  and  $y$ . Adding more layers to the end of the network seemed to completely fix this behaviour. After some researching as to why this could be the case, nothing was found. It only happened with SqueezeNet, and was fixed after adding more layers, so it was not a problem we concerned ourselves with due to prioritisation.

Figure 4.6 shows that MnasNet got stuck in local minimum. Fixing this could probably have been done by altering learning rate. As time to run experiments was running out at this point, work on this was halted. The short time left was spent on running more experiments on the network that had performed well already.

Another interesting thing found was that pre-training the networks with transfer learning could have a positive impact. These pretrained models are pretrained to be specifically better at classification. Therefore, it was interesting that certain networks could perform significantly better using these methods. For some networks the effect was negligible and for some it was negative, but for most it seemed to have a positive effect. This could indicate that better classification networks do better at regression tasks if the regression task only has visual inputs. This gives hope for further work with classification networks if wanted.

One of our initial theories was that general classification networks would perform better for regression than specialised ones. When comparing the results on both ends in table 4.2, it seems like there is a correlation to some degree. Both VGG-19\_BN and Inception-v3 scores significantly lower than the best classification networks in the ImageNet challenge. After this however, it is hard to find any conclusive links between the two results, so the hypothesis is not considered to be true. The vision transformers should most likely have performed better than they did, which would further disprove this hypothesis as well.

### 5.6. Suggestion for future work

Our supervisors expressed interest towards the end of the project in having the machine learning algorithm predict amplitudes instead. This is essential to develop roulette formalism further. This would not be a massively difficult process. Changing the parameters guessed would be very simple. It is unsure if the results would be as clear and as good. A way to extract the amplitudes when generating the images were created for us as well in case we had time. However, running all the tests again to find the best network would be too time consuming. The work already done should at least make it easy to change to predicting amplitude.

Another interesting thing the project could look into would be automatic learning rate adjustment, such as AutoLR with Lion optimiser. It's a method to change the learning rate throughout the training process. Our method consisted of finding the best learning rate manually, but this new method suggests there can be benefits to changing the learning rate over time. Implementing and testing this would be time consuming, and it is not clear to us how much of an improvement it would get.

The end goal of the project consists of running the code on real images. Our project was too early into the process to have a good reason to do this. Obtaining the images and their ground truth before having a good baseline machine learning implementation did not make sense. With the work we have done, changing over to real images in the future should be a lot easier, and hopefully the real data is similar enough to the synthetic data that the implementation requires minimal changes.

Another thing that needs to be looked into is the image generation code. As found in chapter 3.1, images with the same  $\frac{\chi}{R_E}$  should not generate the same images. Our supervisors have found these findings to be very interesting as it proves there is something wrong somewhere. Either the code is wrong or the mathematical framework is wrong. A look into the C++ code used to generate images is needed to look into this and the image artifacts.



## 5. Discussion

In the future, there might be regression models that can rival classification networks using images as inputs. Since regression based models are not currently good enough, it is not recommended to use the currently very simple regression models possible. If the field of regression models catches up, using one of these could prove to increase performance. If this was implemented, inputs could even change from images to statistical data.

Networks struggled with estimating source angle ( $\theta$ ) the most. This is because of some sources being generated spherically, that makes it impossible to estimate rotation. In some close cases this is near impossible. If the program could somehow first determine how spherical or elliptical a source is, it might be possible to improve this. For instance if the program detects a perfect sphere, rotation prediction might be turned off as it's not needed. Otherwise, weights could be used, so that the more spherical the source is, the less impacts the  $\theta$  prediction has.

Another possible implementation could be redshift analysis. This could be used to figure out the distance to the observed object. With a known distance, it should be possible to figure out  $\chi$ , which in turn means it should be possible to determine  $R_E$ . This would eliminate one of the bigger weaknesses of this model, which is the problem with fixed  $\frac{\chi}{R_E}$  ratio making the same images.

## 6. Retrospective

This chapter discusses thoughts about the project from a meta-perspective. Project management, learning outcomes and prerequisite knowledge will be briefly touched upon.

### 6.1. Project management

The group had decided to run a method of development where each person worked on their own. Combined with frequent communication through working together, this yielded a good amount of knowledge gained and shared. Since there was no work that relied on two people working on it at once through the entire project, this worked well. Sometimes when stuck, the group would switch jobs to get new perspectives on the problems. The few times both were stuck, the group had excellent help from the supervisors.

In hindsight the only thing lacking was a better to-do list curated more often. This would circumvent the problems of spending time not working on the most important tasks. Having a formal meeting once per week in addition to the informal conversations every day would help achieve this. In return this would ensure that work was always being focused on what was important.

### 6.2. Prerequisite knowledge

This section will have a list of the prerequisites for doing this task. Without these, the group would struggle to gather enough knowledge to complete the task in time.

List of prerequisite knowledge:

- Python programming
  - Python packages (matplotlib, pandas)
  - Intermediate programming knowledge

## 6. Retrospective

- Basic knowledge of machine learning (theoretical)
- Knowledge of coordinate systems
- Basic physics knowledge

### 6.3. Learning outcomes

This subsection will include points the group learned about while doing the project. Some of the things were built on knowledge already obtained, while other things had to be learned from scratch. List of relevant things we learned this project (bold/italic items are considered important and will be expanded upon below):

- Project management
  - Using GitHub
- **Machine learning**
  - **Machine learning networks**
  - Learning optimizers
  - **Programming neural networks**
  - Python programming
    - \* Using PyTorch
    - \* Good coding practice
  - More things
- Programming
  - Shell scripts
  - Basic C++ understanding

We learned a lot about the structures of machine learning networks, and how to program them. Throughout the project, we learned more and more useful pieces of knowledge

## 6. Retrospective

in a practical manner. Learning to implement and modify so many networks taught us what they have in common, and by extension what makes each unique. This can allow us to effectively use machine learning in the future.

When it comes to machine learning, we were able to put into practice what we had learned previously in the programme. The little machine learning theory we had learned already proved a good starting point, and we were happy to be able to apply it in a real scenario.

## 7. Conclusion

The purpose of this thesis was to attempt to assist in the development of NTNU's research effort on mapping dark matter. Since the given problem was very open ended, we had to set our own goals and expectations. Through machine learning, we found the most likely convolutional neural network candidates for such a task. Due to little previous research on the topic, and none of it being open-source, a lot of testing and prototyping had to be done. In the end, the results achieved were satisfactory. We also found theoretical flaws in previous work. Alongside these, we proposed multiple different improvements the project can implement going forward that we did not have time for. We hope our contributions are helpful in making this project a success.

# Bibliography

- Aghanim, N. et al. (Sept. 2020). «Planck 2018 results». In: *Astronomy & Astrophysics* 641, A6. ISSN: 1432-0746. DOI: 10.1051/0004-6361/201833910. URL: <http://dx.doi.org/10.1051/0004-6361/201833910>.
- Anwar, Aqeel (Jan. 2022). *Difference between alexnet, vggnet, ResNet and inception*. URL: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96>.
- Clarkson, Chris (Nov. 2016). «Roulettes: a weak lensing formalism for strong lensing: II. Derivation and analysis\*». In: *Classical and Quantum Gravity* 33.24, p. 245003. DOI: 10.1088/0264-9381/33/24/245003. URL: <https://dx.doi.org/10.1088/0264-9381/33/24/245003>.
- Congdon, A.B. and C.R. Keeton (2018). *Principles of Gravitational Lensing: Light Deflection as a Probe of Astrophysics and Cosmology*. Springer Praxis Books. Springer International Publishing. ISBN: 9783030021221.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.
- Huang, X. et al. (Mar. 2021). «Discovering New Strong Gravitational Lenses in the DESI Legacy Imaging Surveys». In: *The Astrophysical Journal* 909.1, p. 27. DOI: 10.3847/1538-4357/abd62b. URL: <https://doi.org/10.3847/1538-4357/abd62b>.
- Ingebritsen, Simon et al. (2022). «CosmoAI: A study of gravitational lensing through simulation and machine learning». Bachelor's Thesis. NTNU, pp. 10–17.
- Khramtsov, Vladislav et al. (2019). «KiDS-SQuAD - II. Machine learning selection of bright extragalactic objects to search for new gravitationally lensed quasars». In: *A&A* 632, A56. DOI: 10.1051/0004-6361/201936006. URL: <https://doi.org/10.1051/0004-6361/201936006>.
- Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- Lecun, Y. et al. (1998). «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

## Bibliography

- Li, Zewen et al. (2022). «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects». In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12, pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.
- Magro, Daniel et al. (June 2021). «A comparative study of convolutional neural networks for the detection of strong gravitational lensing». In: *Monthly Notices of the Royal Astronomical Society* 505.4, pp. 6155–6165. ISSN: 0035-8711. DOI: 10.1093/mnras/stab1635. URL: <https://doi.org/10.1093/mnras/stab1635>.
- Masters, Dominic and Carlo Luschi (2018). *Revisiting Small Batch Training for Deep Neural Networks*. arXiv: 1804.07612 [cs.LG].
- Morgan, Robert et al. (2021). «deepenstronomy: A dataset simulation package for strong gravitational lensing». In: *Journal of Open Source Software* 6.58, p. 2854. DOI: 10.21105/joss.02854. URL: <https://doi.org/10.21105/joss.02854>.
- Ranganathan, Hiranmayi et al. (2020). «Deep Active Learning for Image Regression». In: *Deep Learning Applications*. Ed. by M. Arif Wani, Mehmed Kantardzic, and Moamar Sayed-Mouchaweh. Singapore: Springer Singapore, pp. 113–135. ISBN: 978-981-15-1816-4. DOI: 10.1007/978-981-15-1816-4\_7. URL: [https://doi.org/10.1007/978-981-15-1816-4\\_7](https://doi.org/10.1007/978-981-15-1816-4_7).
- Remmen, Grant N. (Nov. 2021). «Exploration of a singular fluid spacetime». In: *General Relativity and Gravitation* 53.11. DOI: 10.1007/s10714-021-02873-5. URL: <https://doi.org/10.1007/s10714-021-02873-5>.
- Rezaei, S et al. (July 2022). «A machine learning based approach to gravitational lens identification with the International LOFAR Telescope». In: *Monthly Notices of the Royal Astronomical Society* 517.1, pp. 1156–1170. ISSN: 0035-8711. DOI: 10.1093/mnras/stac2078. eprint: <https://academic.oup.com/mnras/article-pdf/517/1/1156/46441535/stac2078.pdf>. URL: <https://doi.org/10.1093/mnras/stac2078>.
- Russell, Stuart J. and Peter Norvig (2022). *Artificial Intelligence: a modern approach*. 4th ed. Pearson.
- Tan, Mingxing et al. (2019). *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. arXiv: 1807.11626 [cs.CV].
- Wilde, Joshua et al. (Feb. 2022). «Detecting gravitational lenses using machine learning: exploring interpretability and sensitivity to rare lensing configurations». In: *Monthly Notices of the Royal Astronomical Society* 512.3, pp. 3464–3479. ISSN: 0035-8711. DOI: 10.1093/mnras/stac562. eprint: <https://academic.oup.com/mnras/article-pdf/512/3/3464/43249501/stac562.pdf>. URL: <https://doi.org/10.1093/mnras/stac562>.

# A. Appendix Cosmo-ML code

The code is in a open source Github repo: <https://github.com/ModeS7/Cosmo-ML>  
Most of the changes made in code files for networks are delete code because of our images having one channel non of transfer learning worked and that created allot of errors, so all of it was removed.

The code below originally is created by our supervisor Hans Georg Schaathun and then edited and used by us. The highlighted lines in the code is what we have contributed.

The following code below is from CudaModel.py:

```
#!/usrbin/env python3

"""
The MLSystem class provides defaults for all the components of
a machine learning system. The default implementation uses the CPU.
Subclasses should override key functions for more advanced systems.
"""

import torch
# import torch.nn as nn
# from torch.utils.data import DataLoader
from MLSystem import MLSystem, getArgs

# from Dataset import *
import cudaaux

class CudaModel(MLSystem):
    def __init__(self,model=None,criterion=None,optimizer=None,nepoch=2,
                 learning_rate=0.0001):
        super().__init__(model,criterion,optimizer,nepoch, learning_rate)

        if not torch.cuda.is_available():
            raise Exception( "CUDA is not available" )

        self.device = torch.device( "cuda" )
```



## A. Appendix Cosmo-ML code

```
self.model = self.model.to(self.device)
if args.weights:
    self.model.load_state_dict(torch.load(args.weights))

if __name__ == "__main__":
    args = getArgs()

    print( "CudaModel (CosmoML) test script." )
    cudaaux.cudaDiagnostic()
    print( "Configuring ... " )

    ml = CudaModel(model= 'vgg_pretrained')
    ml.systemTest(args)
```

## A. Appendix Cosmo-ML code

The following code below is from MLSystem.py:

```
#!/usrbin/env python3

"""
The MLSystem class provides defaults for all the components of
a machine learning system. The default implementation uses the CPU.
Subclasses should override key functions for more advanced systems.
"""

import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.utils.data import Subset

import time
# from tqdm import tqdm

from Dataset import *
from EvalObject import EvalObject, PredObject
from Networks.Inception3 import Inception3
from Networks.AlexNet import AlexNet
from Networks.ResNet import resnet18, resnet34, resnet50, \
    resnet101, resnet152, resnext50_32x4d, resnext101_32x8d, \
    resnext101_64x4d, wide_resnet50_2, wide_resnet101_2
from Networks.VGG import vgg11, vgg13, vgg16, \
    vgg19, vgg11_bn, vgg13_bn, vgg16_bn, vgg19_bn
from Networks.DenseNet import densenet121, \
    densenet161, densenet169, densenet201
from Networks.EfficientNet import efficientnet_b0, \
    efficientnet_b2, efficientnet_b3, \
    efficientnet_b4, efficientnet_b5, efficientnet_b6, \
    efficientnet_b7, efficientnet_v2_l, efficientnet_v2_m, \
    efficientnet_v2_s, efficientnet_b3_5, efficientnet_b4_5
from Networks.ConvNeXt import convnext_tiny, \
    convnext_small, convnext_base, convnext_large
# from Networks.NASNet import NASNetAMobile, NASNetALarge
from Networks.MnasNet import mnasnet1_0, mnasnet3_8, mnasnet6_0
from Networks.SqueezeNet import squeezenet1_0, squeezenet1_1
from Networks.Vision_Transformer import vit_b_16, vit_b_32, \
    vit_l_16, vit_l_32, vit_h_14
from Networks.Swin_Transformer import swin_t, swin_s, swin_b, swin_v2_b, \
    swin_v2_t, swin_v2_s
```

## A. Appendix Cosmo-ML code

```

from torchvision.models.inception import BasicConv2d
import torchvision.models as models
import argparse

class MLSystem:
    def __init__(self, model=None, criterion=None, optimizer=None, nepoch=2, learning_rate=0.001):
        """Construct a Machine Learning system for CosmoSim data.

        :param model: a pyTorch model instance; default `Inception3()`
        :param criterion: a pyTorch loss function; default `MSELoss()`
        :param optimizer: a pyTorch optimiser; default `Adam()`
        """

        self.num_epochs = nepoch
        self.batch_size = 10
        self.learning_rate = learning_rate
        self.device = None
        self.nparams = len(CosmoDataset1._columns)
        self.epochstrained = 0
        self.incep = False

        # Initialize your network, loss function, and optimizer
        if model == None:      # For quick testing
            self.model = AlexNet(num_outputs=self.nparams, extra_layers=False)
            # self.model = squeezenet1_1(num_outputs=self.nparams, extra_layers=True)
            # self.model = Inception3(num_outputs=self.nparams, extra_layers=True)
            # self.model = resnet152(num_outputs=self.nparams, extra_layers=True)
            # self.model = vgg19(num_outputs=self.nparams) # 19_bn does not run on
            # self.model = efficientnet_b3_5(num_outputs=self.nparams, extra_layers=True)
            # self.model = densenet201(num_outputs=self.nparams, extra_layers=True)
            # self.model = convnext_tiny(num_outputs=self.nparams)
            # self.model = mnasnet3_8(num_outputs=self.nparams, extra_layers=True)
            # self.model = NASNetAMobile(num_outputs=self.nparams) # does not work,
            # self.model = vit_b_16(num_outputs=self.nparams)
            # self.model = swin_v2_s(num_outputs=self.nparams)

        if model == 'alexnet_pretrained':
            alexnet = models.alexnet(pretrained=True)
            alexnet.features[0] = nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2)
            alexnet.classifier[6] = nn.Linear(4096, self.nparams)
            self.model = alexnet

        if model == 'squeezenet_pretrained':
            squeezenet = models.squeezenet1_1(pretrained=True)
            squeezenet.features[0] = nn.Conv2d(1, 64, kernel_size=3, stride=2)

```

## A. Appendix Cosmo-ML code

```
final_conv = nn.Conv2d(512, 256, kernel_size=3, padding=1)
squeezenet.final_conv = nn.Conv2d(512, 256, kernel_size=3, padding=1)
squeezenet.classifier = nn.Sequential(
    nn.Dropout(p=0.5),
    final_conv,
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2),
    nn.Flatten(),
    nn.Linear(1024 * 6 * 6, 1024),
    nn.ReLU(inplace=True),
    nn.Linear(1024, 512),
    nn.ReLU(inplace=True),
    nn.Linear(512, self.nparams)
)
self.model = squeezenet
if model == 'inception_pretrained':
    inception = models.inception_v3(pretrained=True, transform_input=False)
    inception.Conv2d_1a_3x3 = BasicConv2d(1, 32, kernel_size=3, stride=2)
    inception.fc = nn.Sequential(
        nn.Linear(2048, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, self.nparams))
    self.model = inception
    self.incep = True
if model == 'inception_pretrained_vanila':
    inception = models.inception_v3(pretrained=True, transform_input=False)
    inception.Conv2d_1a_3x3 = BasicConv2d(1, 32, kernel_size=3, stride=2)
    inception.fc = nn.Linear(2048, self.nparams)
    self.model = inception
    self.incep = True
if model == 'resnet_pretrained':
    resnet = models.resnet152(pretrained=True)
    resnet.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
    resnet.fc = nn.Sequential(
        nn.Linear(resnet.fc.in_features, self.nparams))
    self.model = resnet
if model == 'densenet_pretrained':
    densenet = models.densenet201(pretrained=True)
    densenet.features.conv0 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
    densenet.classifier = nn.Sequential(
        nn.Linear(densenet.classifier.in_features, self.nparams),)
    self.model = densenet
```

## A. Appendix Cosmo-ML code

```
if model == 'vgg_pretrained':
    vgg = models.vgg16(pretrained=True)
    vgg.features[0] = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)
    num_features = vgg.classifier[-1].in_features
    vgg.classifier[-1] = torch.nn.Linear(num_features, self.nparams)
    self.model = vgg

if model == 'alexnet':
    self.model = AlexNet(num_outputs=self.nparams, extra_layers=False)
if model == 'squeezenet':
    self.model = squeezenet1_1(num_outputs=self.nparams, extra_layers=True)
if model == 'inception':
    self.model = Inception3(num_outputs=self.nparams, extra_layers=True)
if model == 'inception_vanila':
    self.model = Inception3(num_outputs=self.nparams, extra_layers=False)
if model == 'resnet':
    self.model = resnet152(num_outputs=self.nparams, extra_layers=True)
if model == 'vgg':
    self.model = vgg19(num_outputs=self.nparams)
if model == 'efficientnet':
    self.model = efficientnet_b3_5(num_outputs=self.nparams, extra_layers=True)
if model == 'efficientnet_vanilla':
    self.model = efficientnet_b7(num_outputs=self.nparams, extra_layers=False)
if model == 'efficientnet_v2':
    self.model = efficientnet_v2_1(num_outputs=self.nparams, extra_layers=True)
if model == 'densenet':
    self.model = densenet201(num_outputs=self.nparams, extra_layers=True)
if model == 'convnext':
    self.model = convnext_tiny(num_outputs=self.nparams)
if model == 'mnasnet':
    self.model = mnasnet3_8(num_outputs=self.nparams, extra_layers=True)
if model == 'vit':
    self.model = vit_b_16(num_outputs=self.nparams)
if model == 'swin':
    self.model = swin_v2_s(num_outputs=self.nparams)

if criterion == None:
    # The default criterion is Mean Squared Error
    #self.criterion = nn.MSELoss()
    self.criterion = nn.L1Loss()
    # criterations for evaluation
    self.criterionMSE = nn.MSELoss()
```

## A. Appendix Cosmo-ML code

```

    self.criterionMAE = nn.L1Loss()
if optimizer == None:
    # self.optimizer = torch.optim.SGD(self.model.parameters(),
    #                                   lr=self.learning_rate, momentum=0.9)
    self.optimizer = torch.optim.Adam(self.model.parameters(),
                                       lr=self.learning_rate)
    # self.optimizer = torch.optim.RMSprop(self.model.parameters(),
    #                                       lr=self.learning_rate)

def loadtrainingdata(self, fn="train.csv", dataset=None,
                    dataFraction=0.1, dataFractionTest=0.1):
    """Load the dataset for training.
    The parameter may be either a filename `fn` which would
    be loaded into a `CosmoDataset`, or `dataset` which
    should be a pre-defined `CosmoDataset` object (of any subclass).
    dataFraction decided the fraction of total images to be
    used in training.
    dataFractionTest decides the fraction of total images to
    be used in testing of training data.
    """
    if dataset:
        self.train_dataset = dataset
    else:
        self.train_dataset = CosmoDataset1(fn)
    self.ntrain = len(self.train_dataset)
    self.train_datasetTest = Subset(self.train_dataset,
                                     range(int(self.ntrain * dataFractionTest)))
    self.trainloaderTest = DataLoader(dataset=self.train_datasetTest,
                                       batch_size=self.batch_size, shuffle=True)
    self.ntrainTest = len(self.train_datasetTest)

    self.train_dataset = Subset(self.train_dataset,
                                 range(int(self.ntrain * dataFraction)))
    self.ntrain = len(self.train_dataset)

    self.trainloader = DataLoader(dataset=self.train_dataset,
                                   batch_size=self.batch_size, shuffle=True)
    self.img_size = self.train_dataset[0][0].shape

def loadtestdata(self, fn="test.csv", dataset=None):
    """Load the dataset for testing.
    The parameter may be either a filename `fn` which would
    be loaded into a `CosmoDataset1`, or a pre-defined

```

## A. Appendix Cosmo-ML code

```
`CosmoDataset` object (of any subclass).  
dataFraction decided the fraction of total  
images to be used in training.  
"""  
if dataset:  
    self.test_dataset = dataset  
else:  
    self.test_dataset = CosmoDataset1(fn)  
self.testloader = DataLoader(dataset=self.test_dataset,  
                             batch_size=self.batch_size)  
self.ntrain = len(self.train_dataset)  
self.ntrainTest = len(self.test_dataset)  
  
def printparam(self):  
    """Print statistics of the training scenario to stdout.  
    This includes epoch number, dataset sizes, and image size."""  
    print(f'num_epochs: {self.num_epochs}, '  
          + f'batch size: {self.batch_size}, lr: {self.learning_rate}')  
    print(f'image size: {self.img_size}')  
    print(f'train samples: {self.ntrain}({self.ntrainTest}) '  
          + f'test samples: {self.ntrainTest}\n')  
  
def trainOne(self, verbose=True):  
    """Train the network for one epoch.  
    Return the total loss.  
    If `verbose` is `True` the training loss is printed on stdout  
    for each minibatch.  
    """  
    tloss = 0.0  
    epoch = self.epochstrained  
    for i, (images, params, index) in enumerate(self.trainloader):  
        if self.device:  
            images = images.to(self.device)  
            params = params.to(self.device)  
            self.optimizer.zero_grad()  
  
            # Forward + Backward + Optimiser  
            output = self.model(images)  
            if epoch == 0 and self.incep:  
                output = output[0]  
            loss = self.criterion(output, params)  
            loss.backward()  
            self.optimizer.step()
```

## A. Appendix Cosmo-ML code

```
tloss += loss.item() * len(images)
if verbose:
    print(f"Batch no. {epoch + 1}-{i + 1}: loss = {loss.item()}; "
          f"tloss = {tloss}")
self.epochstrained += 1
return tloss

def train(self, nepoch=None, test=False, verbose=True):
    """Train the network.

    The return value is a list of training losses per epoch if
    `test` is `False`. If `test` is `True`, the return value
    is an `EvalObject` containing comprehensive performance
    statistics.

    :param nepoch: Number of epochs; if None the object default is used.
    :param test: If True, the model is tested after each epoch
    :param verbose: If True, training loss is written for each epoch
    """
    timer = time.time()
    lossAcrossEpochs = []
    if nepoch == None:
        nepoch = self.num_epochs
    else:
        nepoch += self.epochstrained
    startidx = self.epochstrained
    try:
        # This is based on Listing 3-4.
        self.model.train()
        for epoch in range(startidx, nepoch):

            tloss = self.trainOne(verbose=verbose)
            if verbose:
                print(f"Epoch {epoch + 1}: Loss = {tloss}")
            if test:
                ob = {"loss": tloss,
                    "training": self.getLoss(trainingset=True),
                    "test": self.getLoss()
                    }
                if verbose: print(ob)
                lossAcrossEpochs.append(ob)
            else:
                lossAcrossEpochs.append(tloss)
```



## A. Appendix Cosmo-ML code

```
except KeyboardInterrupt:
    print("Training aborted by keyboard interrupt.")

if test:
    lossAcrossEpochs = EvalObject(lossAcrossEpochs)
    lossAcrossEpochs.setHeaders(self.test_dataset.getSlice())
return lossAcrossEpochs

def getLoss(self, printDetails=True, trainingset=False):
    """
    Test the model and report various performance heuristics.

    :param printDetails: if True, heuristics are printed on stdout
    :param trainingset: if True, the test is made on the traininset
    :returns: A `dict` with various heuristics
    """
    total_loss, total_MSE_loss, total_MAE_loss = 0, 0, 0

    self.model.eval()
    errors = []
    if trainingset:
        dataloader = self.trainloaderTest
    else:
        dataloader = self.testloader
    with torch.no_grad():
        for (images, params, index) in dataloader:
            if self.device:
                images = images.to(self.device)
                params = params.to(self.device)
            output = self.model(images)
            loss = self.criterion(output, params)
            MSE_loss = self.criterionMSE(output, params)
            MAE_loss = self.criterionMAE(output, params)
            total_loss += loss * len(images)
            total_MSE_loss += MSE_loss * len(images)
            total_MAE_loss += MAE_loss * len(images)
            for i, param in enumerate(params):
                error = output[i] - param
                mse = error ** 2
                mse = mse.sum().item() / self.nparams
                errors.append(error)
            if printDetails:
                niceoutput = [round(n, 3) for n in output[i].tolist()]
                niceparam = [round(n, 3) for n in param.tolist()]
```

## A. Appendix Cosmo-ML code

```
        if i < 1: # Print only the first image in the batch
            print(f"{'f'{round(mse, 4)} Correct: {niceparam}' : <40}"
                  f"{'f'Output: {niceoutput}' : ^40}")
# The average is wrong if the last batch has fewer images
errorMat = torch.stack(errors)
errorAbs = errorMat.abs()
mean = errorAbs.mean(axis=0)
stdev = errorAbs.std(axis=0, unbiased=True)
mean2 = errorMat.mean(axis=0)
stdev2 = errorMat.std(axis=0, unbiased=True)
if printDetails:
    print("Mean absolute error", mean)
    print("Standard deviation", stdev)
    print("Mean signed error", mean2)
    print("Standard deviation", stdev2)
    print("Total MSE loss", total_MSE_loss)
    print("Total MAE loss", total_MAE_loss)
    print(f"Loss/sample = {total_loss / self.ntest}")
return {"TotalMSE": total_MSE_loss,
        "TotalMAE": total_MAE_loss,
        "ErrorMean": mean2,
        "ErrorStDev": stdev2,
        "AbsMean": mean,
        "AbsStDev": stdev
        }

def getPred(self, trainingset=False):
    """
    Test the model and return the prediction results.

    :param trainingset: if True, the test is made on the traininset
    :returns: A tensor containing index, ground truth, and prediction.
    """
    self.model.eval()
    t1 = []
    if trainingset:
        dataloader = self.trainloader
    else:
        dataloader = self.testloader
    i = 0
    with torch.no_grad():
        for (images, params, index) in dataloader:
            if self.device:
                images = images.to(self.device)
```

## A. Appendix Cosmo-ML code

```
        params = params.to(self.device)
        output = self.model(images)
        idxtensor = index.reshape((len(index), 1))
        print("batch number:",i)
        i += 1
        tl.append(torch.cat([
            idxtensor, params.cpu(), output.cpu()], axis=1))
    return torch.cat(tl)

def savemodel(self, fn="save-model"):
    """
    Save the pyTorch model to file.
    Note that the `MLSystem` object is not stored; only the
    actual trained model.
    """
    torch.save(self.model.state_dict(), fn)

def systemTest(self, args):
    """Test the entire system with training and testing.
    The input is an object returned by getArgs(), and thus
    representing CLI arguments.
    """
    if args.imagedir:
        imgdir = args.imagedir
    else:
        imgdir = "./"
    if args.imagedirtest:
        imgdirtest = args.imagedirtest
    else:
        imgdirtest = "./"

    if args.amp6:
        ob = CosmoDataset2(csvfile=args.train, imgdir=imgdir)
        self.loadtrainingdata(dataset=ob)
        ob = CosmoDataset2(csvfile=args.test, imgdirtest=imgdirtest)
        self.loadtestdata(dataset=ob)
    else:
        ob = CosmoDataset1(csvfile=args.train, imgdir=imgdir)
        self.loadtrainingdata(dataset=ob)
        ob = CosmoDataset1(csvfile=args.test, imgdirtest=imgdirtest)
        self.loadtestdata(dataset=ob)
```

## A. Appendix Cosmo-ML code

```
self.printparam()
print("Training ...")
if args.epochs:
    nepochs = int(args.epochs)
else:
    nepochs = None
if args.evalfile:
    res = self.train(nepoch=nepochs, test=True)
    res.writecsv(args.evalfile)
else:
    self.train(nepoch=nepochs, test=False)

if args.msavfile:
    self.savemodel(args.msavfile)
else:
    self.savemodel("model.pt")

if args.evalfile == False:
    print("Post-training test ...")
    loss = self.getLoss()
    print('Loss Results:', loss)

if args.predictionfile:
    pred = self.getPred()
    ob = PredObject(pred)
    ob.setHeaders(self.test_dataset._columns)
    ob.writecsv(args.predictionfile)

def getArgs():
    parser = argparse.ArgumentParser(
        prog='CosmoML test script (cuda version)',
        description='Train and test a regression model',
        epilog='')

    parser.add_argument('-t', '--train',
                        help="Training data set")
    parser.add_argument('-T', '--test',
                        help="Testing data set")
    parser.add_argument('-i', '--imagedir',
                        help="Image directory")
    parser.add_argument('-I', '--imagedirtest',
                        help='Test image directory')
    parser.add_argument('-o', '--evalfile',
```

## A. Appendix Cosmo-ML code

```
        help="Filename for evaluation output")
parser.add_argument('-p', '--predictionfile',
                    help="Filename for prediction output")
parser.add_argument('-e', '--epochs',
                    help="Testing data set")
parser.add_argument('-W', '--weights',
                    help="File with pre trained weights")
parser.add_argument('-s', '--msavefile',
                    help="File for trained weights, end in .pt")
parser.add_argument('-a', '--amp6', action='store_true',
                    help="Estimate amplitude with 6 parameters")

return parser.parse_args()

if __name__ == "__main__":
    print("MLSystem test script.\nConfiguring ... ")

    args = getArgs()
    ml = MLSystem()
    ml.systemTest(args)
```

## A. Appendix Cosmo-ML code

The following code below is from `cuadaaux.py`:

```
import torch

def cudaDiagnostic():
    print(f"CUDA Availability:          {torch.cuda.is_available()}")
    print(f"CUDA version:                 {torch.version.cuda}")
    cuda_id = torch.cuda.current_device()
    print(f"ID of current CUDA device:   {cuda_id}")
    print(f"Name of current CUDA device: {torch.cuda.get_device_name(cuda_id)}")
```

## A. Appendix Cosmo-ML code

The following code below is from Dataset.py

```
"""
The Dataset class manages loading and access to a dataset
in the CosmoAI model.
"""

import torch
import os
import numpy as np
from skimage import io
from torch.utils.data import Dataset
import pandas as pd

class CosmoDataset(Dataset):
    """CosmoAI dataset."""
    _columns = ["x", "y", "einsteinR", "sigma", "sigma2", "theta"]

    def getSlice(self):
        return self._columns

    def getDim(self):
        return len(self.getSlice())

    def __init__(self, csvfile, imgdir=".", imgdirtest='.', columns=None):
        """
        Args:
            csvfile (string): Path to the csv file with annotations.
            imgdir (string): Directory with all the images.
        """
        self.frame = pd.read_csv(csvfile)
        self.imgdir = imgdir
        self.imgdirtest = imgdirtest
        if columns != None:
            self._columns = columns

    def __len__(self):
        return len(self.frame)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
```

## A. Appendix Cosmo-ML code

```
fn = os.path.join(self.imgdir, self.imgdirtest,
                  self.frame.iloc[idx, 1])
image = io.imread(fn)[np.newaxis, :, :].astype(np.float32) / 255
image = torch.from_numpy(image)
targets = self.frame.loc[idx, self.getSlice()]
targets = np.array(targets).astype(np.float32)
targets = torch.from_numpy(targets)
index = int(self.frame.loc[idx, "index"])

return (image, targets, index)

class CosmoDataset1(CosmoDataset):
    #_columns = ["R", "phi", "einsteinR", "sigma"]
    #_columns = ["R", "phi", "einsteinR", "sigma", "sigma2", "theta"]
    _columns = ["x", "y", "einsteinR", "sigma", "sigma2", "theta"]
class CosmoDataset2(CosmoDataset):
    _columns = ["alpha[1][0]", "alpha[1][2]", "beta[1][2]", "alpha[2][1]", "beta[2][1]",
```



## A. Appendix Cosmo-ML code

The following code below is from EvalObject.py:

```
#!/usr/bin/env python

import pandas as pd
from skimage import io
import torch
import numpy as np

class EvalObject:
    def __init__(self, ev):
        """The `EvalObject` is instantiated with a list of dict objects,
        where each dict object is the output from `MLSystem.trainOne()`. """
        tr = [x["training"] for x in ev]
        tt = [x["test"] for x in ev]
        self.loss = torch.tensor([x["loss"] for x in ev])
        self.loss.reshape((len(self.loss), 1))
        self.testMSELoss = torch.stack([x["TotalMSE"] for x in tt])
        self.testMAELoss = torch.stack([x["TotalMAE"] for x in tt])
        self.testErrorMean = torch.stack([x["ErrorMean"] for x in tt])
        self.testErrorStDev = torch.stack([x["ErrorStDev"] for x in tt])
        self.testAbsMean = torch.stack([x["AbsMean"] for x in tt])
        self.testAbsStDev = torch.stack([x["AbsStDev"] for x in tt])

        self.trainingMSELoss = torch.stack([x["TotalMSE"] for x in tr])
        self.trainingMAELoss = torch.stack([x["TotalMAE"] for x in tr])
        self.trainingErrorMean = torch.stack([x["ErrorMean"] for x in tr])
        self.trainingErrorStDev = torch.stack([x["ErrorStDev"] for x in tr])
        self.trainingAbsMean = torch.stack([x["AbsMean"] for x in tr])
        self.trainingAbsStDev = torch.stack([x["AbsStDev"] for x in tr])
        self._mat = None
        self._headers = ["Loss"]
    def getMatrix(self):
        """Return the evaluation statistics as a numpy `array`."""
        if self._mat == None:
            loss = self.loss.numpy()
            s = loss.shape
            if len(s) == 1:
                loss = loss.reshape((s[0], 1))
            trainingMSELoss = self.trainingMSELoss.cpu().numpy().reshape((s[0], 1))
            trainingMAELoss = self.trainingMAELoss.cpu().numpy().reshape((s[0], 1))
            testMSELoss = self.testMSELoss.cpu().numpy().reshape((s[0], 1))
            testMAELoss = self.testMAELoss.cpu().numpy().reshape((s[0], 1))
            ls = [loss]
```

## A. Appendix Cosmo-ML code

```

ls.append(trainingMSELoss)
ls.append(trainingMAELoss)
ls.append(testMSELoss)
ls.append(testMAELoss)
ls.append(self.testErrorMean.cpu().numpy())
ls.append(self.testErrorStDev.cpu().numpy())
ls.append(self.testAbsMean.cpu().numpy())
ls.append(self.testAbsStDev.cpu().numpy())
ls.append(self.trainingErrorMean.cpu().numpy())
ls.append(self.trainingErrorStDev.cpu().numpy())
ls.append(self.trainingAbsMean.cpu().numpy())
ls.append(self.trainingAbsStDev.cpu().numpy())
self._mat = np.hstack(ls)
return self._mat
def setHeaders(self,h):
    """Set the headers for CSV output.
    The input should be the same list of labels that are
    used by `CosmoDataset` to define the columns."""
    ls = (["tLoss"] +
        ["tMSELoss (Training Set)"] +
        ["tMAELoss (Training Set)"] +
        ["tMSELoss (Test Set)"] +
        ["tMAELoss (Test Set)"] +
        [x + " (Error Mean - Test Set)" for x in h] +
        [x + " (Error StDev - Test Set)" for x in h] +
        [x + " (AbsError Mean - Test Set)" for x in h] +
        [x + " (AbsError StDev - Test Set)" for x in h] +
        [x + " (Error Mean - Training Set)" for x in h] +
        [x + " (Error StDev - Training Set)" for x in h] +
        [x + " (AbsError Mean - Training Set)" for x in h] +
        [x + " (AbsError StDev - Training Set)" for x in h])
    self._headers = dict(enumerate(ls))
def writecsv(self,fn="eval.csv"):
    "Write the evaluation results to the given CSV file."
    m = self.getMatrix()
    csv = pd.DataFrame(m)
    csv.rename(columns=self._headers, inplace=True)
    csv.to_csv(fn)

class PredObject:
    def __init__(self,pred):
        """The `PredObject` is instantiated with the output from
        `MLSystem.getPred()`. """

```

## A. Appendix Cosmo-ML code

```
    self._mat = pred.cpu().numpy()
    self._headers = {0: "Index"}
def getMatrix(self):
    "Return the evaluation statistics as a numpy `array`."
    return self._mat
def setHeaders(self,h):
    """Set the headers for CSV output.
    The input should be the same list of labels that are
    used by `CosmoDataset` to define the columns."""
    ls = ["Index"] +
        [x + " (Ground Truth)" for x in h] +
        [x + " (Predicted)" for x in h]
    self._headers = dict(enumerate(ls))
def writecsv(self,fn="pred.csv"):
    "Write the evaluation results to the given CSV file."
    m = self.getMatrix()
    csv = pd.DataFrame(m)
    csv.rename(columns=self._headers, inplace=True)
    csv.to_csv(fn)
```

## A. Appendix Cosmo-ML code

The code below is for running code on IDUN. The following code below is from the VGG\_exampel.py file.

```
import torch
from MLSystem import MLSystem, getArgs
import cudaaux

class CudaModel(MLSystem):
    def __init__(self,model='vgg',criterion=None,optimizer=None,nepoch=1,learning_rate=None):
        super().__init__(model,criterion,optimizer,nepoch,learning_rate)

        if not torch.cuda.is_available():
            raise Exception( "CUDA is not available" )

        self.device = torch.device( "cuda" )
        self.model = self.model.to(self.device)
        if args.weights:
            self.model.load_state_dict(torch.load(args.weights))

if __name__ == "__main__":
    args = getArgs()

    print( "CudaModel (CosmoML) test script." )
    cudaaux.cudaDiagnostic()
    print( "Configuring ... " )

    ml = CudaModel()
    ml.systemTest(args)
```

## A. Appendix Cosmo-ML code

The code below is for running code on IDUN. The following code below is from the VGG.slurm file.

```
#!/bin/sh
#SBATCH --partition=GPUQ           # Use a GPU
#SBATCH --account=ie-idi
#SBATCH --time=63:00:00          # Max wait time
#SBATCH --nodes=1
#SBATCH -c4                      # Number of cores
#SBATCH --gres=gpu:1            # Require 1 GPU
#SBATCH --mem-per-gpu=15G       # Require certain amount of GPU memory
#SBATCH --constraint="A100"     # Don't have to use A100
#SBATCH --job-name="CosmoML_VGG"
#SBATCH --output=VGG.out
#SBATCH --mail-user=modestas@stud.ntnu.no
#SBATCH --mail-type=ALL

WORKDIR=${SLURM_SUBMIT_DIR}
cd ${WORKDIR}
echo "Job Name:           $SLURM_JOB_NAME"
echo "Working directory: $SLURM_SUBMIT_DIR"
echo "Job ID:             $SLURM_JOB_ID"
echo "Nodes used:        $SLURM_JOB_NODELIST"
echo "Number of nodes:   $SLURM_JOB_NUM_NODES"
echo "Cores (per node):  $SLURM_CPUS_ON_NODE"
echo "Total cores:       $SLURM_NTASKS"

# module load torchvision/0.8.2-fosscuda-2020b-PyTorch-1.7.1
# module load PyTorch/1.8.1-fosscuda-2020b
# module swap NCCL/2.8.3-CUDA-11.1.1 NCCL/2.8.3-GCCcore-10.2.0-CUDA-11.1.1
# module swap PyTorch/1.7.1-fosscuda-2020b PyTorch/1.8.1-fosscuda-2020b
# module load scikit-learn/0.23.2-fosscuda-2020b

module purge
module load SciPy-bundle/2021.10-foss-2021b
module list

source /cluster/work/modestas/CosmoML/venv/bin/activate

PH=/cluster/work/modestas/CosmoML/src/
export PYTHONPATH=$PH:$PYTHONPATH

time python3 $PH/VGG_example.py -t "/cluster/work/modestas/train100k/train.csv" -i "
```

## A. Appendix Cosmo-ML code

The code below is for all networks used in this project. The highlighted lines in the code is what we have contributed. They're taken from [https://github.com/pytorch/vision/blob/main/torchvision](https://github.com/pytorch/vision/blob/main/torchvision/models/alexnet.py). The following code below is from the AlexNet.py file.

```
import torch
import torch.nn as nn

class AlexNet(nn.Module):
    def __init__(self, num_outputs: int = 1000, dropout: float = 0.5, extra_layers: b
        super().__init__()
        #_log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        if extra_layers:
            self.classifier = nn.Sequential(
                nn.Dropout(p=dropout),
                nn.Linear(256 * 6 * 6, 4096),
                nn.ReLU(inplace=True),
                nn.Dropout(p=dropout),
                nn.Linear(4096, 4096),
                nn.ReLU(inplace=True),
                nn.Linear(4096, 1024), # Adding two additional fully connected
                nn.ReLU(inplace=True), # layers.
                nn.Linear(1024, 512), #
                nn.ReLU(inplace=True),
                nn.Linear(512, num_outputs),)
        else:
            self.classifier = nn.Sequential(
```

## A. Appendix Cosmo-ML code

```
nn.Dropout(p=dropout),
nn.Linear(256 * 6 * 6, 4096),
nn.ReLU(inplace=True),
nn.Dropout(p=dropout),
nn.Linear(4096, 4096),
nn.ReLU(inplace=True),
nn.Linear(4096, num_outputs),)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    x = self.features(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.classifier(x)
    return x
```

## A. Appendix Cosmo-ML code

The following code below is from the Inception3.py file.

```
import os
import shutil
import warnings
from typing import Optional, List, Callable, Tuple

import torch
from torch import Tensor
import torch.nn as nn
from torchvision import models

from torchvision.models import InceptionOutputs
from torchvision.models.inception import BasicConv2d, InceptionB, InceptionD, \
    InceptionAux, InceptionA, InceptionC, InceptionE

class Inception3(nn.Module):
    def __init__(
        self,
        num_outputs: int = 4,
        aux_logits: bool = True,
        transform_input: bool = False,
        inception_blocks: Optional[List[Callable[..., nn.Module]]] = None,
        init_weights: Optional[bool] = True,
        dropout: float = 0.5,
        extra_layers: bool = False,
    ) -> None:
        super().__init__()
        #_log_api_usage_once(self)
        if inception_blocks is None:
            inception_blocks = [BasicConv2d, InceptionA, InceptionB,
                               InceptionC, InceptionD, InceptionE, InceptionAux]
        if len(inception_blocks) != 7:
            raise ValueError(f"length of inception_blocks should be 7 instead of {len(
conv_block = inception_blocks[0]
inception_a = inception_blocks[1]
inception_b = inception_blocks[2]
inception_c = inception_blocks[3]
inception_d = inception_blocks[4]
inception_e = inception_blocks[5]
```



## A. Appendix Cosmo-ML code

```

inception_aux = inception_blocks[6]

self.extra_layers = extra_layers
self.aux_logits = aux_logits
self.transform_input = transform_input
self.Conv2d_1a_3x3 = conv_block(1, 32, kernel_size=3, stride=2)
self.Conv2d_2a_3x3 = conv_block(32, 32, kernel_size=3)
self.Conv2d_2b_3x3 = conv_block(32, 64, kernel_size=3, padding=1)
self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2)
self.Conv2d_3b_1x1 = conv_block(64, 80, kernel_size=1)
self.Conv2d_4a_3x3 = conv_block(80, 192, kernel_size=3)
self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2)
self.Mixed_5b = inception_a(192, pool_features=32)
self.Mixed_5c = inception_a(256, pool_features=64)
self.Mixed_5d = inception_a(288, pool_features=64)
self.Mixed_6a = inception_b(288)
self.Mixed_6b = inception_c(768, channels_7x7=128)
self.Mixed_6c = inception_c(768, channels_7x7=160)
self.Mixed_6d = inception_c(768, channels_7x7=160)
self.Mixed_6e = inception_c(768, channels_7x7=192)
self.AuxLogits: Optional[nn.Module] = None
if aux_logits:
    self.AuxLogits = inception_aux(768, num_outputs)
self.Mixed_7a = inception_d(768)
self.Mixed_7b = inception_e(1280)
self.Mixed_7c = inception_e(2048)
self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
self.dropout = nn.Dropout(p=dropout)
if extra_layers:
    self.fc = nn.Sequential(
        nn.Linear(2048, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, num_outputs))
else:
    self.fc = nn.Linear(2048, num_outputs)
if init_weights:
    for m in self.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            stddev = float(m.stddev) if hasattr(m, "stddev") else 0.1 # type
            torch.nn.init.trunc_normal_(m.weight, mean=0.0, std=stddev, a=-2,
        elif isinstance(m, nn.BatchNorm2d):
            nn.init.constant_(m.weight, 1)

```

## A. Appendix Cosmo-ML code

```
nn.init.constant_(m.bias, 0)

def _transform_input(self, x: Tensor) -> Tensor:
    if self.transform_input:
        x_ch0 = torch.unsqueeze(x[:, 0], 1) * (0.229 / 0.5) + (0.485 - 0.5) / 0.5
        x_ch1 = torch.unsqueeze(x[:, 1], 1) * (0.224 / 0.5) + (0.456 - 0.5) / 0.5
        x_ch2 = torch.unsqueeze(x[:, 2], 1) * (0.225 / 0.5) + (0.406 - 0.5) / 0.5
        x = torch.cat((x_ch0, x_ch1, x_ch2), 1)
    return x

def _forward(self, x: Tensor) -> Tuple[Tensor, Optional[Tensor]]:
    # N x 3 x 299 x 299
    x = self.Conv2d_1a_3x3(x)
    # N x 32 x 149 x 149
    x = self.Conv2d_2a_3x3(x)
    # N x 32 x 147 x 147
    x = self.Conv2d_2b_3x3(x)
    # N x 64 x 147 x 147
    x = self.maxpool1(x)
    # N x 64 x 73 x 73
    x = self.Conv2d_3b_1x1(x)
    # N x 80 x 73 x 73
    x = self.Conv2d_4a_3x3(x)
    # N x 192 x 71 x 71
    x = self.maxpool2(x)
    # N x 192 x 35 x 35
    x = self.Mixed_5b(x)
    # N x 256 x 35 x 35
    x = self.Mixed_5c(x)
    # N x 288 x 35 x 35
    x = self.Mixed_5d(x)
    # N x 288 x 35 x 35
    x = self.Mixed_6a(x)
    # N x 768 x 17 x 17
    x = self.Mixed_6b(x)
    # N x 768 x 17 x 17
    x = self.Mixed_6c(x)
    # N x 768 x 17 x 17
    x = self.Mixed_6d(x)
    # N x 768 x 17 x 17
    x = self.Mixed_6e(x)
    # N x 768 x 17 x 17
    aux: Optional[Tensor] = None
    if self.AuxLogits is not None:
```

## A. Appendix Cosmo-ML code

```
        if self.training:
            aux = self.AuxLogits(x)
# N x 768 x 17 x 17
x = self.Mixed_7a(x)
# N x 1280 x 8 x 8
x = self.Mixed_7b(x)
# N x 2048 x 8 x 8
x = self.Mixed_7c(x)
# N x 2048 x 8 x 8
# Adaptive average pooling
x = self.avgpool(x)
# N x 2048 x 1 x 1
x = self.dropout(x)
# N x 2048 x 1 x 1
x = torch.flatten(x, 1)
# N x 2048
x = self.fc(x)
# N x 1000 (num_outputs)
return x, aux

@torch.jit.unused
def eager_outputs(self, x: Tensor, aux: Optional[Tensor]) -> InceptionOutputs:
    if self.training and self.aux_logits:
        return InceptionOutputs(x, aux)
    else:
        return x # type: ignore[return-value]

def forward(self, x: Tensor) -> InceptionOutputs:
    #x = self._transform_input(x)
    x, aux = self._forward(x)
    return x
    #aux_defined = self.training and self.aux_logits
    #if torch.jit.is_scripting():
    #    if not aux_defined:
    #        warnings.warn("Scripted Inception3 always returns Inception3 Tuple")
    #    return InceptionOutputs(x, aux)
    #else:
    #    return self.eager_outputs(x, aux)
```

## A. Appendix Cosmo-ML code

The following code below is from the SqueezeNet.py file.

```
from functools import partial
from typing import Any, Optional

import torch
import torch.nn as nn
import torch.nn.init as init

from torchvision.models.squeezenet import Fire

class SqueezeNet(nn.Module):
    def __init__(
        self,
        version: str = "1_0",
        num_outputs: int = 6,
        dropout: float = 0.5,
        extra_layers: bool = False
    ) -> None:
        super().__init__()
        self.num_outputs = num_outputs
        self.extra_layers = extra_layers
        if version == "1_0":
            self.features = nn.Sequential(
                nn.Conv2d(1, 96, kernel_size=7, stride=2),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
                Fire(96, 16, 64, 64),
                Fire(128, 16, 64, 64),
                Fire(128, 32, 128, 128),
                nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
                Fire(256, 32, 128, 128),
                Fire(256, 48, 192, 192),
                Fire(384, 48, 192, 192),
                Fire(384, 64, 256, 256),
                nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
                Fire(512, 64, 256, 256),
            )
        elif version == "1_1":
            self.features = nn.Sequential(
                nn.Conv2d(1, 64, kernel_size=3, stride=2),
```

## A. Appendix Cosmo-ML code

```

nn.ReLU(inplace=True),
nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
Fire(64, 16, 64, 64),
Fire(128, 16, 64, 64),
nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
Fire(128, 32, 128, 128),
Fire(256, 32, 128, 128),
nn.MaxPool2d(kernel_size=3, stride=2, ceil_mode=True),
Fire(256, 48, 192, 192),
Fire(384, 48, 192, 192),
Fire(384, 64, 256, 256),
Fire(512, 64, 256, 256),
)
else:
    raise ValueError(f"Unsupported SqueezeNet version {version}: 1_0 or 1_1 e

# Final convolution is initialized differently from the rest
if extra_layers:
    final_conv = nn.Conv2d(512, 256, kernel_size=3, padding=1)
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout),
        final_conv,
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2),
        nn.Flatten(),
        nn.Linear(1024 * 6 * 6, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, self.num_outputs)
    )
else:
    final_conv = nn.Conv2d(512, self.num_outputs, kernel_size=1)
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout),
        final_conv,
        nn.ReLU(inplace=True),
        nn.AdaptiveAvgPool2d((1, 1))
    )

for m in self.modules():
    if isinstance(m, nn.Conv2d):
        if m is final_conv:
            init.normal_(m.weight, mean=0.0, std=0.01)

```

## A. Appendix Cosmo-ML code

```
        else:
            init.kaiming_uniform_(m.weight)
            if m.bias is not None:
                init.constant_(m.bias, 0)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    x = self.features(x)
    x = self.classifier(x)
    if self.extra_layers==False:
        x = torch.flatten(x, 1)
    return x

def _squeezenet(version: str, **kwargs: Any) -> SqueezeNet:
    model = SqueezeNet(version, **kwargs)
    return model

def squeezenet1_0(
    *, progress: bool = True, **kwargs: Any
) -> SqueezeNet:
    """SqueezeNet model architecture from the `SqueezeNet: AlexNet-level
    accuracy with 50x fewer parameters and <0.5MB model size
    <https://arxiv.org/abs/1602.07360>`_ paper.
    Args:
        **kwargs: parameters passed to the ``torchvision.models.squeezenet.SqueezeNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/squeezenet.py>`_
            for more details about this class.
    """
    return _squeezenet("1_0", **kwargs)

def squeezenet1_1(
    *, progress: bool = True, **kwargs: Any
) -> SqueezeNet:
    """SqueezeNet 1.1 model from the `official SqueezeNet repo
    <https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet\_v1.1>`.
    SqueezeNet 1.1 has 2.4x less computation and slightly fewer parameters
    than SqueezeNet 1.0, without sacrificing accuracy.
    Args:
        **kwargs: parameters passed to the ``torchvision.models.squeezenet.SqueezeNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/squeezenet.py>`_
            for more details about this class.
    """
```

*A. Appendix Cosmo-ML code*

```
return _squeezenet("1_1", **kwargs)
```

## A. Appendix Cosmo-ML code

The following code below is from the ResNet.py file.

```
from functools import partial
from typing import Any, Callable, List, Optional, Type, Union

import torch
import torch.nn as nn
from torch import Tensor

from torchvision.models.resnet import BasicBlock, Bottleneck, conv1x1
from torchvision.models._utils import _overwrite_named_param, handle_legacy_interface

class ResNet(nn.Module):
    def __init__(
        self,
        block: Type[Union[BasicBlock, Bottleneck]],
        layers: List[int],
        num_outputs: int = 4,
        zero_init_residual: bool = False,
        groups: int = 1,
        width_per_group: int = 64,
        replace_stride_with_dilation: Optional[List[bool]] = None,
        norm_layer: Optional[Callable[..., nn.Module]] = None,
        extra_layers: bool = False,
    ) -> None:
        super().__init__()
        #_log_api_usage_once(self)
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        self._norm_layer = norm_layer
        self.extra_layers = extra_layers

        self.inplanes = 64
        self.dilation = 1
        if replace_stride_with_dilation is None:
            # each element in the tuple indicates if we should replace
            # the 2x2 stride with a dilated convolution instead
            replace_stride_with_dilation = [False, False, False]
        if len(replace_stride_with_dilation) != 3:
            raise ValueError(
                "replace_stride_with_dilation should be None "
                f"or a 3-element tuple, got {replace_stride_with_dilation}"
            )
```



## A. Appendix Cosmo-ML code

```

    )
self.groups = groups
self.base_width = width_per_group
self.conv1 = nn.Conv2d(1, self.inplanes, kernel_size=7, stride=2, padding=3,
self.bn1 = norm_layer(self.inplanes)
self.relu = nn.ReLU(inplace=True)
self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
self.layer1 = self._make_layer(block, 64, layers[0])
self.layer2 = self._make_layer(block, 128, layers[1], stride=2, dilate=replac
self.layer3 = self._make_layer(block, 256, layers[2], stride=2, dilate=replac
self.layer4 = self._make_layer(block, 512, layers[3], stride=2, dilate=replac
self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
if self.extra_layers:
    self.fc = nn.Sequential(
        nn.Linear(512 * block.expansion, 256),
        nn.ReLU(inplace=True),
        nn.Linear(256, 128),
        nn.ReLU(inplace=True),
        nn.Linear(128, num_outputs)
    )
else:
    self.fc = nn.Linear(512 * block.expansion, num_outputs)

for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode="fan_out", nonlinearity="relu")
    elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

# Zero-initialize the last BN in each residual branch,
# so that the residual branch starts with zeros, and each residual block be
# This improves the model by 0.2~0.3% according to https://arxiv.org/abs/17
if zero_init_residual:
    for m in self.modules():
        if isinstance(m, Bottleneck) and m.bn3.weight is not None:
            nn.init.constant_(m.bn3.weight, 0) # type: ignore[arg-type]
        elif isinstance(m, BasicBlock) and m.bn2.weight is not None:
            nn.init.constant_(m.bn2.weight, 0) # type: ignore[arg-type]

def _make_layer(
    self,
    block: Type[Union[BasicBlock, Bottleneck]],
    planes: int,

```

## A. Appendix Cosmo-ML code

```
blocks: int,
stride: int = 1,
dilate: bool = False,
) -> nn.Sequential:
    norm_layer = self._norm_layer
    downsample = None
    previous_dilation = self.dilation
    if dilate:
        self.dilation *= stride
        stride = 1
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride),
            norm_layer(planes * block.expansion),
        )

    layers = []
    layers.append(
        block(
            self.inplanes, planes, stride, downsample, self.groups, self.base_wid
        )
    )
    self.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(
            block(
                self.inplanes,
                planes,
                groups=self.groups,
                base_width=self.base_width,
                dilation=self.dilation,
                norm_layer=norm_layer,
            )
        )

    return nn.Sequential(*layers)

def _forward_impl(self, x: Tensor) -> Tensor:
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)
```

## A. Appendix Cosmo-ML code

```
x = self.layer1(x)
x = self.layer2(x)
x = self.layer3(x)
x = self.layer4(x)

x = self.avgpool(x)
x = torch.flatten(x, 1)
x = self.fc(x)

return x

def forward(self, x: Tensor) -> Tensor:
    return self._forward_impl(x)

def _resnet(
    block: Type[Union[BasicBlock, Bottleneck]],
    layers: List[int],
    **kwargs: Any,
) -> ResNet:
    model = ResNet(block, layers, **kwargs)
    return model

@handle_legacy_interface()
def resnet18(*, progress: bool = False, **kwargs: Any) -> ResNet:
    """ResNet-18 from `Deep Residual Learning for Image Recognition <https://arxiv.
    Args:
        progress (bool, optional): If True, displays a progress bar of the
            download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.resnet.ResNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.
            for more details about this class.
    .. autoclass:: torchvision.models.ResNet18_Weights
       :members:
    """

    return _resnet(BasicBlock, [2, 2, 2, 2], **kwargs)

def resnet34(*, progress: bool = True, **kwargs: Any) -> ResNet:
    """ResNet-34 from `Deep Residual Learning for Image Recognition <https://arxiv.
    .. note::
        The bottleneck of TorchVision places the stride for downsampling to the seco
```

## A. Appendix Cosmo-ML code

```
convolution while the original paper places it to the first 1x1 convolution.
This variant improves the accuracy and is known as `ResNet V1.5
<https://ngc.nvidia.com/catalog/model-scripts/nvidia:resnet\_50\_v1\_5\_for\_pyto
Args:
    progress (bool, optional): If True, displays a progress bar of the
        download to stderr. Default is True.
    **kwargs: parameters passed to the ``torchvision.models.resnet.ResNet``
        base class. Please refer to the `source code
        <https://github.com/pytorch/vision/blob/main/torchvision/models/resnet
        for more details about this class.

.. autoclass:: torchvision.models.ResNet34_Weights
    :members:
    """
return _resnet(BasicBlock, [3, 4, 6, 3], **kwargs)

def resnet50(*, progress: bool = True, **kwargs: Any) -> ResNet:
    return _resnet(Bottleneck, [3, 4, 6, 3], **kwargs)

def resnet101(*, progress: bool = True, **kwargs: Any) -> ResNet:
    return _resnet(Bottleneck, [3, 4, 23, 3], **kwargs)

def resnet152(*, progress: bool = True, **kwargs: Any) -> ResNet:
    return _resnet(Bottleneck, [3, 8, 36, 3], **kwargs)

def resnext50_32x4d(*, progress: bool = True, **kwargs: Any) -> ResNet:
    """ResNeXt-50 32x4d model from
    `Aggregated Residual Transformation for Deep Neural Networks <https://arxiv.org
    """
    _overwrite_named_param(kwargs, "groups", 32)
    _overwrite_named_param(kwargs, "width_per_group", 4)
    return _resnet(Bottleneck, [3, 4, 6, 3], **kwargs)

def resnext101_32x8d(*, progress: bool = True, **kwargs: Any) -> ResNet:
    _overwrite_named_param(kwargs, "groups", 32)
    _overwrite_named_param(kwargs, "width_per_group", 8)
    return _resnet(Bottleneck, [3, 4, 23, 3], **kwargs)

def resnext101_64x4d(*, progress: bool = True, **kwargs: Any) -> ResNet:
    _overwrite_named_param(kwargs, "groups", 64)
    _overwrite_named_param(kwargs, "width_per_group", 4)
    return _resnet(Bottleneck, [3, 4, 23, 3], **kwargs)
```

## A. Appendix Cosmo-ML code

```
def wide_resnet50_2(*, progress: bool = True, **kwargs: Any) -> ResNet:
    """Wide ResNet-50-2 model from
    `Wide Residual Networks <https://arxiv.org/abs/1605.07146>`.
    The model is the same as ResNet except for the bottleneck number of channels
    which is twice larger in every block. The number of channels in outer 1x1
    convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048
    channels, and in Wide ResNet-50-2 has 2048-1024-2048.
    """
    _overwrite_named_param(kwargs, "width_per_group", 64 * 2)
    return _resnet(Bottleneck, [3, 4, 6, 3], **kwargs)

def wide_resnet101_2(*, progress: bool = True, **kwargs: Any) -> ResNet:
    _overwrite_named_param(kwargs, "width_per_group", 64 * 2)
    return _resnet(Bottleneck, [3, 4, 23, 3], **kwargs)
```

## A. Appendix Cosmo-ML code

The following code below is from the DenseNet.py file.

```
import re
from collections import OrderedDict
from functools import partial
from typing import Any, List, Optional, Tuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.checkpoint as cp
from torch import Tensor

from torchvision.models._utils import handle_legacy_interface
from torchvision.models.densenet import _DenseLayer, _DenseBlock, _Transition

class DenseNet(nn.Module):
    r"""Densenet-BC model class, based on
    `Densely Connected Convolutional Networks` <https://arxiv.org/pdf/1608.06993.p
    Args:
        growth_rate (int) - how many filters to add each layer (`k` in paper)
        block_config (list of 4 ints) - how many layers in each pooling block
        num_init_features (int) - the number of filters to learn in the first convo
        bn_size (int) - multiplicative factor for number of bottle neck layers
            (i.e. bn_size * k features in the bottleneck layer)
        drop_rate (float) - dropout rate after each dense layer
        num_classes (int) - number of classification classes
        memory_efficient (bool) - If True, uses checkpointing. Much more memory eff
            but slower. Default: *False*. See `paper` <https://arxiv.org/pdf/1707.06
    """
    def __init__(
        self,
        growth_rate: int = 32,
        block_config: Tuple[int, int, int, int] = (6, 12, 24, 16),
        num_init_features: int = 64,
        bn_size: int = 4,
        drop_rate: float = 0,
        num_outputs: int = 4,
        memory_efficient: bool = False,
        extra_layers: bool = False,
    ) -> None:
```

## A. Appendix Cosmo-ML code

```

super().__init__()
#_log_api_usage_once(self)

# First convolution
self.features = nn.Sequential(
    OrderedDict(
        [
            ("conv0", nn.Conv2d(1, num_init_features, kernel_size=7, stride=2, padding=3)),
            ("norm0", nn.BatchNorm2d(num_init_features)),
            ("relu0", nn.ReLU(inplace=True)),
            ("pool0", nn.MaxPool2d(kernel_size=3, stride=2, padding=1)),
        ]
    )
)

# Each denseblock
num_features = num_init_features
for i, num_layers in enumerate(block_config):
    block = _DenseBlock(
        num_layers=num_layers,
        num_input_features=num_features,
        bn_size=bn_size,
        growth_rate=growth_rate,
        drop_rate=drop_rate,
        memory_efficient=memory_efficient,
    )
    self.features.add_module("denseblock%d" % (i + 1), block)
    num_features = num_features + num_layers * growth_rate
    if i != len(block_config) - 1:
        trans = _Transition(num_input_features=num_features, num_output_features=num_features // 2)
        self.features.add_module("transition%d" % (i + 1), trans)
        num_features = num_features // 2

# Final batch norm
self.features.add_module("norm5", nn.BatchNorm2d(num_features))

# Linear layer
if extra_layers:
    self.classifier = nn.Sequential(
        nn.Linear(num_features, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, 256),
        nn.ReLU(inplace=True),
    )

```

## A. Appendix Cosmo-ML code

```
        nn.Linear(256, num_outputs),
    )
else:
    self.classifier = nn.Linear(num_features, num_outputs)

# Official init from torch repo.
for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight)
    elif isinstance(m, nn.BatchNorm2d):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)
    elif isinstance(m, nn.Linear):
        nn.init.constant_(m.bias, 0)

def forward(self, x: Tensor) -> Tensor:
    features = self.features(x)
    out = F.relu(features, inplace=True)
    out = F.adaptive_avg_pool2d(out, (1, 1))
    out = torch.flatten(out, 1)
    out = self.classifier(out)
    return out

def _densenet(
    growth_rate: int,
    block_config: Tuple[int, int, int, int],
    num_init_features: int,
    **kwargs: Any,
) -> DenseNet:
    model = DenseNet(growth_rate, block_config, num_init_features, **kwargs)
    return model

@handle_legacy_interface()
def densenet121(*, progress: bool = True, **kwargs: Any) -> DenseNet:
    r"""Densenet-121 model from
    `Densely Connected Convolutional Networks <https://arxiv.org/abs/1608.06993>`_.

    Args:
        progress (bool, optional): If True, displays a progress bar of the download
        **kwargs: parameters passed to the ``torchvision.models.densenet.DenseNet``
            base class. Please refer to the `source code`
```



## A. Appendix Cosmo-ML code

*<<https://github.com/pytorch/vision/blob/main/torchvision/models/densenet>  
for more details about this class.*

```
.. autoclass:: torchvision.models.DenseNet121_Weights  
   :members:  
   """
```

```
return _densenet(32, (6, 12, 24, 16), 64, **kwargs)
```

```
@handle_legacy_interface()
```

```
def densenet161(*, progress: bool = True, **kwargs: Any) -> DenseNet:  
    return _densenet(48, (6, 12, 36, 24), 96, **kwargs)
```

```
@handle_legacy_interface()
```

```
def densenet169(*, progress: bool = True, **kwargs: Any) -> DenseNet:  
    return _densenet(32, (6, 12, 32, 32), 64, **kwargs)
```

```
@handle_legacy_interface()
```

```
def densenet201(*, progress: bool = True, **kwargs: Any) -> DenseNet:  
    return _densenet(32, (6, 12, 48, 32), 64, **kwargs)
```

## A. Appendix Cosmo-ML code

The following code below is from the EfficientNet.py file.

```
import copy
import math
from dataclasses import dataclass
from functools import partial
from typing import Any, Callable, Dict, List, Optional, Sequence, Tuple, Union

import torch
from torch import nn, Tensor

from torchvision.ops.misc import Conv2dNormActivation
from torchvision.models._utils import handle_legacy_interface
from torchvision.models.efficientnet import MBConvConfig, FusedMBConvConfig, _MBConvConfig

class EfficientNet(nn.Module):
    def __init__(
        self,
        inverted_residual_setting: Sequence[Union[MBConvConfig, FusedMBConvConfig]],
        dropout: float,
        stochastic_depth_prob: float = 0.2,
        num_outputs: int = 4,
        norm_layer: Optional[Callable[..., nn.Module]] = None,
        last_channel: Optional[int] = None,
        extra_layers: bool = False,
    ) -> None:
        """
        EfficientNet V1 and V2 main class

        Args:
            inverted_residual_setting (Sequence[Union[MBConvConfig, FusedMBConvConfig]]):
            dropout (float): The dropout probability
            stochastic_depth_prob (float): The stochastic depth probability
            num_classes (int): Number of classes
            norm_layer (Optional[Callable[..., nn.Module]]): Module specifying the
            last_channel (int): The number of channels on the penultimate layer
        """
        super().__init__()

        if not inverted_residual_setting:
            raise ValueError("The inverted_residual_setting should not be empty")
        elif not (
            isinstance(inverted_residual_setting, Sequence)

```

## A. Appendix Cosmo-ML code

```

    and all([isinstance(s, _MBConvConfig) for s in inverted_residual_setting]
):
    raise TypeError("The inverted_residual_setting should be List[MBConvConfig]")

if norm_layer is None:
    norm_layer = nn.BatchNorm2d

layers: List[nn.Module] = []

# building first layer
firstconv_output_channels = inverted_residual_setting[0].input_channels
layers.append(
    Conv2dNormActivation(
        1, firstconv_output_channels, kernel_size=3, stride=2, norm_layer=norm_layer
    )
)

# building inverted residual blocks
total_stage_blocks = sum(cnf.num_layers for cnf in inverted_residual_setting)
stage_block_id = 0
for cnf in inverted_residual_setting:
    stage: List[nn.Module] = []
    for _ in range(cnf.num_layers):
        # copy to avoid modifications. shallow copy is enough
        block_cnf = copy.copy(cnf)

        # overwrite info if not the first conv in the stage
        if stage:
            block_cnf.input_channels = block_cnf.out_channels
            block_cnf.stride = 1

        # adjust stochastic depth probability based on the depth of the stage
        sd_prob = stochastic_depth_prob * float(stage_block_id) / total_stage_blocks

        stage.append(block_cnf.block(block_cnf, sd_prob, norm_layer))
        stage_block_id += 1

    layers.append(nn.Sequential(*stage))

# building last several layers
lastconv_input_channels = inverted_residual_setting[-1].out_channels
lastconv_output_channels = last_channel if last_channel is not None else 4 *
layers.append(
    Conv2dNormActivation(

```

## A. Appendix Cosmo-ML code

```
        lastconv_input_channels,
        lastconv_output_channels,
        kernel_size=1,
        norm_layer=norm_layer,
        activation_layer=nn.SiLU,
    )
)

self.features = nn.Sequential(*layers)
self.avgpool = nn.AdaptiveAvgPool2d(1)
if extra_layers:
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout, inplace=True),
        nn.Linear(lastconv_output_channels, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, 256),
        nn.ReLU(inplace=True),
        nn.Linear(256, num_outputs),)
else:
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout, inplace=True),
        nn.Linear(lastconv_output_channels, num_outputs),
    )

for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode="fan_out")
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
        nn.init.ones_(m.weight)
        nn.init.zeros_(m.bias)
    elif isinstance(m, nn.Linear):
        init_range = 1.0 / math.sqrt(m.out_features)
        nn.init.uniform_(m.weight, -init_range, init_range)
        nn.init.zeros_(m.bias)

def _forward_impl(self, x: Tensor) -> Tensor:
    x = self.features(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)

    x = self.classifier(x)
```

## A. Appendix Cosmo-ML code

```

    return x

def forward(self, x: Tensor) -> Tensor:
    return self._forward_impl(x)

def _efficientnet(
    inverted_residual_setting: Sequence[Union[MBCConvConfig, FusedMBCConvConfig]],
    dropout: float,
    last_channel: Optional[int],
    **kwargs: Any,
) -> EfficientNet:
    model = EfficientNet(inverted_residual_setting, dropout, last_channel=last_channel)
    return model

def _efficientnet_conf(
    arch: str,
    **kwargs: Any,
) -> Tuple[Sequence[Union[MBCConvConfig, FusedMBCConvConfig]], Optional[int]]:
    inverted_residual_setting: Sequence[Union[MBCConvConfig, FusedMBCConvConfig]]
    if arch.startswith("efficientnet_b"):
        bneck_conf = partial(MBCConvConfig, width_mult=kwargs.pop("width_mult"), depth_mult=1)
        inverted_residual_setting = [
            bneck_conf(1, 3, 1, 32, 16, 1),
            bneck_conf(6, 3, 2, 16, 24, 2),
            bneck_conf(6, 5, 2, 24, 40, 2),
            bneck_conf(6, 3, 2, 40, 80, 3),
            bneck_conf(6, 5, 1, 80, 112, 3),
            bneck_conf(6, 5, 2, 112, 192, 4),
            bneck_conf(6, 3, 1, 192, 320, 1),
        ]
        last_channel = None
    elif arch.startswith("efficientnet_v2_s"):
        inverted_residual_setting = [
            FusedMBCConvConfig(1, 3, 1, 24, 24, 2),
            FusedMBCConvConfig(4, 3, 2, 24, 48, 4),
            FusedMBCConvConfig(4, 3, 2, 48, 64, 4),
            MBCConvConfig(4, 3, 2, 64, 128, 6),
            MBCConvConfig(6, 3, 1, 128, 160, 9),
            MBCConvConfig(6, 3, 2, 160, 256, 15),
        ]
        last_channel = 1280

```

## A. Appendix Cosmo-ML code

```

elif arch.startswith("efficientnet_v2_m"):
    inverted_residual_setting = [
        FusedMBConvConfig(1, 3, 1, 24, 24, 3),
        FusedMBConvConfig(4, 3, 2, 24, 48, 5),
        FusedMBConvConfig(4, 3, 2, 48, 80, 5),
        MBConvConfig(4, 3, 2, 80, 160, 7),
        MBConvConfig(6, 3, 1, 160, 176, 14),
        MBConvConfig(6, 3, 2, 176, 304, 18),
        MBConvConfig(6, 3, 1, 304, 512, 5),
    ]
    last_channel = 1280
elif arch.startswith("efficientnet_v2_l"):
    inverted_residual_setting = [
        FusedMBConvConfig(1, 3, 1, 32, 32, 4),
        FusedMBConvConfig(4, 3, 2, 32, 64, 7),
        FusedMBConvConfig(4, 3, 2, 64, 96, 7),
        MBConvConfig(4, 3, 2, 96, 192, 10),
        MBConvConfig(6, 3, 1, 192, 224, 19),
        MBConvConfig(6, 3, 2, 224, 384, 25),
        MBConvConfig(6, 3, 1, 384, 640, 7),
    ]
    last_channel = 1280
else:
    raise ValueError(f"Unsupported model type {arch}")

return inverted_residual_setting, last_channel

def efficientnet_b0(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    """EfficientNet B0 model architecture from the `EfficientNet: Rethinking Model
    Neural Networks <https://arxiv.org/abs/1905.11946>`_ paper.

    Args:
        progress (bool, optional): If True, displays a progress bar of the
            download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.efficientnet.EfficientNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/efficientnet.py>`
            for more details about this class.
    .. autoclass:: torchvision.models.EfficientNet_BO_Weights
       :members:
    """

    inverted_residual_setting, last_channel = _efficientnet_conf(

```

## A. Appendix Cosmo-ML code

```
    "efficientnet_b0", width_mult=1.0, depth_mult=1.0)
return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.2), last_

def efficientnet_b1(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b1", width_mult=1.0, depth_mult=1.1)
    return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.2), last_

def efficientnet_b2(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b2", width_mult=1.1, depth_mult=1.2)
    return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.3), last_

def efficientnet_b3(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b3", width_mult=1.2, depth_mult=1.4)
    return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.3), last_

def efficientnet_b3_5(*, progress: bool = True, **kwargs: Any) -> EfficientNet: # my
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b3_5", width_mult=1.2, depth_mult=1.6)
    return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.3), last_

def efficientnet_b4(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b4", width_mult=1.4, depth_mult=1.8)
    return _efficientnet(inverted_residual_setting, kwargs.pop("dropout", 0.4), last_

def efficientnet_b4_5(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b4_5", width_mult=1.8, depth_mult=2.0)
    return _efficientnet(
        inverted_residual_setting,
        kwargs.pop("dropout", 0.4),
        last_channel,
        norm_layer=partial(nn.BatchNorm2d, eps=0.001, momentum=0.01),
        **kwargs,)

def efficientnet_b5(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf(
        "efficientnet_b5", width_mult=1.6, depth_mult=2.2)
    return _efficientnet(
        inverted_residual_setting,
```

## A. Appendix Cosmo-ML code

```
kwargs.pop("dropout", 0.4),
last_channel,
norm_layer=partial(nn.BatchNorm2d, eps=0.001, momentum=0.01),
**kwargs,
)

def efficientnet_b6(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf("efficientnet_b6", w
    return _efficientnet(
        inverted_residual_setting,
        kwargs.pop("dropout", 0.5),
        last_channel,
        norm_layer=partial(nn.BatchNorm2d, eps=0.001, momentum=0.01),
        **kwargs,
    )

def efficientnet_b7(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf("efficientnet_b7", w
    return _efficientnet(
        inverted_residual_setting,
        kwargs.pop("dropout", 0.5),
        last_channel,
        norm_layer=partial(nn.BatchNorm2d, eps=0.001, momentum=0.01),
        **kwargs,
    )

def efficientnet_v2_s(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf("efficientnet_v2_s")
    return _efficientnet(
        inverted_residual_setting,
        kwargs.pop("dropout", 0.2),
        last_channel,
        norm_layer=partial(nn.BatchNorm2d, eps=1e-03),
        **kwargs,
    )

def efficientnet_v2_m(*, progress: bool = True, **kwargs: Any) -> EfficientNet:
    inverted_residual_setting, last_channel = _efficientnet_conf("efficientnet_v2_m")
    return _efficientnet(
        inverted_residual_setting,
        kwargs.pop("dropout", 0.3),
        last_channel,
        norm_layer=partial(nn.BatchNorm2d, eps=1e-03),
        **kwargs,
    )
```



## A. Appendix Cosmo-ML code

```
)  
  
def efficientnet_v2_1(*, progress: bool = True, **kwargs: Any) -> EfficientNet:  
    inverted_residual_setting, last_channel = _efficientnet_conf("efficientnet_v2_1")  
    return _efficientnet(  
        inverted_residual_setting,  
        kwargs.pop("dropout", 0.4),  
        last_channel,  
        norm_layer=partial(nn.BatchNorm2d, eps=1e-03),  
        **kwargs,  
    )
```

## A. Appendix Cosmo-ML code

The following code below is from the MnasNet.py file.

```
import warnings
from functools import partial
from typing import Any, Dict, List, Optional

import torch
import torch.nn as nn
from torch import Tensor

from torchvision.models._api import register_model
from torchvision.models._utils import _overwrite_named_param, handle_legacy_interface
from torchvision.models.mnasnet import _stack, _round_to_multiple_of, _get_depths, _

# Paper suggests 0.9997 momentum, for TensorFlow. Equivalent PyTorch momentum is
# 1.0 - tensorflow.
_BN_MOMENTUM = 1 - 0.9997

class MNASNet(torch.nn.Module):
    """MNASNet, as described in https://arxiv.org/pdf/1807.11626.pdf. This
    implements the B1 variant of the model.
    >>> model = MNASNet(1.0, num_outputs=1000)
    >>> x = torch.rand(1, 3, 224, 224)
    >>> y = model(x)
    >>> y.dim()
    2
    >>> y.nelement()
    1000
    """

    # Version 2 adds depth scaling in the initial stages of the network.
    _version = 2

    def __init__(
        self,
        alpha: float,
        num_outputs: int = 6,
        dropout: float = 0.2,
        extra_layers: bool = False,
    ) -> None:
        super().__init__()
        if alpha <= 0.0:
```

## A. Appendix Cosmo-ML code

```

    raise ValueError(f"alpha should be greater than 0.0 instead of {alpha}")
self.alpha = alpha
self.num_outputs = num_outputs
depths = _get_depths(alpha)
layers = [
    # First layer: regular conv.
    nn.Conv2d(1, depths[0], 3, padding=1, stride=2, bias=False),
    nn.BatchNorm2d(depths[0], momentum=_BN_MOMENTUM),
    nn.ReLU(inplace=True),
    # Depthwise separable, no skip.
    nn.Conv2d(depths[0], depths[0], 3, padding=1, stride=1, groups=depths[0]),
    nn.BatchNorm2d(depths[0], momentum=_BN_MOMENTUM),
    nn.ReLU(inplace=True),
    nn.Conv2d(depths[0], depths[1], 1, padding=0, stride=1, bias=False),
    nn.BatchNorm2d(depths[1], momentum=_BN_MOMENTUM),
    # MNASNet blocks: stacks of inverted residuals.
    _stack(depths[1], depths[2], 3, 2, 3, 3, _BN_MOMENTUM),
    _stack(depths[2], depths[3], 5, 2, 3, 3, _BN_MOMENTUM),
    _stack(depths[3], depths[4], 5, 2, 6, 3, _BN_MOMENTUM),
    _stack(depths[4], depths[5], 3, 1, 6, 2, _BN_MOMENTUM),
    _stack(depths[5], depths[6], 5, 2, 6, 4, _BN_MOMENTUM),
    _stack(depths[6], depths[7], 3, 1, 6, 1, _BN_MOMENTUM),
    # Final mapping to classifier input.
    nn.Conv2d(depths[7], 1280, 1, padding=0, stride=1, bias=False),
    nn.BatchNorm2d(1280, momentum=_BN_MOMENTUM),
    nn.ReLU(inplace=True),
]
self.layers = nn.Sequential(*layers)
if extra_layers:
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout, inplace=True),
        nn.Linear(1280, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, num_outputs))
else:
    self.classifier = nn.Sequential(
        nn.Dropout(p=dropout, inplace=True),
        nn.Linear(1280, num_outputs))

for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode="fan_out", nonlinearity="relu")

```

## A. Appendix Cosmo-ML code

```
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.BatchNorm2d):
        nn.init.ones_(m.weight)
        nn.init.zeros_(m.bias)
    elif isinstance(m, nn.Linear):
        nn.init.kaiming_uniform_(m.weight, mode="fan_out", nonlinearity="sigmoid")
        nn.init.zeros_(m.bias)

def forward(self, x: Tensor) -> Tensor:
    x = self.layers(x)
    # Equivalent to global avgpool and removing H and W dimensions.
    x = x.mean([2, 3])
    return self.classifier(x)

def _load_from_state_dict(
    self,
    state_dict: Dict,
    prefix: str,
    local_metadata: Dict,
    strict: bool,
    missing_keys: List[str],
    unexpected_keys: List[str],
    error_msgs: List[str],
) -> None:
    version = local_metadata.get("version", None)
    if version not in [1, 2]:
        raise ValueError(f"version should be set to 1 or 2 instead of {version}")

    if version == 1 and not self.alpha == 1.0:
        # In the initial version of the model (v1), stem was fixed-size.
        # All other layer configurations were the same. This will patch
        # the model so that it's identical to v1. Model with alpha 1.0 is
        # unaffected.
        depths = _get_depths(self.alpha)
        v1_stem = [
            nn.Conv2d(3, 32, 3, padding=1, stride=2, bias=False),
            nn.BatchNorm2d(32, momentum=BN_MOMENTUM),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 32, 3, padding=1, stride=1, groups=32, bias=False),
            nn.BatchNorm2d(32, momentum=BN_MOMENTUM),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 16, 1, padding=0, stride=1, bias=False),
            nn.BatchNorm2d(16, momentum=BN_MOMENTUM),
```

## A. Appendix Cosmo-ML code

```

        _stack(16, depths[2], 3, 2, 3, 3, _BN_MOMENTUM),
    ]
    for idx, layer in enumerate(v1_stem):
        self.layers[idx] = layer

    # The model is now identical to v1, and must be saved as such.
    self._version = 1
    warnings.warn(
        "A new version of MNASNet model has been implemented. "
        "Your checkpoint was saved using the previous version. "
        "This checkpoint will load and work as before, but "
        "you may want to upgrade by training a newer model or "
        "transfer learning from an updated ImageNet checkpoint.",
        UserWarning,
    )

    super()._load_from_state_dict(
        state_dict, prefix, local_metadata, strict, missing_keys, unexpected_keys
    )

def _mnasnet(alpha: float, **kwargs: Any) -> MNASNet:
    model = MNASNet(alpha, **kwargs)
    return model

def mnasnet1_0(*, progress: bool = True, **kwargs: Any) -> MNASNet:
    """MNASNet with depth multiplier of 0.5 from
    `MnasNet: Platform-Aware Neural Architecture Search for Mobile
    <https://arxiv.org/pdf/1807.11626.pdf>`_ paper.
    Args:
        **kwargs: parameters passed to the ``torchvision.models.mnasnet.MNASNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/mnasnet
            for more details about this class.
    .. autoclass:: torchvision.models.MNASNet0_5_Weights
       :members:
    """
    return _mnasnet(1.0, **kwargs)

def mnasnet3_8(*, progress: bool = True, **kwargs: Any) -> MNASNet:
    return _mnasnet(3.8, **kwargs)

def mnasnet6_0(*, progress: bool = True, **kwargs: Any) -> MNASNet:
    return _mnasnet(6, **kwargs)

```

## A. Appendix Cosmo-ML code

The following code below is from the VGG.py file.

```
from functools import partial
from typing import Any, cast, Dict, List, Optional, Union

import torch
import torch.nn as nn

from torchvision.models._utils import handle_legacy_interface

class VGG(nn.Module):
    def __init__(
        self, features: nn.Module, num_outputs: int = 6, init_weights: bool = True, d
    ) -> None:
        super().__init__()
        #_log_api_usage_once(self)
        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, num_outputs),
        )
        if init_weights:
            for m in self.modules():
                if isinstance(m, nn.Conv2d):
                    nn.init.kaiming_normal_(m.weight, mode="fan_out", nonlinearity="r
                    if m.bias is not None:
                        nn.init.constant_(m.bias, 0)
                elif isinstance(m, nn.BatchNorm2d):
                    nn.init.constant_(m.weight, 1)
                    nn.init.constant_(m.bias, 0)
                elif isinstance(m, nn.Linear):
                    nn.init.normal_(m.weight, 0, 0.01)
                    nn.init.constant_(m.bias, 0)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
```

## A. Appendix Cosmo-ML code

```

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.classifier(x)
    return x

def make_layers(cfg: List[Union[str, int]], batch_norm: bool = False) -> nn.Sequential:
    layers: List[nn.Module] = []
    in_channels = 1
    for v in cfg:
        if v == "M":
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            v = cast(int, v)
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)

cfgs: Dict[str, List[Union[str, int]]] = {
    "A": [64, "M", 128, "M", 256, 256, "M", 512, 512, "M", 512, 512, "M"],
    "B": [64, 64, "M", 128, 128, "M", 256, 256, "M", 512, 512, "M", 512, 512, "M"],
    "D": [64, 64, "M", 128, 128, "M", 256, 256, 256, "M", 512, 512, 512, "M", 512, 512],
    "E": [64, 64, "M", 128, 128, "M", 256, 256, 256, 256, "M", 512, 512, 512, 512, "M"]
}

def _vgg(cfg: str, batch_norm: bool, **kwargs: Any) -> VGG:
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm), **kwargs)
    return model

@handle_legacy_interface()
def vgg11(*, progress: bool = True, **kwargs: Any) -> VGG:
    """VGG-11 from `Very Deep Convolutional Networks for Large-Scale Image Recognition`
    Args:
        progress (bool, optional): If True, displays a progress bar of the
            download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.vgg.VGG``
            base class. Please refer to the `source code`
    """

```

## A. Appendix Cosmo-ML code

*<<https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py>>  
for more details about this class.*

```
.. autoclass:: torchvision.models.VGG11_Weights
    :members:
    """
return _vgg("A", False, **kwargs)

@handle_legacy_interface()
def vgg11_bn(*, progress: bool = True, **kwargs: Any) -> VGG:
    """VGG-11-BN from `Very Deep Convolutional Networks for Large-Scale Image Recog
    Args:
        progress (bool, optional): If True, displays a progress bar of the
            download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.vgg.VGG``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py>
            for more details about this class.

    .. autoclass:: torchvision.models.VGG11_BN_Weights
        :members:
        """
return _vgg("A", True, **kwargs)

@handle_legacy_interface()
def vgg13(*, progress: bool = True, **kwargs: Any) -> VGG:
    return _vgg("B", False, **kwargs)

@handle_legacy_interface()
def vgg13_bn(*, progress: bool = True, **kwargs: Any) -> VGG:

    return _vgg("B", True, **kwargs)

@handle_legacy_interface()
def vgg16(*, progress: bool = True, **kwargs: Any) -> VGG:
    return _vgg("D", False, **kwargs)

@handle_legacy_interface()
def vgg16_bn(*, progress: bool = True, **kwargs: Any) -> VGG:
    return _vgg("D", True, **kwargs)

@handle_legacy_interface()
def vgg19(*, progress: bool = True, **kwargs: Any) -> VGG:
    return _vgg("E", False, **kwargs)
```



## A. Appendix Cosmo-ML code

```
@handle_legacy_interface()
def vgg19_bn(*, progress: bool = True, **kwargs: Any) -> VGG:
    return _vgg("E", True, **kwargs)
```

## A. Appendix Cosmo-ML code

The following code below is from the Vision\_Transformer.py file.

```
import math
from collections import OrderedDict
from functools import partial
from typing import Any, Callable, Dict, List, NamedTuple, Optional

import torch
import torch.nn as nn

from torchvision.ops.misc import Conv2dNormActivation, MLP
from torchvision.models.vision_transformer import ConvStemConfig, \
    MLPBlock, EncoderBlock, Encoder, interpolate_embeddings

class VisionTransformer(nn.Module):
    """Vision Transformer as per https://arxiv.org/abs/2010.11929."""

    def __init__(
        self,
        image_size: int,
        patch_size: int,
        num_layers: int,
        num_heads: int,
        hidden_dim: int,
        mlp_dim: int,
        dropout: float = 0.0,
        attention_dropout: float = 0.0,
        num_outputs: int = 1000,
        norm_layer: Callable[..., torch.nn.Module] = partial(nn.LayerNorm, eps=1e-6),
        conv_stem_configs: Optional[List[ConvStemConfig]] = None,
        extra_layers: bool = False,
    ):
        super().__init__()
        torch._assert(image_size % patch_size == 0, "Input shape indivisible by patch_size")
        self.image_size = image_size
        self.patch_size = patch_size
        self.hidden_dim = hidden_dim
        self.mlp_dim = mlp_dim
        self.attention_dropout = attention_dropout
        self.dropout = dropout
        self.num_outputs = num_outputs
        self.norm_layer = norm_layer
```

## A. Appendix Cosmo-ML code

```
if conv_stem_configs is not None:
    # As per https://arxiv.org/abs/2106.14881
    seq_proj = nn.Sequential()
    prev_channels = 3
    for i, conv_stem_layer_config in enumerate(conv_stem_configs):
        seq_proj.add_module(
            f"conv_bn_relu_{i}",
            Conv2dNormActivation(
                in_channels=prev_channels,
                out_channels=conv_stem_layer_config.out_channels,
                kernel_size=conv_stem_layer_config.kernel_size,
                stride=conv_stem_layer_config.stride,
                norm_layer=conv_stem_layer_config.norm_layer,
                activation_layer=conv_stem_layer_config.activation_layer,
            ),
        )
        prev_channels = conv_stem_layer_config.out_channels
    seq_proj.add_module(
        "conv_last", nn.Conv2d(in_channels=prev_channels, out_channels=hidden_dim)
    )
    self.conv_proj: nn.Module = seq_proj
else:
    self.conv_proj = nn.Conv2d(
        in_channels=1, out_channels=hidden_dim, kernel_size=patch_size, stride=1
    )

seq_length = (image_size // patch_size) ** 2

# Add a class token
self.class_token = nn.Parameter(torch.zeros(1, 1, hidden_dim))
seq_length += 1

self.encoder = Encoder(
    seq_length,
    num_layers,
    num_heads,
    hidden_dim,
    mlp_dim,
    dropout,
    attention_dropout,
    norm_layer,
)
self.seq_length = seq_length
```

## A. Appendix Cosmo-ML code

```

heads_layers: OrderedDict[str, nn.Module] = OrderedDict()

if extra_layers:
    heads_layers["pre_logits1"] = nn.Linear(hidden_dim, 512)
    heads_layers["ac1"] = nn.ReLU()
    heads_layers["pre_logits2"] = nn.Linear(512, 256)
    heads_layers["act2"] = nn.ReLU()
    heads_layers["head"] = nn.Linear(256, num_outputs)
    self.heads = nn.Sequential(heads_layers)
else:
    heads_layers["head"] = nn.Linear(hidden_dim, num_outputs)
    self.heads = nn.Sequential(heads_layers)

if isinstance(self.conv_proj, nn.Conv2d):
    # Init the patchify stem
    fan_in = self.conv_proj.in_channels * self.conv_proj.kernel_size[0] * self.conv_proj.kernel_size[1]
    nn.init.trunc_normal_(self.conv_proj.weight, std=math.sqrt(1 / fan_in))
    if self.conv_proj.bias is not None:
        nn.init.zeros_(self.conv_proj.bias)
elif self.conv_proj.conv_last is not None and isinstance(self.conv_proj.conv_last, nn.Conv2d):
    # Init the last 1x1 conv of the conv stem
    nn.init.normal_(
        self.conv_proj.conv_last.weight, mean=0.0, std=math.sqrt(2.0 / self.conv_proj.in_channels)
    )
    if self.conv_proj.conv_last.bias is not None:
        nn.init.zeros_(self.conv_proj.conv_last.bias)

if hasattr(self.heads, "pre_logits") and isinstance(self.heads.pre_logits, nn.Linear):
    fan_in = self.heads.pre_logits.in_features
    nn.init.trunc_normal_(self.heads.pre_logits.weight, std=math.sqrt(1 / fan_in))
    nn.init.zeros_(self.heads.pre_logits.bias)

if isinstance(self.heads.head, nn.Linear):
    nn.init.zeros_(self.heads.head.weight)
    nn.init.zeros_(self.heads.head.bias)

def _process_input(self, x: torch.Tensor) -> torch.Tensor:
    n, c, h, w = x.shape
    p = self.patch_size
    torch._assert(h == self.image_size, f"Wrong image height! Expected {self.image_size} but got {h}")
    torch._assert(w == self.image_size, f"Wrong image width! Expected {self.image_size} but got {w}")
    n_h = h // p
    n_w = w // p

```

## A. Appendix Cosmo-ML code

```
# (n, c, h, w) -> (n, hidden_dim, n_h, n_w)
x = self.conv_proj(x)
# (n, hidden_dim, n_h, n_w) -> (n, hidden_dim, (n_h * n_w))
x = x.reshape(n, self.hidden_dim, n_h * n_w)

# (n, hidden_dim, (n_h * n_w)) -> (n, (n_h * n_w), hidden_dim)
# The self attention layer expects inputs in the format (N, S, E)
# where S is the source sequence length, N is the batch size, E is the
# embedding dimension
x = x.permute(0, 2, 1)

return x

def forward(self, x: torch.Tensor):
    # Reshape and permute the input tensor
    x = self._process_input(x)
    n = x.shape[0]

    # Expand the class token to the full batch
    batch_class_token = self.class_token.expand(n, -1, -1)
    x = torch.cat([batch_class_token, x], dim=1)

    x = self.encoder(x)

    # Classifier "token" as used by standard language architectures
    x = x[:, 0]

    x = self.heads(x)

    return x

def _vision_transformer(
    patch_size: int,
    num_layers: int,
    num_heads: int,
    hidden_dim: int,
    mlp_dim: int,
    **kwargs: Any,
) -> VisionTransformer:
    image_size = kwargs.pop("image_size", 400)
    model = VisionTransformer(
        image_size=image_size,
```

## A. Appendix Cosmo-ML code

```
        patch_size=patch_size,
        num_layers=num_layers,
        num_heads=num_heads,
        hidden_dim=hidden_dim,
        mlp_dim=mlp_dim,
        **kwargs,
    )

    return model

def vit_b_16(*, progress: bool = True, **kwargs: Any) -> VisionTransformer:
    """
    Constructs a vit_b_16 architecture from
    `An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale <ht
    Args:
        **kwargs: parameters passed to the ``torchvision.models.vision_transformer.
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/vision\_
            for more details about this class.
    .. autoclass:: torchvision.models.ViT_B_16_Weights
       :members:
    """
    return _vision_transformer(
        patch_size=16,
        num_layers=12,
        num_heads=12,
        hidden_dim=768,
        mlp_dim=3072,
        **kwargs,
    )

def vit_b_32(*, progress: bool = True, **kwargs: Any) -> VisionTransformer:
    return _vision_transformer(
        patch_size=32,
        num_layers=12,
        num_heads=12,
        hidden_dim=768,
        mlp_dim=3072,
        **kwargs,
    )

def vit_l_16(*, progress: bool = True, **kwargs: Any) -> VisionTransformer:
```

## A. Appendix Cosmo-ML code

```
return _vision_transformer(  
    patch_size=16,  
    num_layers=24,  
    num_heads=16,  
    hidden_dim=1024,  
    mlp_dim=4096,  
    **kwargs,  
)  
  
def vit_l_32(*, progress: bool = True, **kwargs: Any) -> VisionTransformer:  
    return _vision_transformer(  
        patch_size=32,  
        num_layers=24,  
        num_heads=16,  
        hidden_dim=1024,  
        mlp_dim=4096,  
        **kwargs,  
    )  
  
def vit_h_14(*, progress: bool = True, **kwargs: Any) -> VisionTransformer:  
    return _vision_transformer(  
        patch_size=14,  
        num_layers=32,  
        num_heads=16,  
        hidden_dim=1280,  
        mlp_dim=5120,  
        **kwargs,  
    )
```

## A. Appendix Cosmo-ML code

The following code below is from the Swin\_Transformer.py file.

```
import math
from functools import partial
from typing import Any, Callable, List, Optional

import torch
import torch.nn.functional as F
from torch import nn, Tensor

from torchvision.ops.misc import MLP, Permute
from torchvision.models.swin_transformer import _get_relative_position_bias, \
    _patch_merging_pad, PatchMerging, PatchMergingV2, shifted_window_attention, \
    ShiftedWindowAttention, ShiftedWindowAttentionV2, SwinTransformerBlock, \
    SwinTransformerBlockV2

class SwinTransformer(nn.Module):
    """
    Implements Swin Transformer from the "Swin Transformer: Hierarchical Vision Tr
    Shifted Windows" <https://arxiv.org/pdf/2103.14030>_ paper.
    Args:
        patch_size (List[int]): Patch size.
        embed_dim (int): Patch embedding dimension.
        depths (List(int)): Depth of each Swin Transformer layer.
        num_heads (List(int)): Number of attention heads in different layers.
        window_size (List[int]): Window size.
        mlp_ratio (float): Ratio of mlp hidden dim to embedding dim. Default: 4.0.
        dropout (float): Dropout rate. Default: 0.0.
        attention_dropout (float): Attention dropout rate. Default: 0.0.
        stochastic_depth_prob (float): Stochastic depth rate. Default: 0.1.
        num_outputs (int): Number of outputs for classification head. Default: 1000
        block (nn.Module, optional): SwinTransformer Block. Default: None.
        norm_layer (nn.Module, optional): Normalization layer. Default: None.
        downsample_layer (nn.Module): Downsample layer (patch merging). Default: Pa
    """
    def __init__(
        self,
        patch_size: List[int],
        embed_dim: int,
        depths: List[int],
```



## A. Appendix Cosmo-ML code

```

num_heads: List[int],
window_size: List[int],
mlp_ratio: float = 4.0,
dropout: float = 0.0,
attention_dropout: float = 0.0,
stochastic_depth_prob: float = 0.1,
num_outputs: int = 1000,
norm_layer: Optional[Callable[..., nn.Module]] = None,
block: Optional[Callable[..., nn.Module]] = None,
downsample_layer: Callable[..., nn.Module] = PatchMerging,
extra_layers: bool = False,
):
    super().__init__()
    self.num_outputs = num_outputs

    if block is None:
        block = SwinTransformerBlock
    if norm_layer is None:
        norm_layer = partial(nn.LayerNorm, eps=1e-5)

    layers: List[nn.Module] = []
    # split image into non-overlapping patches
    layers.append(
        nn.Sequential(
            nn.Conv2d(
                1, embed_dim, kernel_size=(patch_size[0], patch_size[1]), stride=
            ),
            Permute([0, 2, 3, 1]),
            norm_layer(embed_dim),
        )
    )

    total_stage_blocks = sum(depths)
    stage_block_id = 0
    # build SwinTransformer blocks
    for i_stage in range(len(depths)):
        stage: List[nn.Module] = []
        dim = embed_dim * 2**i_stage
        for i_layer in range(depths[i_stage]):
            # adjust stochastic depth probability based on the depth of the sta
            sd_prob = stochastic_depth_prob * float(stage_block_id) / (total_stag
            stage.append(
                block(
                    dim,

```

## A. Appendix Cosmo-ML code

```

        num_heads[i_stage],
        window_size=window_size,
        shift_size=[0 if i_layer % 2 == 0 else w // 2 for w in window_size],
        mlp_ratio=mlp_ratio,
        dropout=dropout,
        attention_dropout=attention_dropout,
        stochastic_depth_prob=sd_prob,
        norm_layer=norm_layer,
    )
    )
    stage_block_id += 1
    layers.append(nn.Sequential(*stage))
    # add patch merging layer
    if i_stage < (len(depths) - 1):
        layers.append(downsample_layer(dim, norm_layer))
self.features = nn.Sequential(*layers)

num_features = embed_dim * 2 ** (len(depths) - 1)
self.norm = norm_layer(num_features)
self.permute = Permute([0, 3, 1, 2]) # B H W C -> B C H W
self.avgpool = nn.AdaptiveAvgPool2d(1)
self.flatten = nn.Flatten(1)
if extra_layers:
    self.head = nn.Sequential(
        nn.Linear(num_features, num_features/2),
        nn.ReLU(inplace=True),
        nn.Linear(num_features/2, num_features/4),
        nn.ReLU(inplace=True),
        nn.Linear(num_features/4, num_outputs),
    )
else:
    self.head = nn.Linear(num_features, num_outputs)

for m in self.modules():
    if isinstance(m, nn.Linear):
        nn.init.trunc_normal_(m.weight, std=0.02)
        if m.bias is not None:
            nn.init.zeros_(m.bias)

def forward(self, x):
    x = self.features(x)
    x = self.norm(x)
    x = self.permute(x)
    x = self.avgpool(x)

```

## A. Appendix Cosmo-ML code

```
x = self.flatten(x)
x = self.head(x)
return x

def _swin_transformer(
    patch_size: List[int],
    embed_dim: int,
    depths: List[int],
    num_heads: List[int],
    window_size: List[int],
    stochastic_depth_prob: float,
    **kwargs: Any,
) -> SwinTransformer:
    model = SwinTransformer(
        patch_size=patch_size,
        embed_dim=embed_dim,
        depths=depths,
        num_heads=num_heads,
        window_size=window_size,
        stochastic_depth_prob=stochastic_depth_prob,
        **kwargs,
    )
    return model

def swin_t(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    """
    Constructs a swin_tiny architecture from
    `Swin Transformer: Hierarchical Vision Transformer using Shifted Windows <https://arxiv.org/abs/2103.14030>`
    Args:
        **kwargs: parameters passed to the ``torchvision.models.swin_transformer.SwinTransformer``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/swin_transformer.py>`
            for more details about this class.
    .. autoclass:: torchvision.models.Swin_T_Weights
       :members:
    """
    return _swin_transformer(
        patch_size=[4, 4],
        embed_dim=96,
        depths=[2, 2, 6, 2],
        num_heads=[3, 6, 12, 24],
        window_size=[7, 7],
        stochastic_depth_prob=0.2,
```

## A. Appendix Cosmo-ML code

```
        **kwargs,
    )

def swin_s(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    return _swin_transformer(
        patch_size=[4, 4],
        embed_dim=96,
        depths=[2, 2, 18, 2],
        num_heads=[3, 6, 12, 24],
        window_size=[7, 7],
        stochastic_depth_prob=0.3,
        **kwargs,
    )

def swin_b(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    return _swin_transformer(
        patch_size=[4, 4],
        embed_dim=128,
        depths=[2, 2, 18, 2],
        num_heads=[4, 8, 16, 32],
        window_size=[7, 7],
        stochastic_depth_prob=0.5,
        **kwargs,
    )

def swin_v2_t(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    return _swin_transformer(
        patch_size=[4, 4],
        embed_dim=96,
        depths=[2, 2, 6, 2],
        num_heads=[3, 6, 12, 24],
        window_size=[8, 8],
        stochastic_depth_prob=0.2,
        block=SwinTransformerBlockV2,
        downsample_layer=PatchMergingV2,
        **kwargs,
    )

def swin_v2_s(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    return _swin_transformer(
```

## A. Appendix Cosmo-ML code

```
    patch_size=[4, 4],
    embed_dim=96,
    depths=[2, 2, 18, 2],
    num_heads=[3, 6, 12, 24],
    window_size=[8, 8],
    stochastic_depth_prob=0.3,
    block=SwinTransformerBlockV2,
    downsample_layer=PatchMergingV2,
    **kwargs,
)
```

```
def swin_v2_b(*, progress: bool = True, **kwargs: Any) -> SwinTransformer:
    return _swin_transformer(
        patch_size=[4, 4],
        embed_dim=128,
        depths=[2, 2, 18, 2],
        num_heads=[4, 8, 16, 32],
        window_size=[8, 8],
        stochastic_depth_prob=0.5,
        block=SwinTransformerBlockV2,
        downsample_layer=PatchMergingV2,
        **kwargs,
    )
```

## B. Appendix CosmoSim code

To add cropping of data set pictures below changes have to be made. Add these code lines to Image.py file:

```
def cropImage(im):
    m,n = im.shape
    if m > 400 and n > 400:
        x = (m - 400) / 2
        y = (n - 400) / 2
        im = im[int(x):int(x+400),int(y):int(y+400)]
    return im
```

Add two last lines below after the first two lines of code in datagen.py file.

```
if args.reflines:      # This is in datagen.py
    drawAxes(im)      # This is in datagen.py
if args.crop:
    im = cropImage(im)
```

Add two lines below after all other parse arguments in datagen.py file.

```
parser.add_argument('-q', '--crop', action='store_true',
                    help="Cropping of the image to 400x400")
```

## C. Other results

This appendix consists of all the rest of the results gathered in this project. None of it was deemed interesting enough to put in the main report for one reason or another. It is gathered here for documentation purposes, or in case the all results are needed later.

For all the graphs showing total loss (MAE or MSE), orange lines always indicate test data set performance, and blue lines always indicate training set performance.

C. Other results

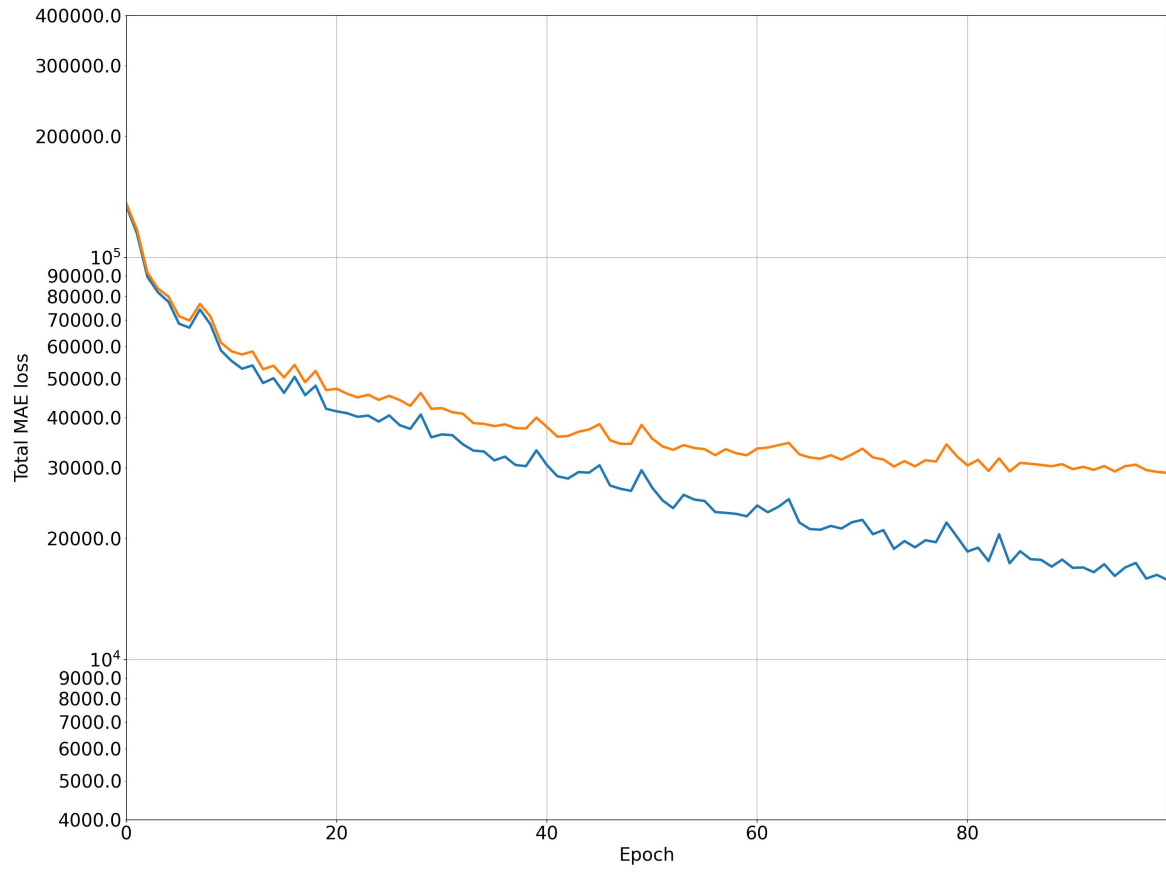


Figure C.1.: AlexNet total loss performance with logarithmic graph



C. Other results

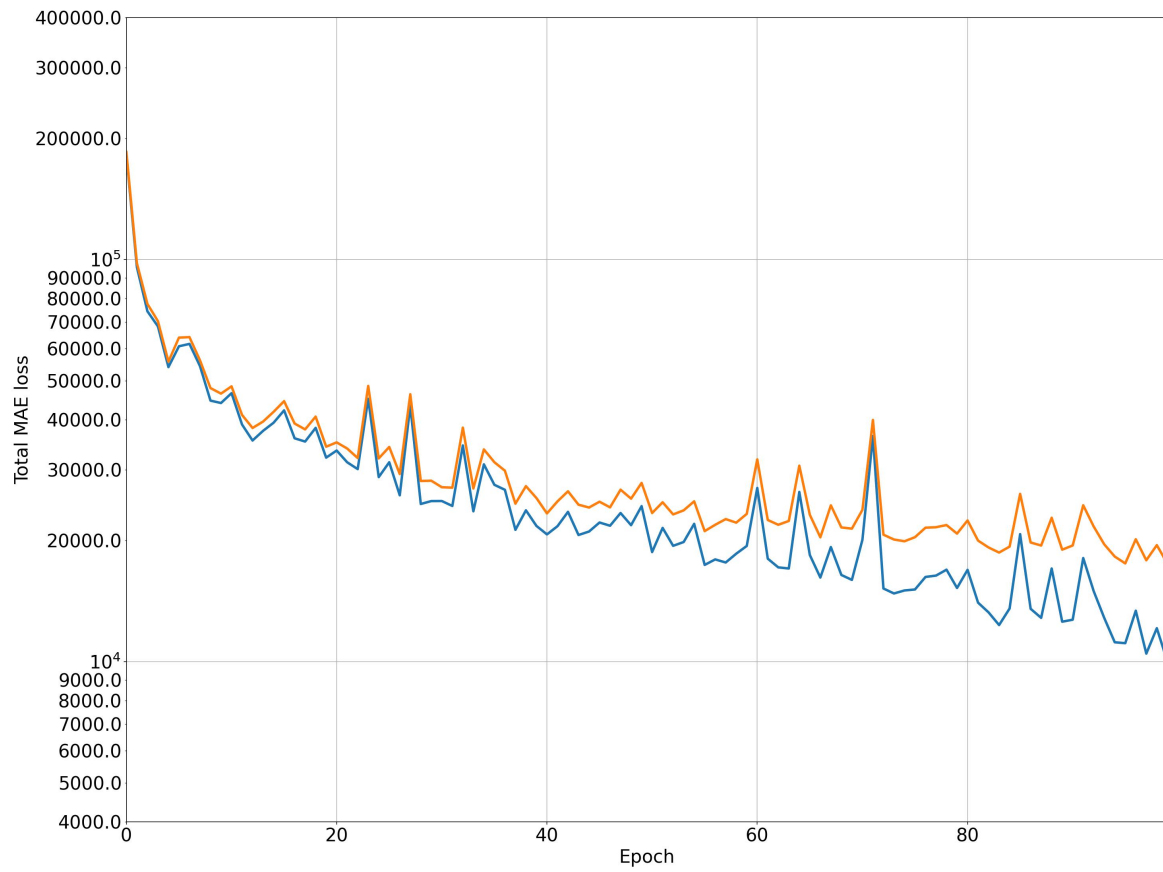


Figure C.2.: ConvNeXt total loss performance with logarithmic graph

C. Other results

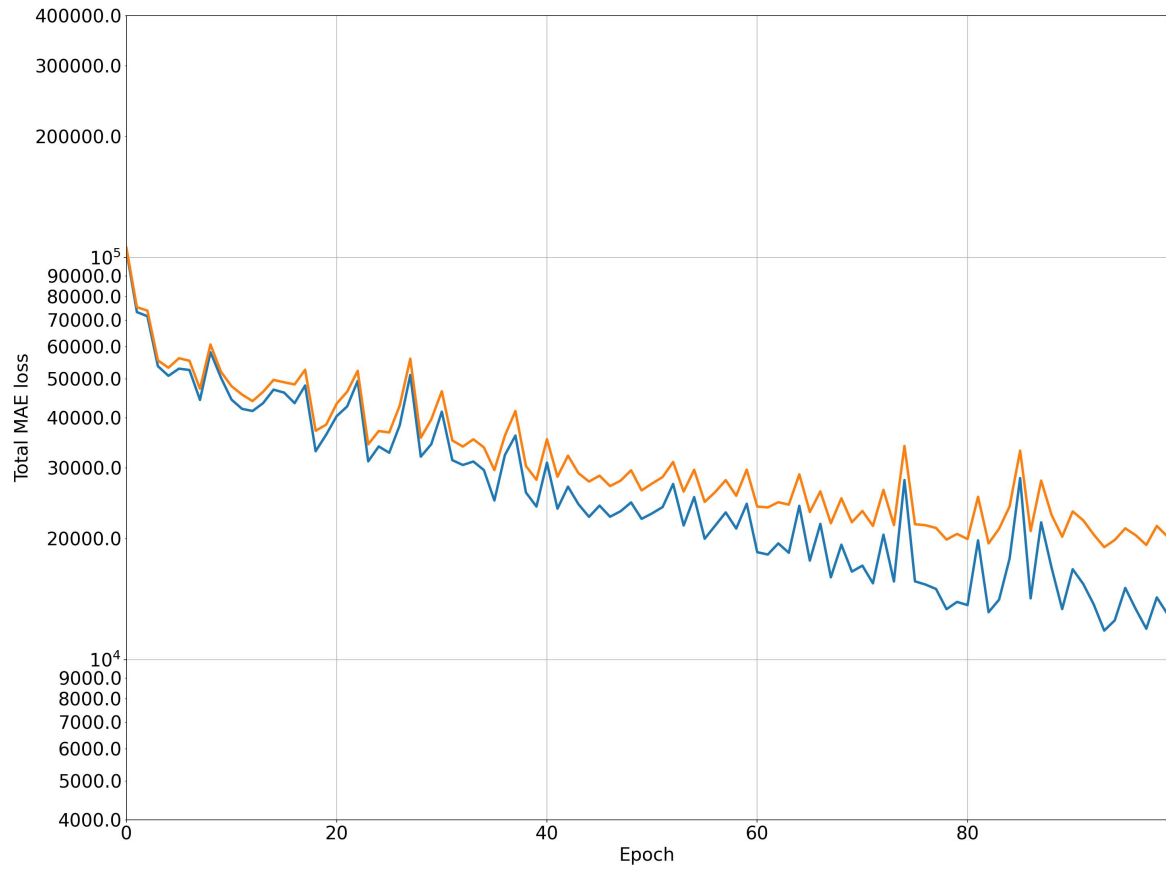


Figure C.3.: DenseNet total loss performance with logarithmic graph

C. Other results

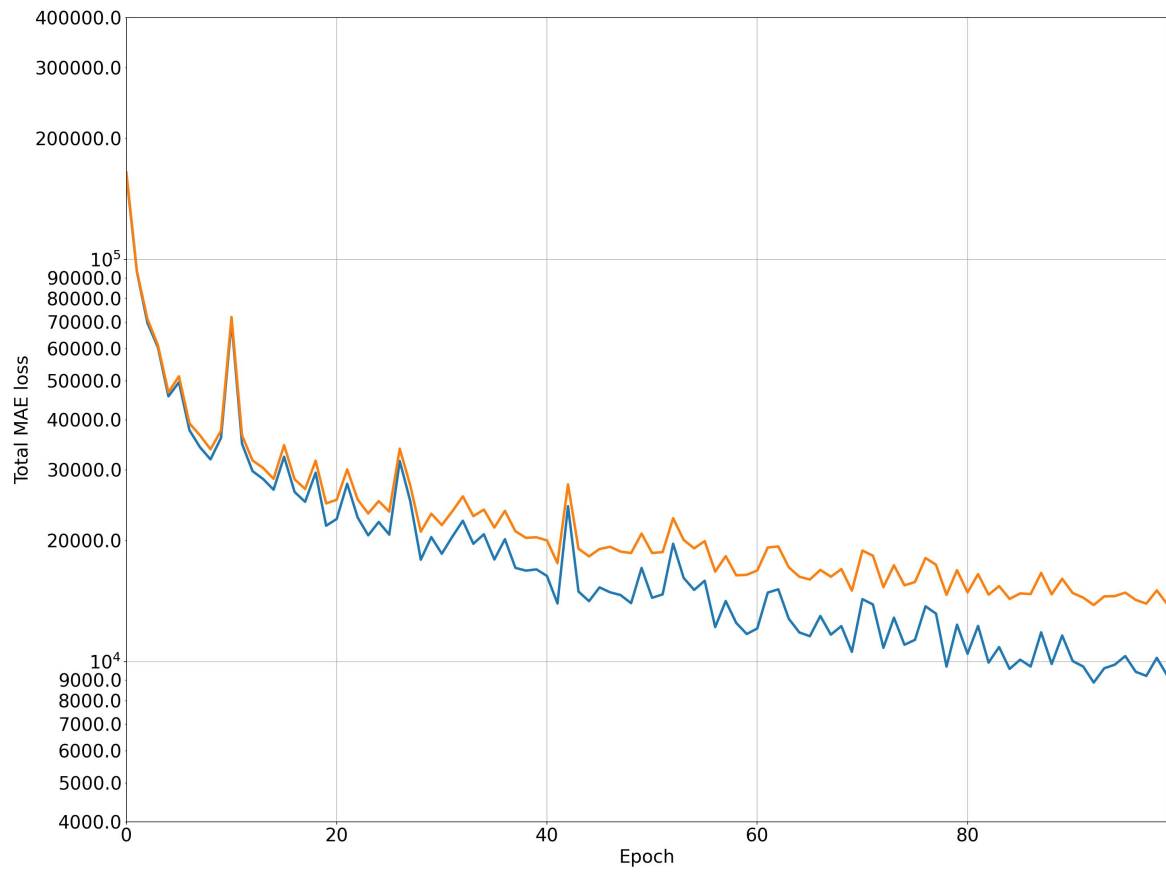


Figure C.4.: EfficientNet-B7 total loss performance with logarithmic graph

C. Other results

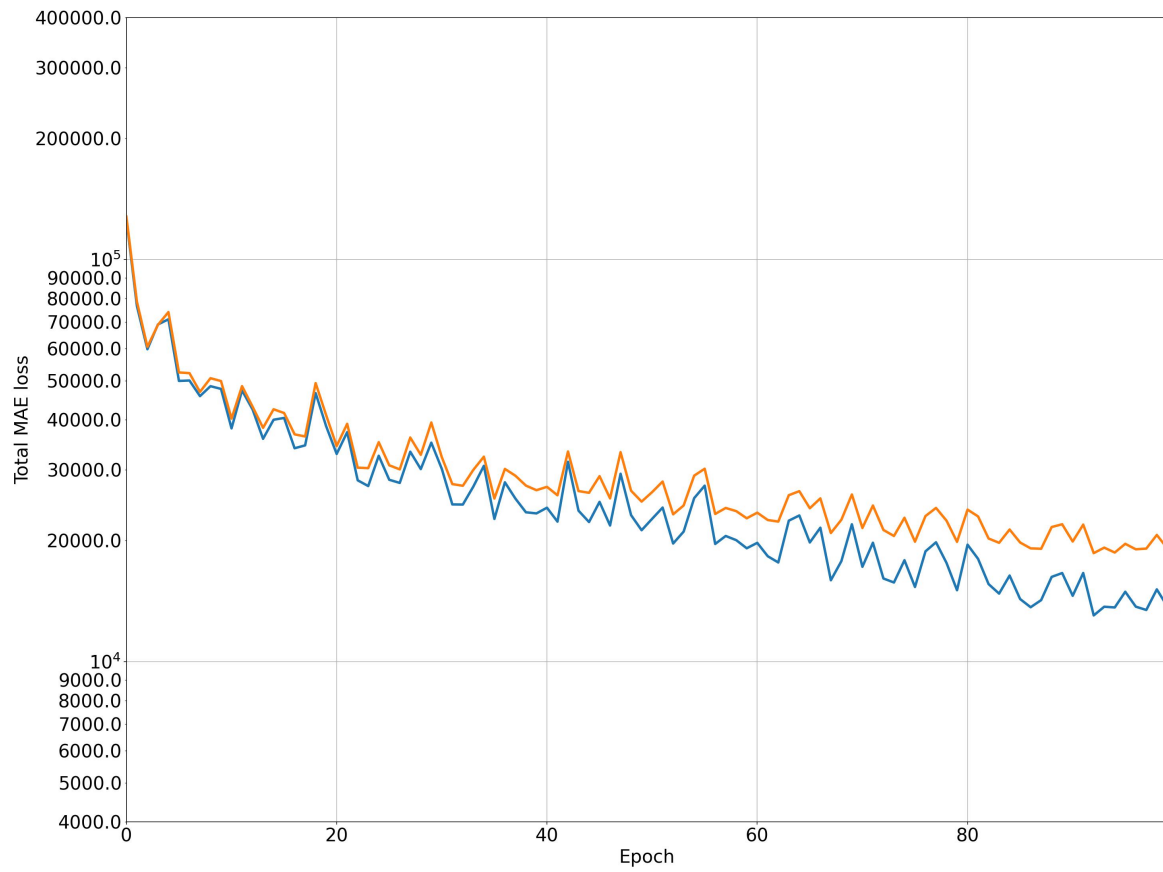


Figure C.5.: EfficientNet-v2.1 total loss performance with logarithmic graph

C. Other results

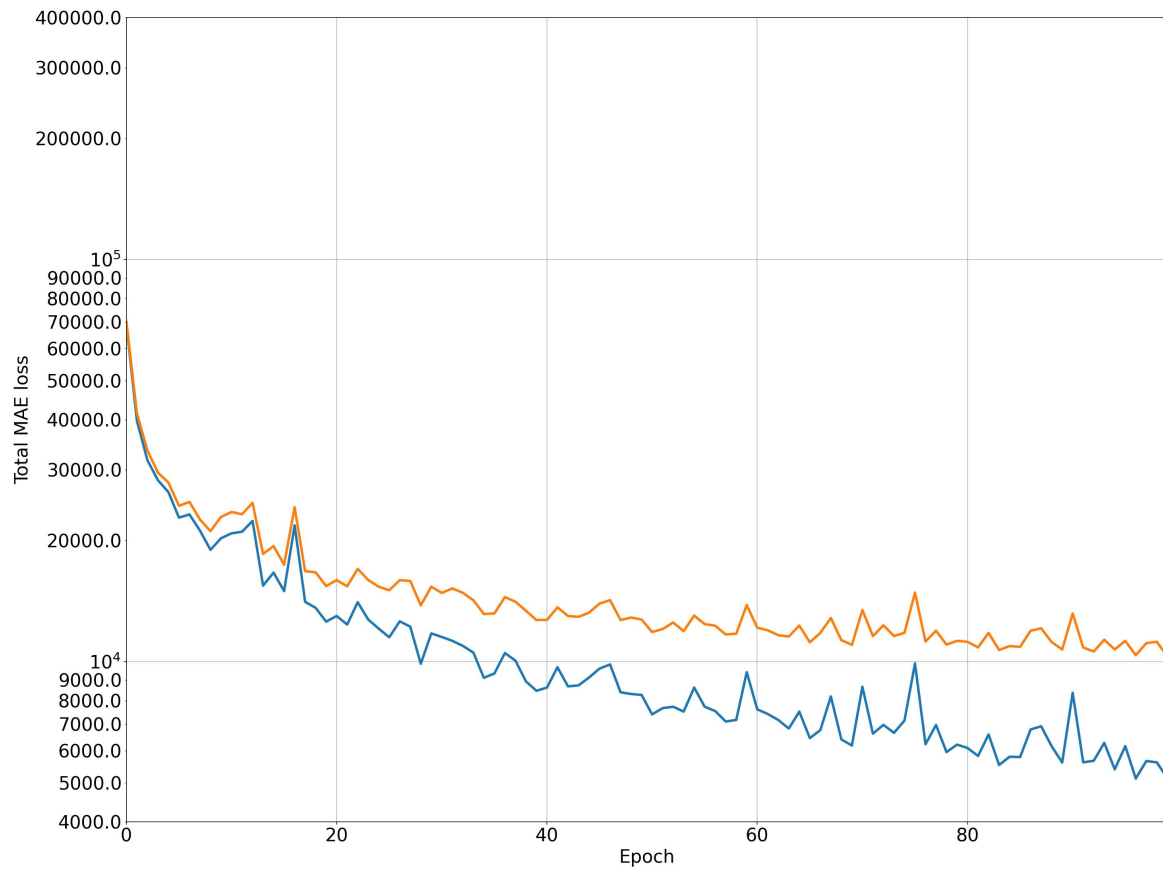


Figure C.6.: Inception-v3 total loss performance with logarithmic graph

C. Other results

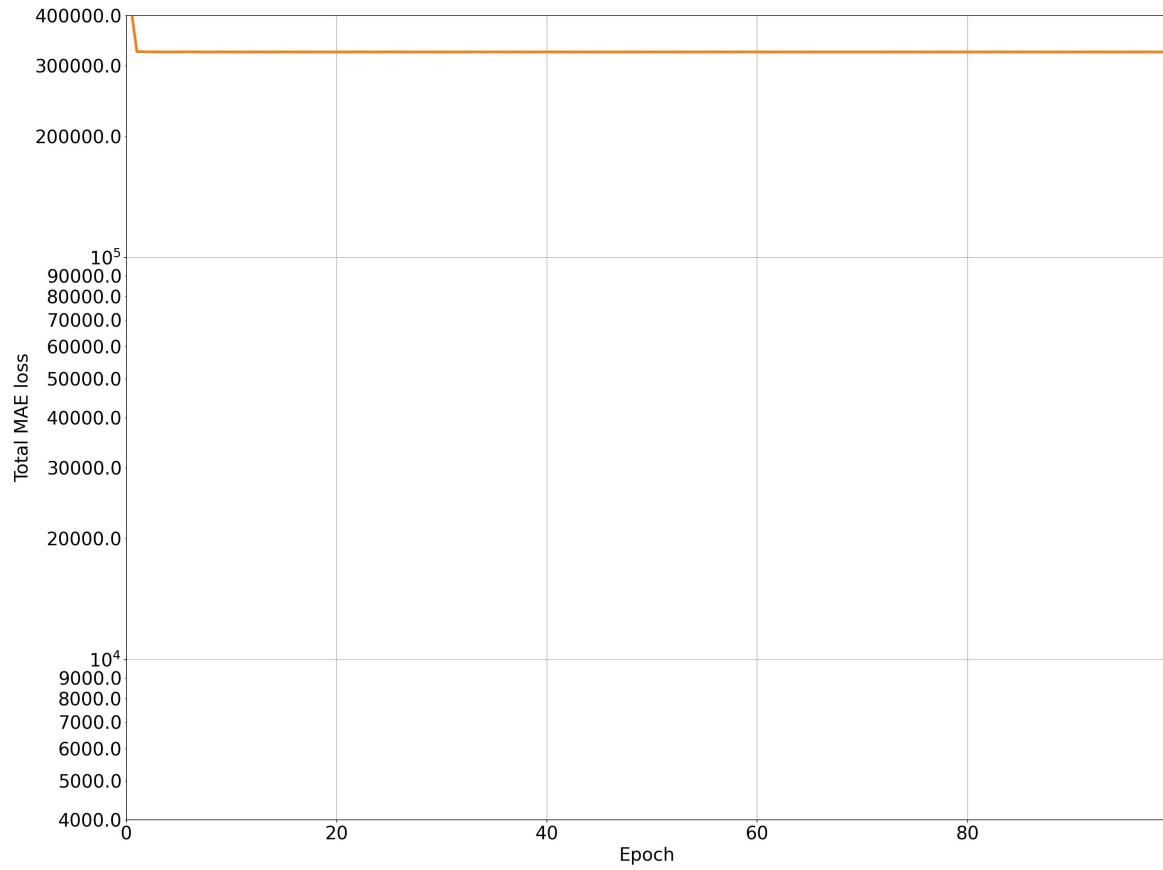


Figure C.7.: MnasNet total loss performance with logarithmic graph

C. Other results

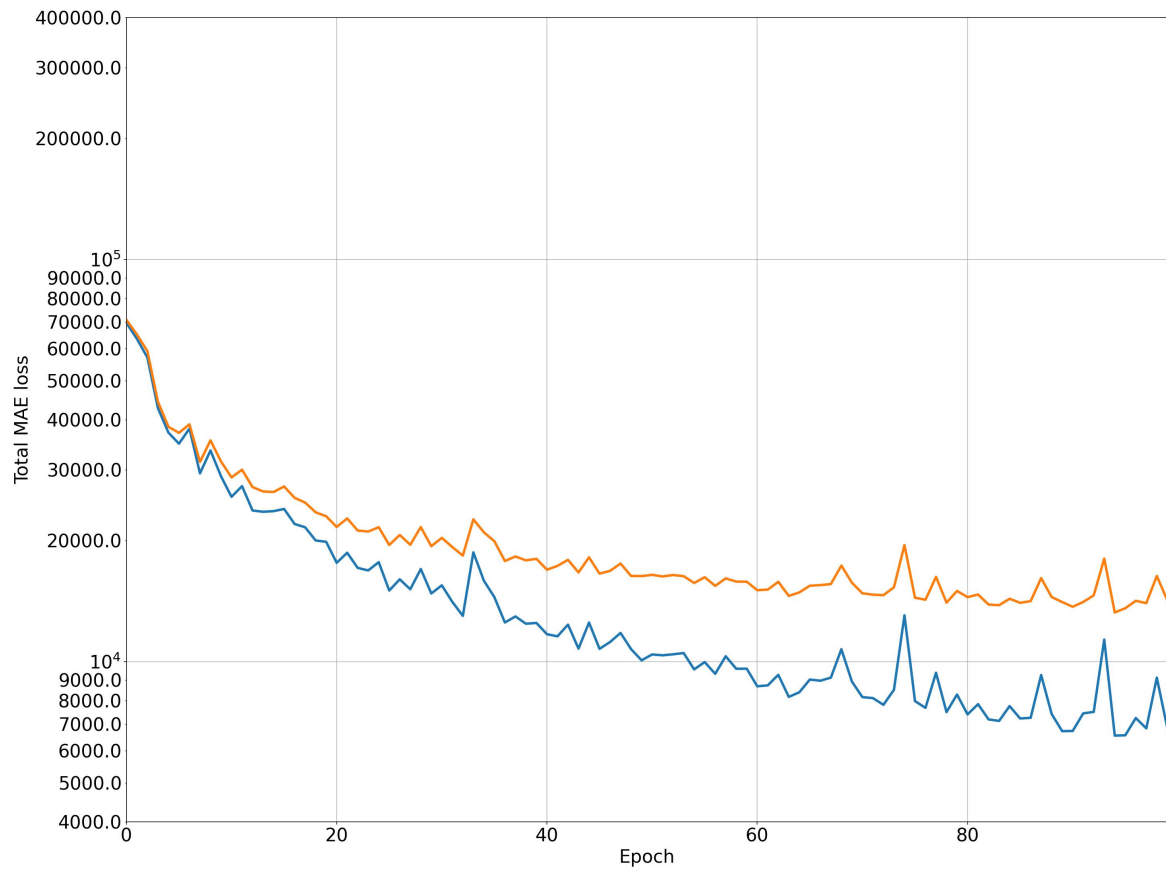


Figure C.8.: ResNet-152 total loss performance with logarithmic graph

C. Other results

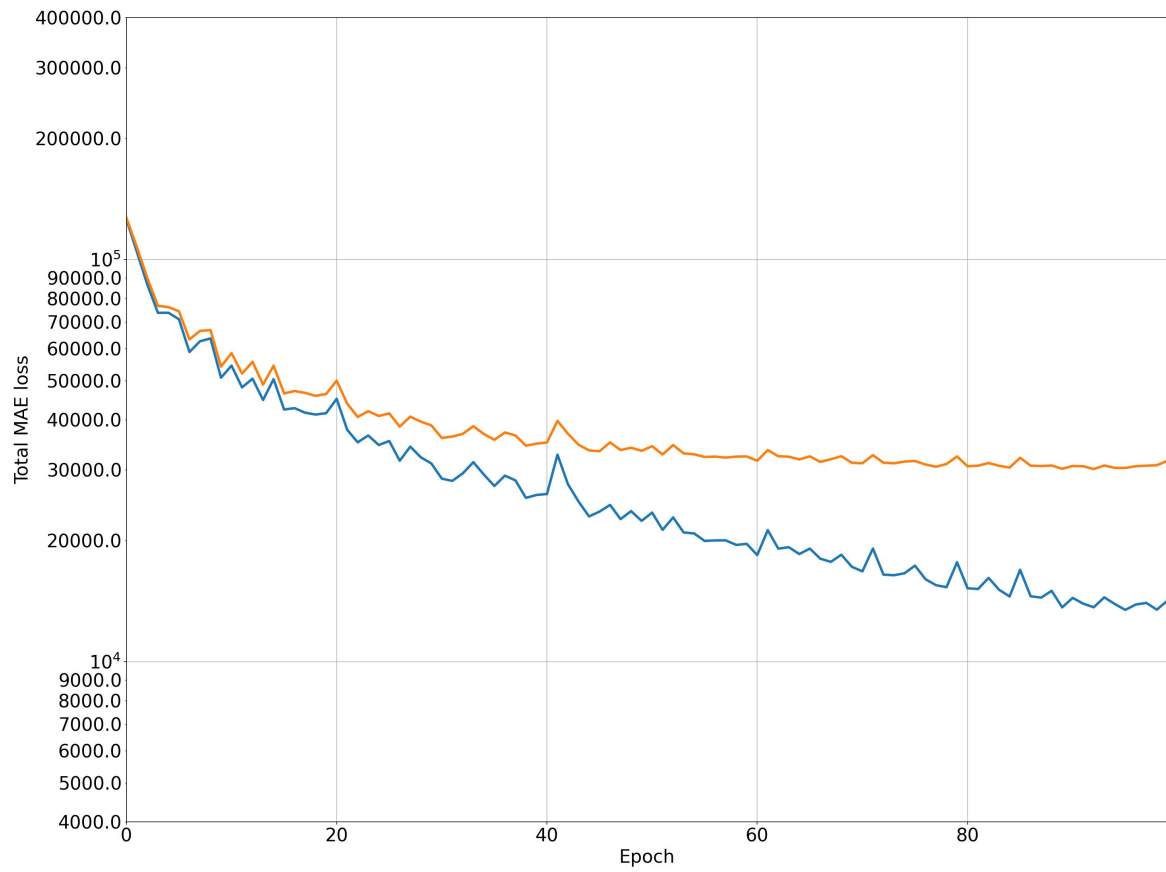


Figure C.9.: SqueezeNet-v1.1 total loss performance with logarithmic graph



C. Other results

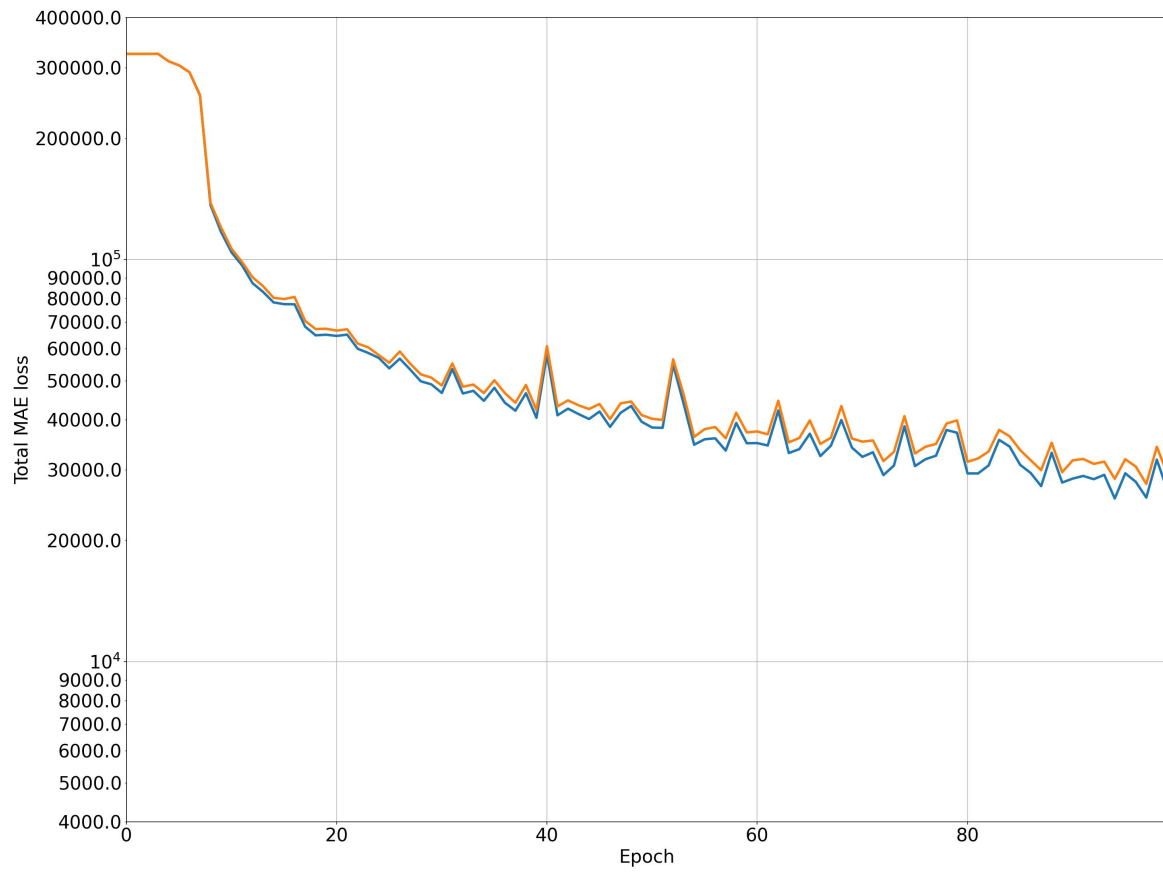


Figure C.10.: Swin-v2-b total loss performance with logarithmic graph

C. Other results

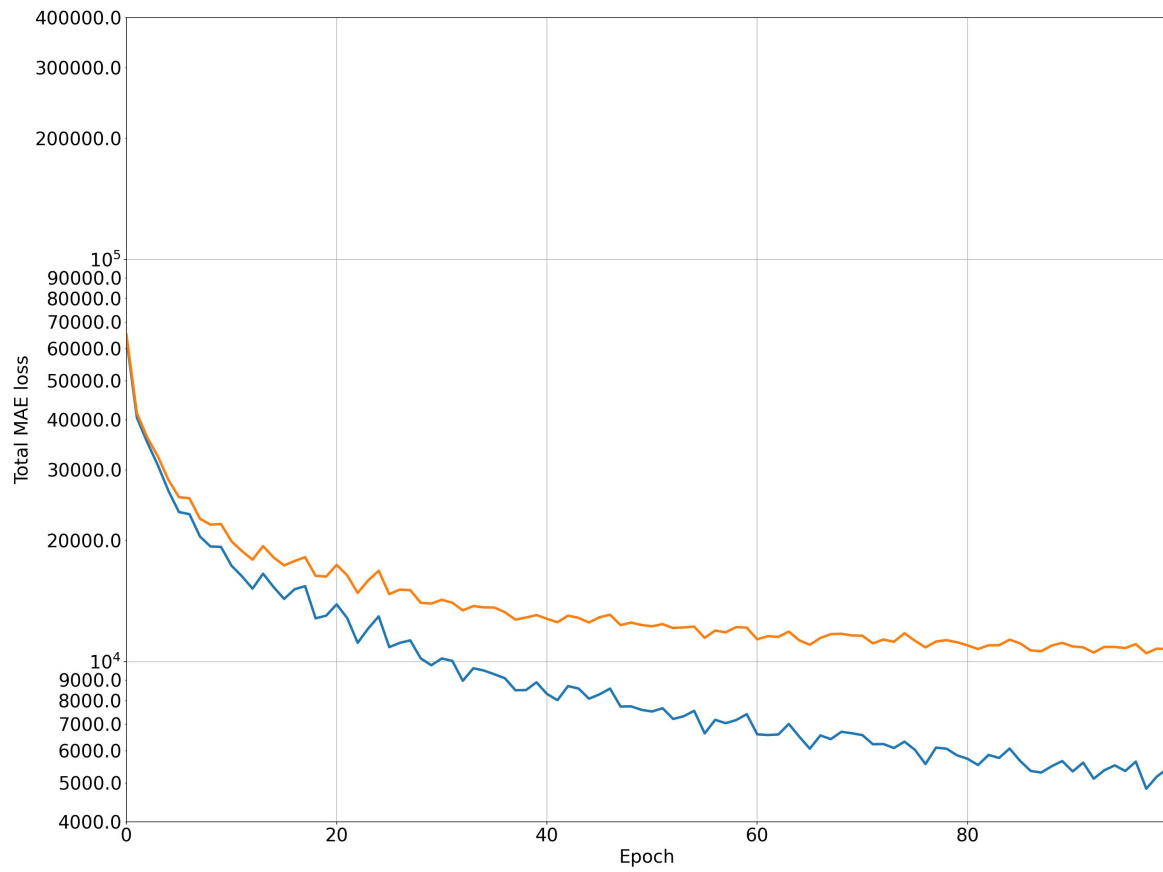


Figure C.11.: VGG-19\_bn total loss performance with logarithmic graph

C. Other results

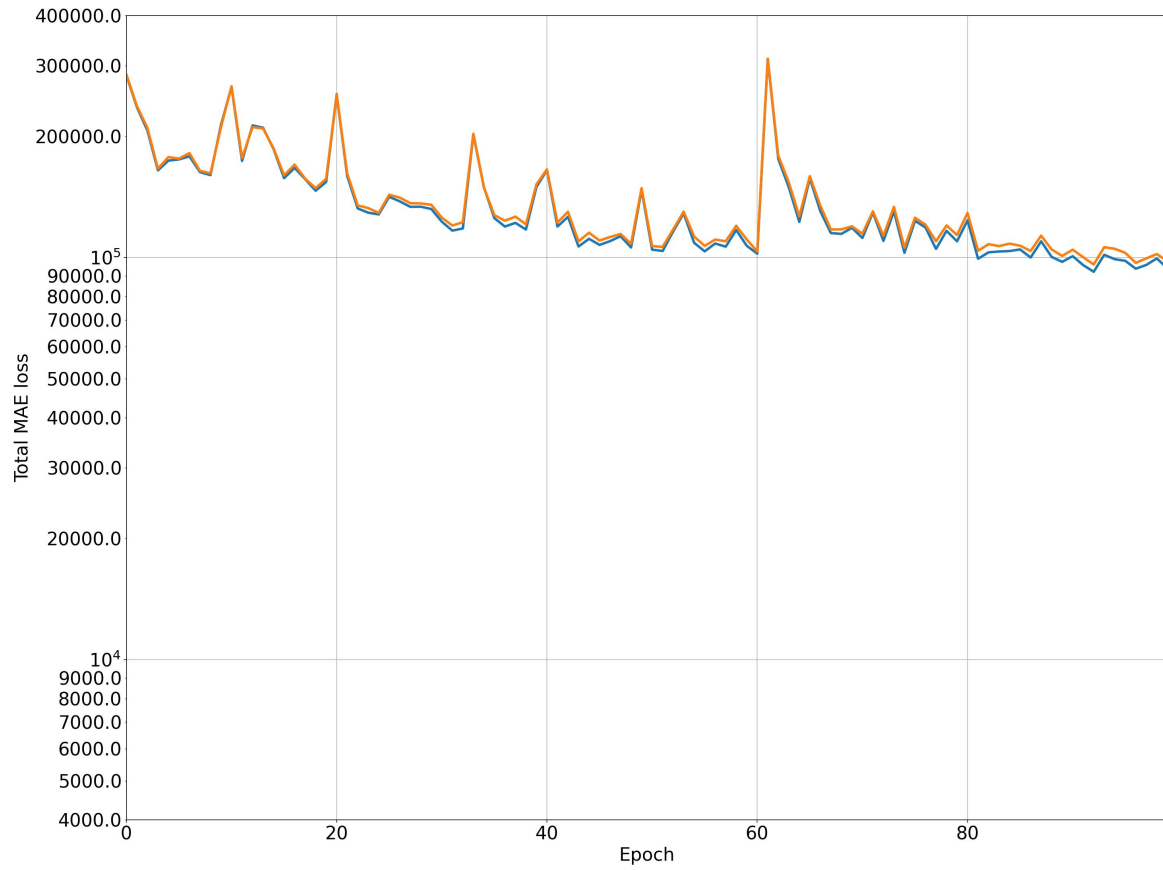


Figure C.12.: ViT-16-b total loss performance with logarithmic graph

C. Other results

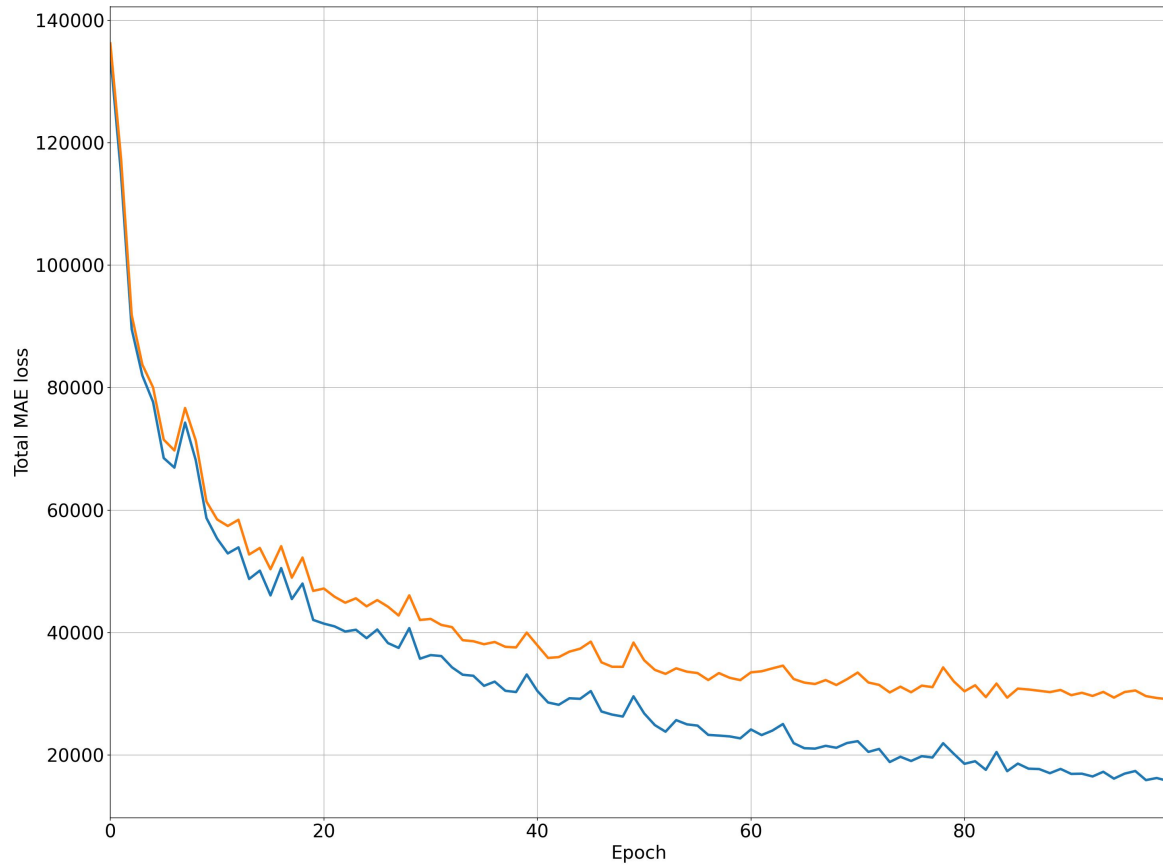


Figure C.13.: AlexNet total loss performance with linear graph

C. Other results

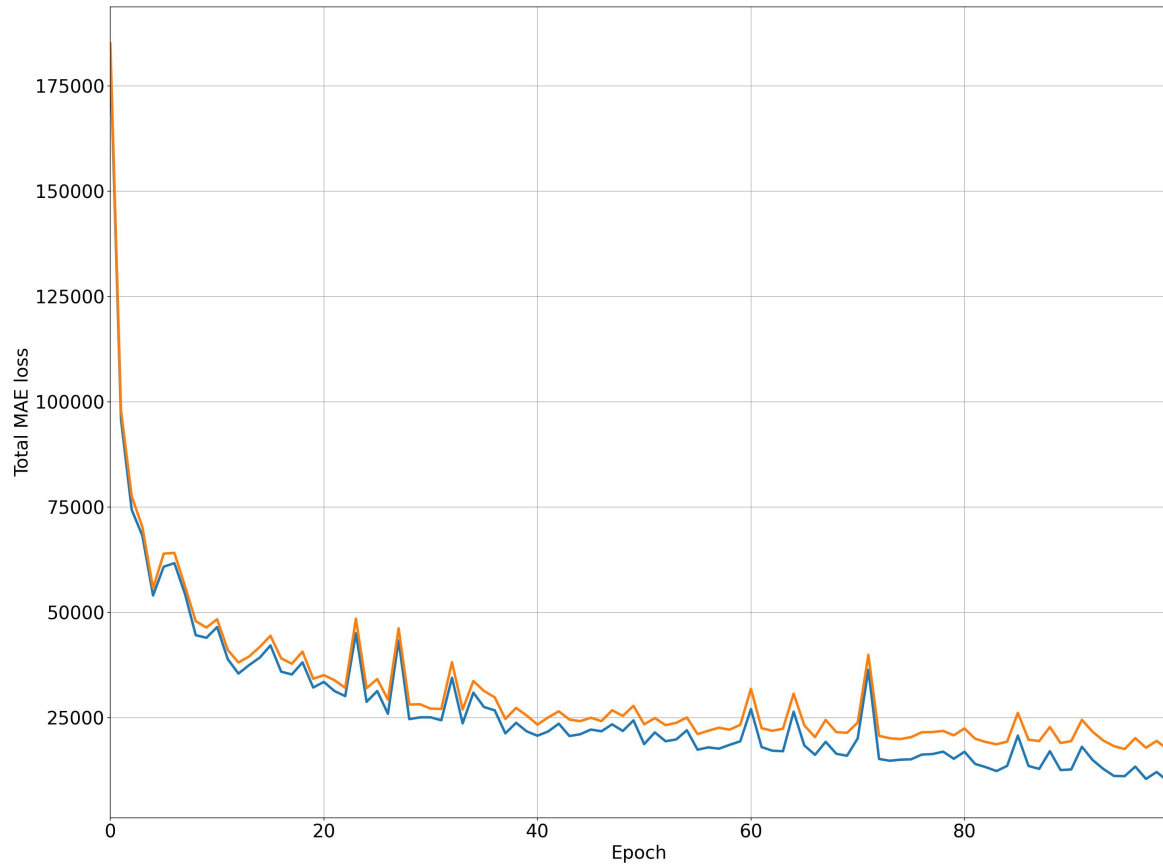


Figure C.14.: ConvNeXt total loss performance with linear graph

C. Other results

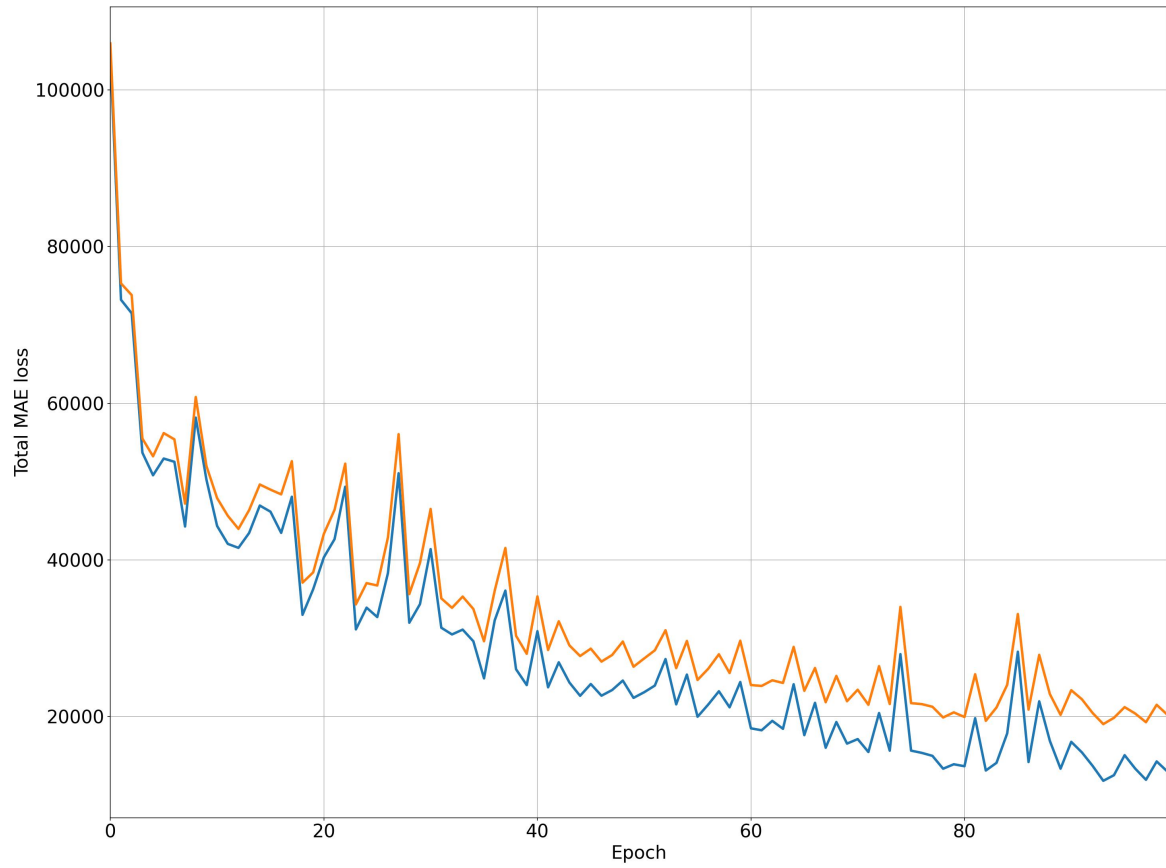


Figure C.15.: DenseNet-201 total loss performance with linear graph

C. Other results

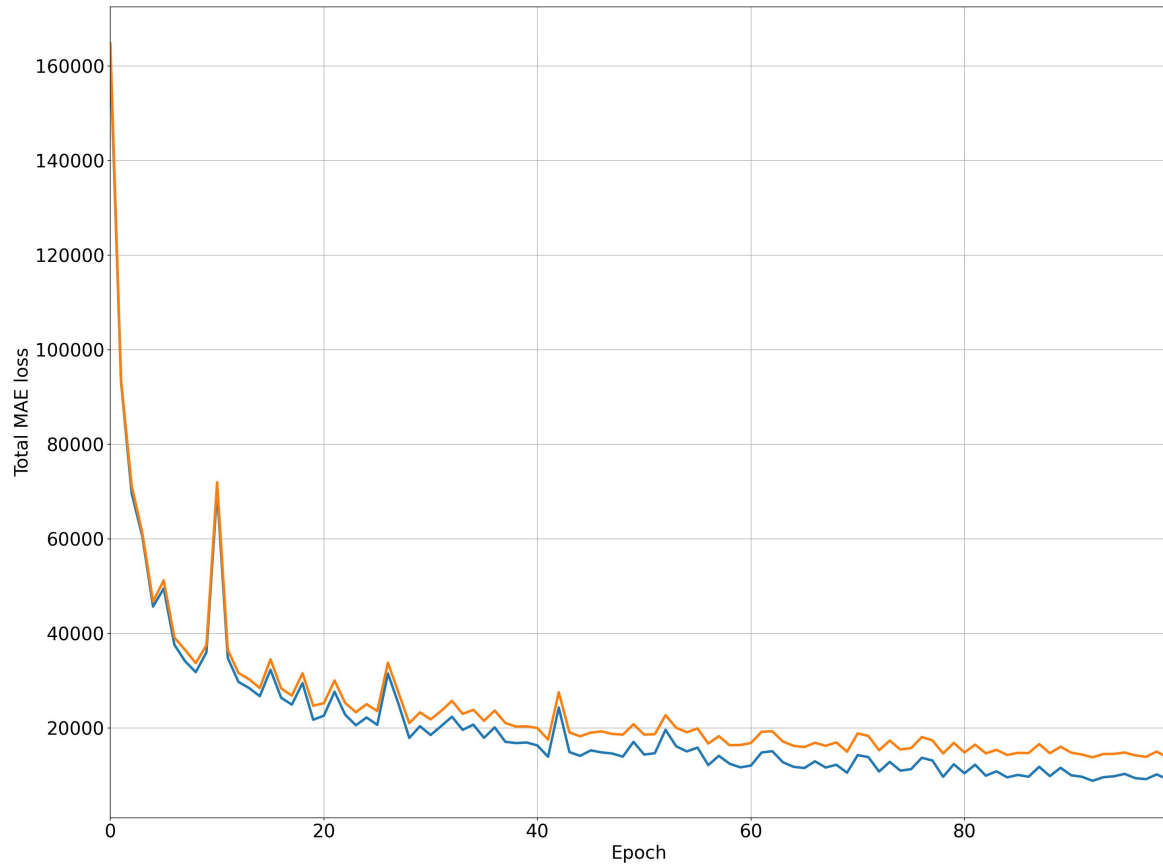


Figure C.16.: EfficientNet-B7 total loss performance with linear graph

C. Other results

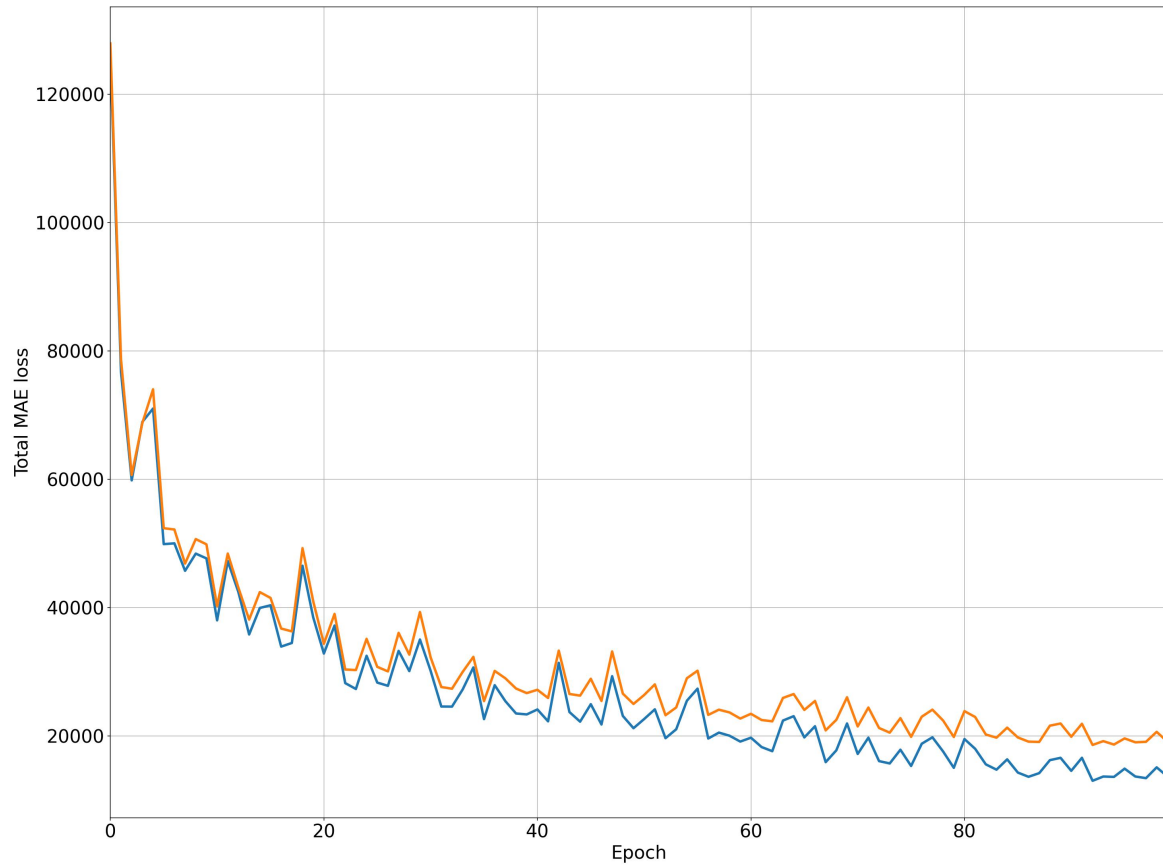


Figure C.17.: EfficientNet-v2\_1 total loss performance with linear graph



C. Other results

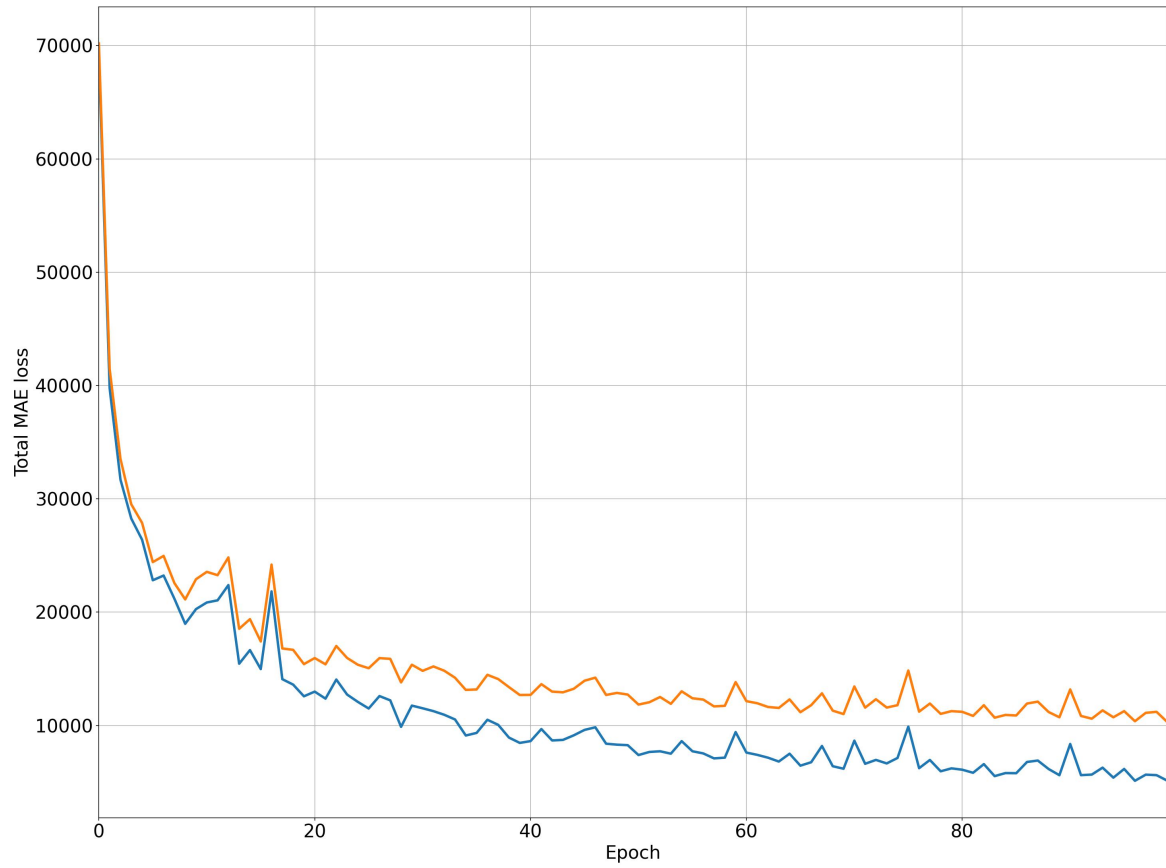


Figure C.18.: Inception-V3 total loss performance with linear graph

C. Other results

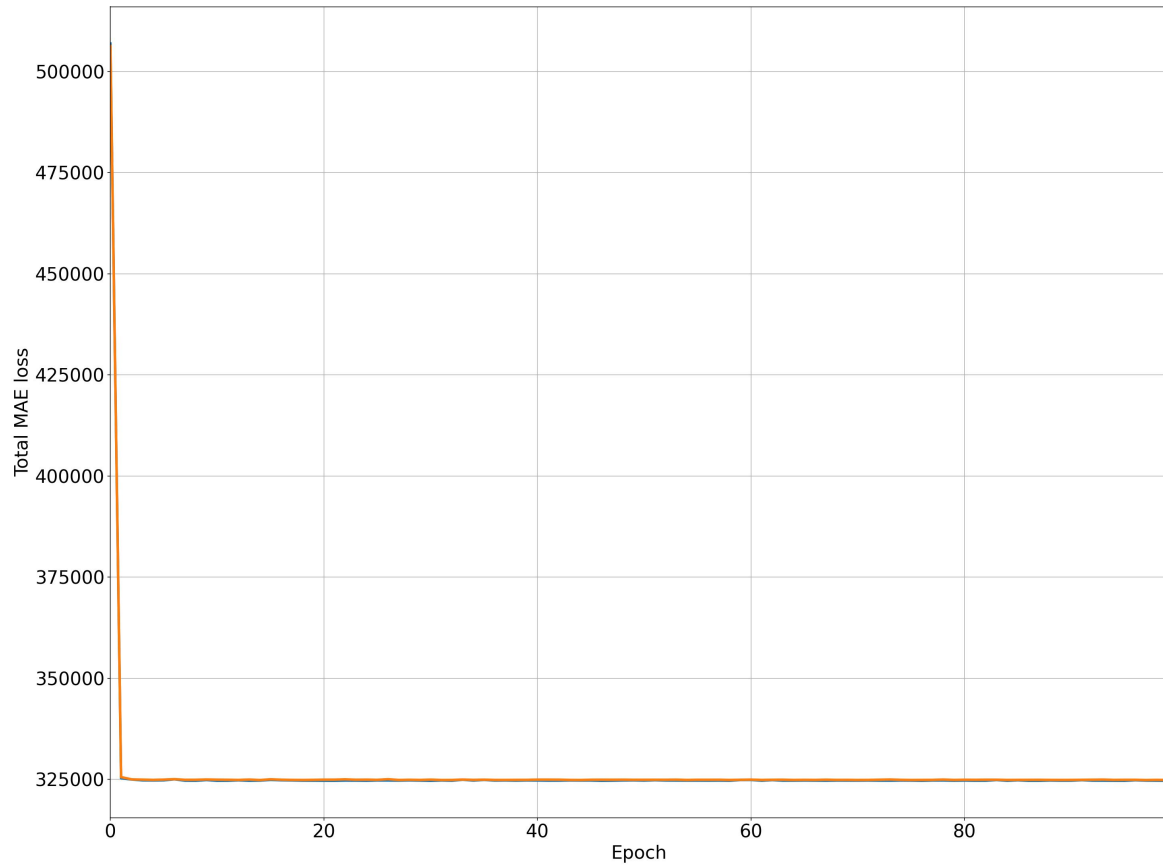


Figure C.19.: MnasNet-6.0 total loss performance with linear graph

C. Other results

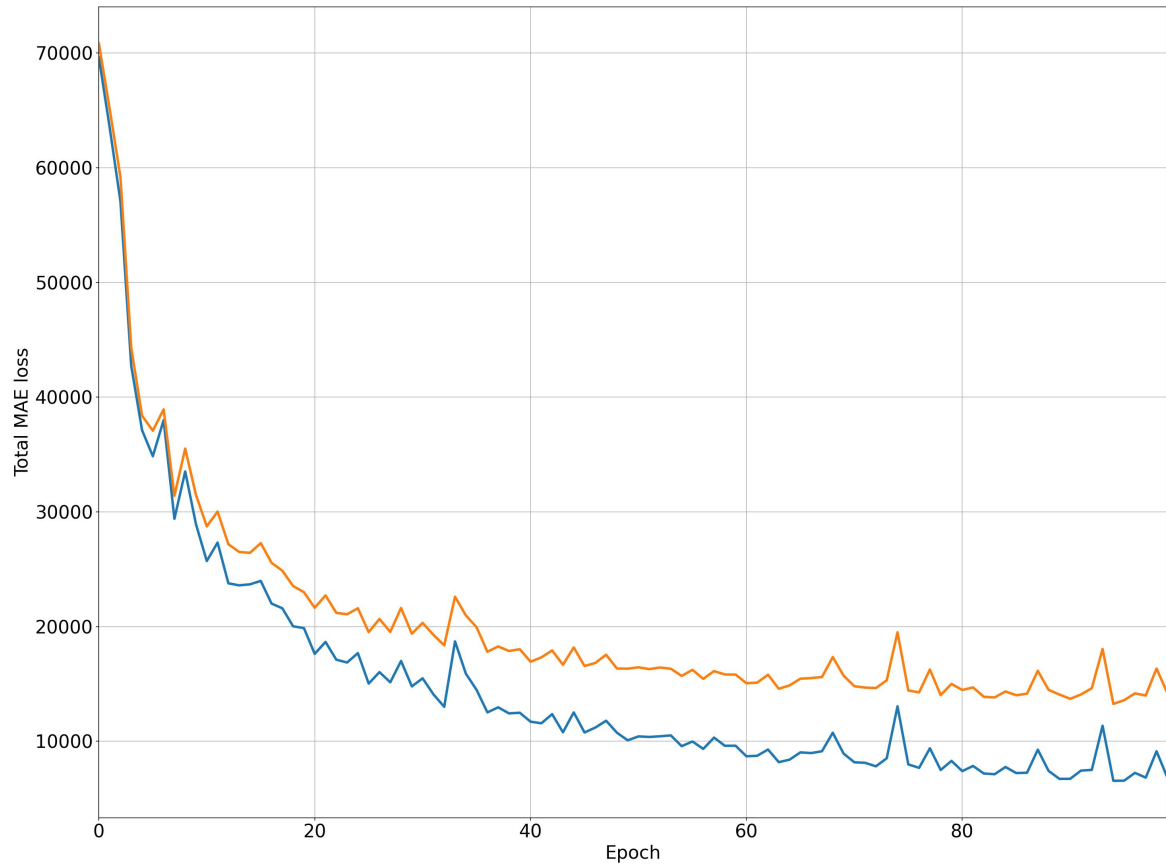


Figure C.20.: ResNet-152 total loss performance with linear graph

C. Other results

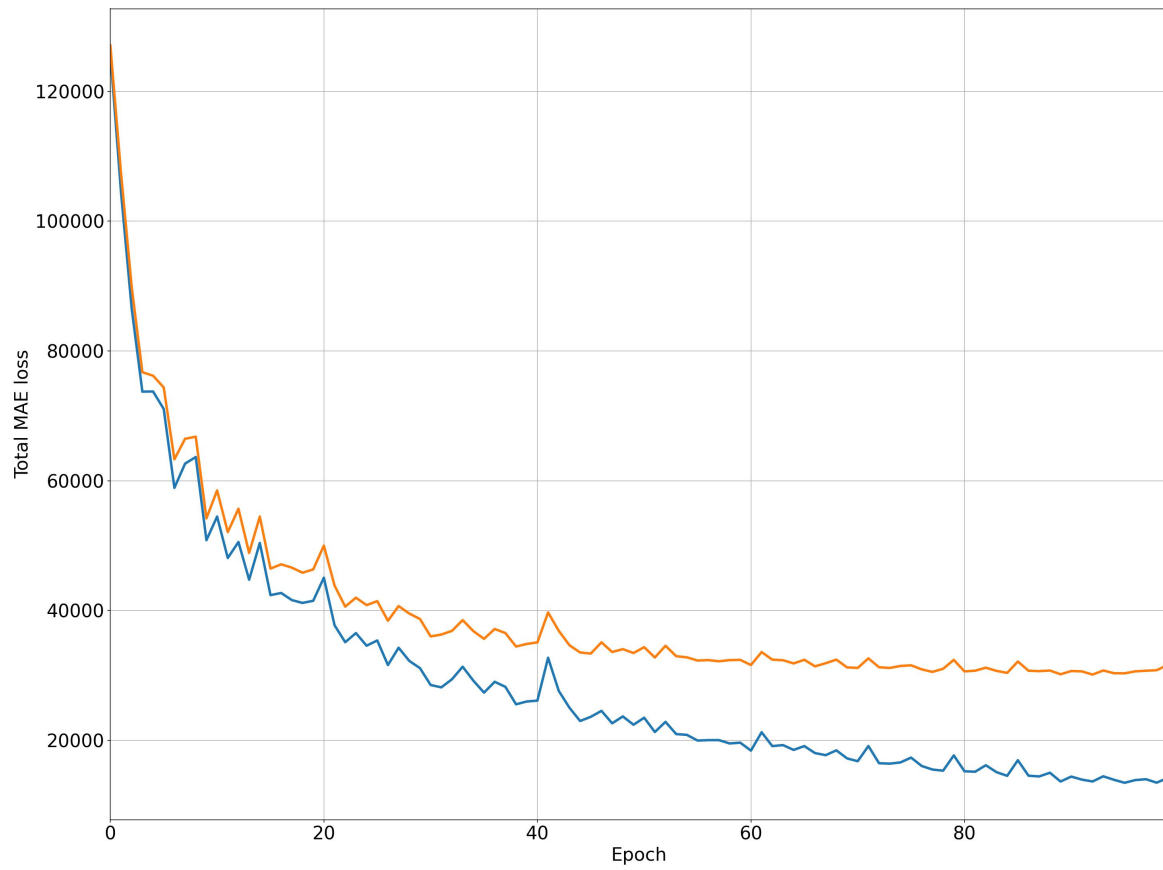


Figure C.21.: SqueezeNet-V1.1 total loss performance with linear graph

C. Other results

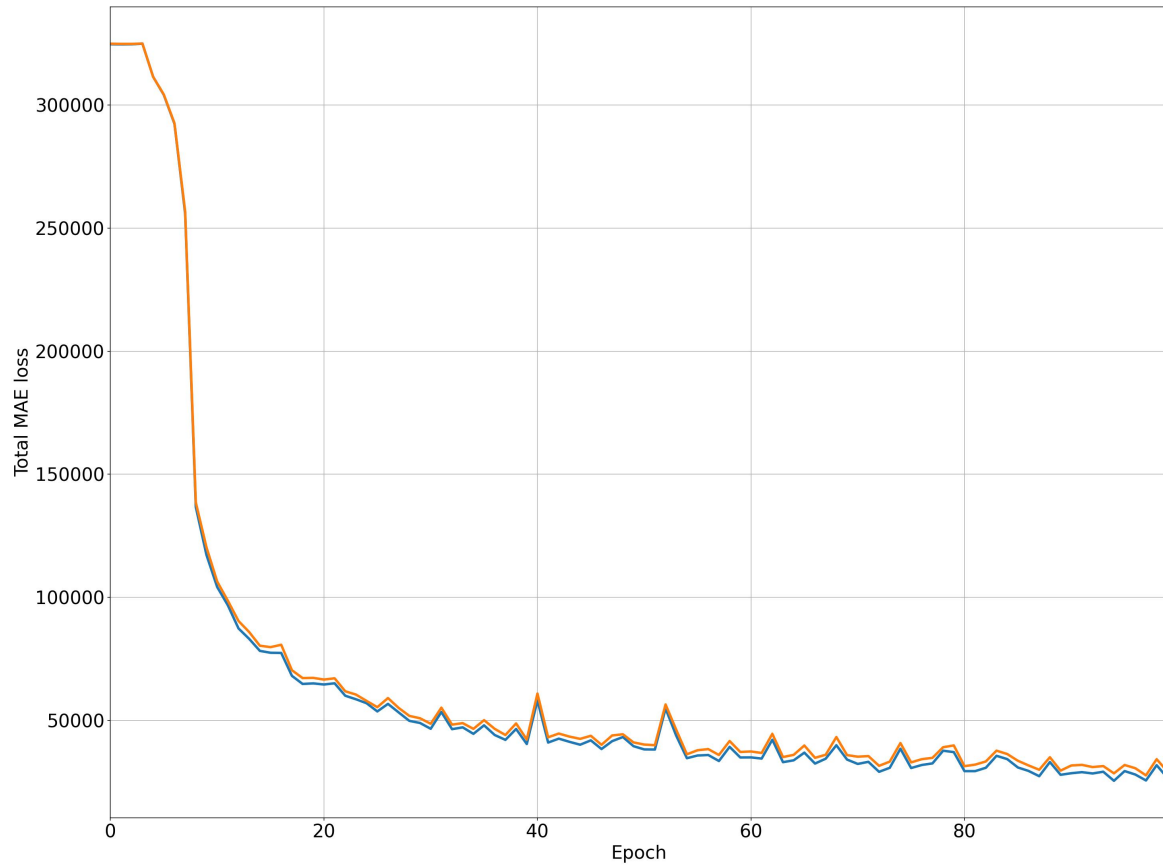


Figure C.22.: Swin-v2-b total loss performance with linear graph

C. Other results

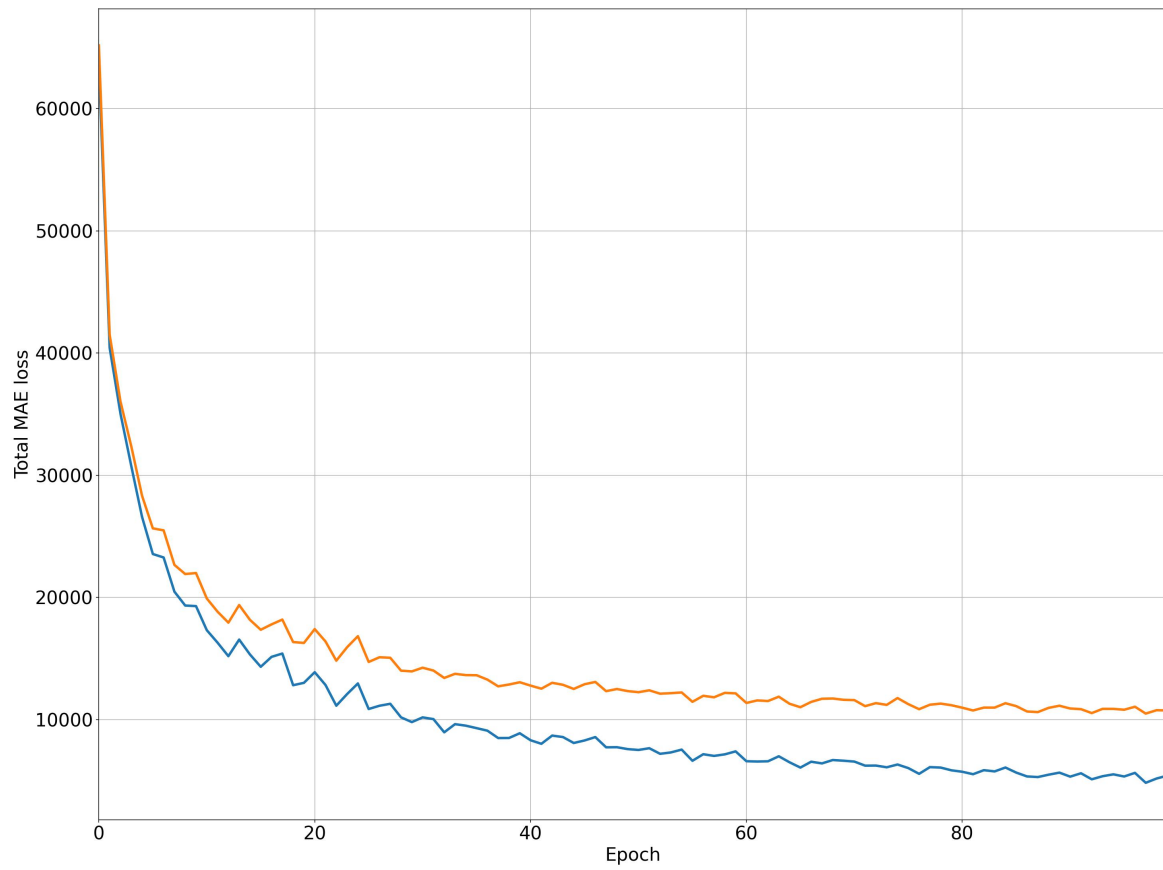


Figure C.23.: VGG-19\_bn total loss performance with linear graph

C. Other results

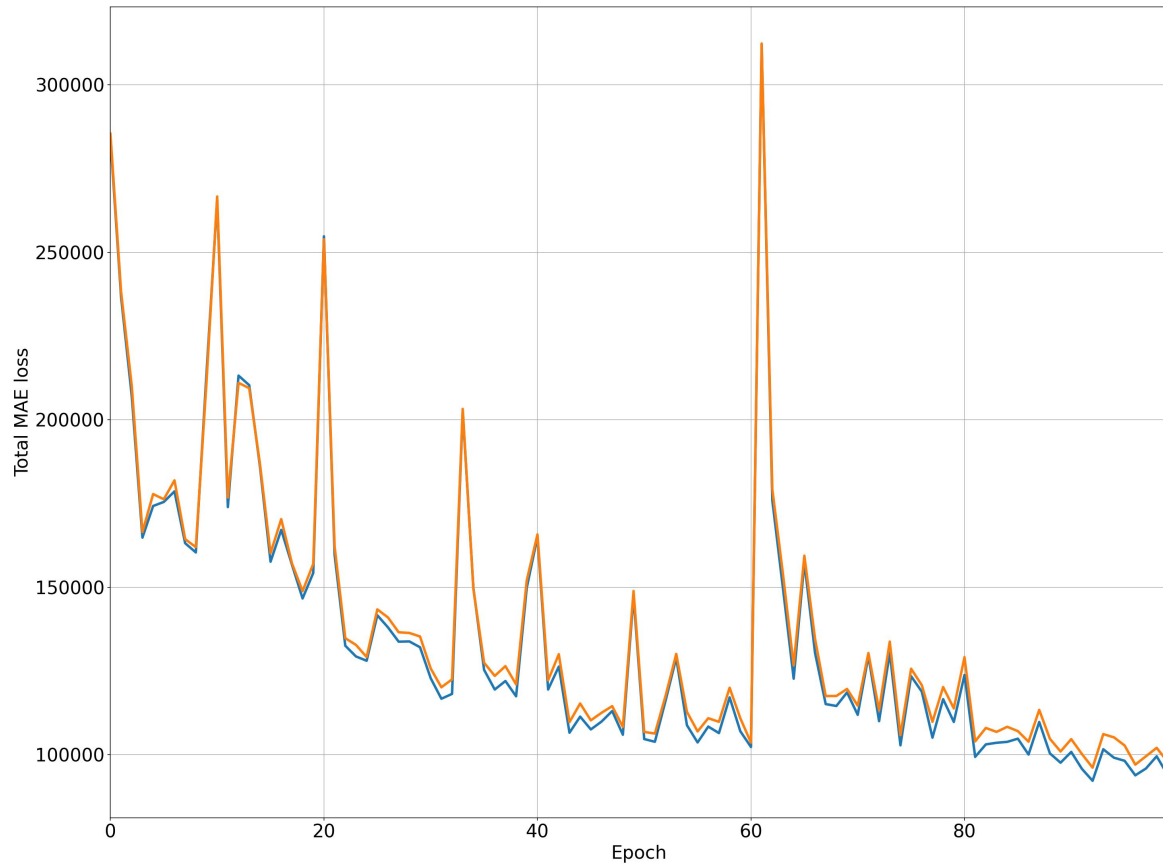


Figure C.24.: ViT-16-b total loss performance with linear graph

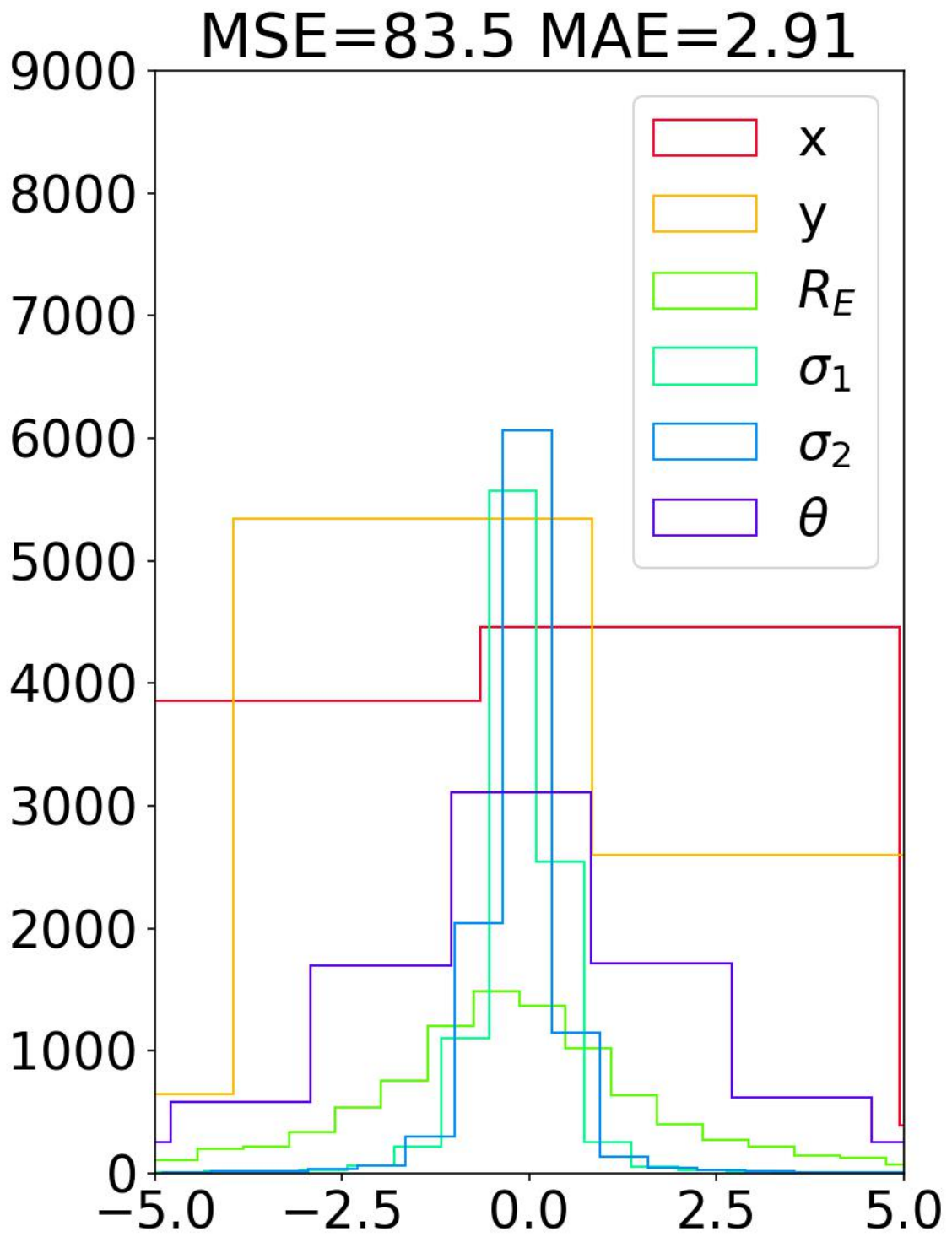


Figure C.25.: AlexNet histogram pre-trained with 0.0001 learning rate



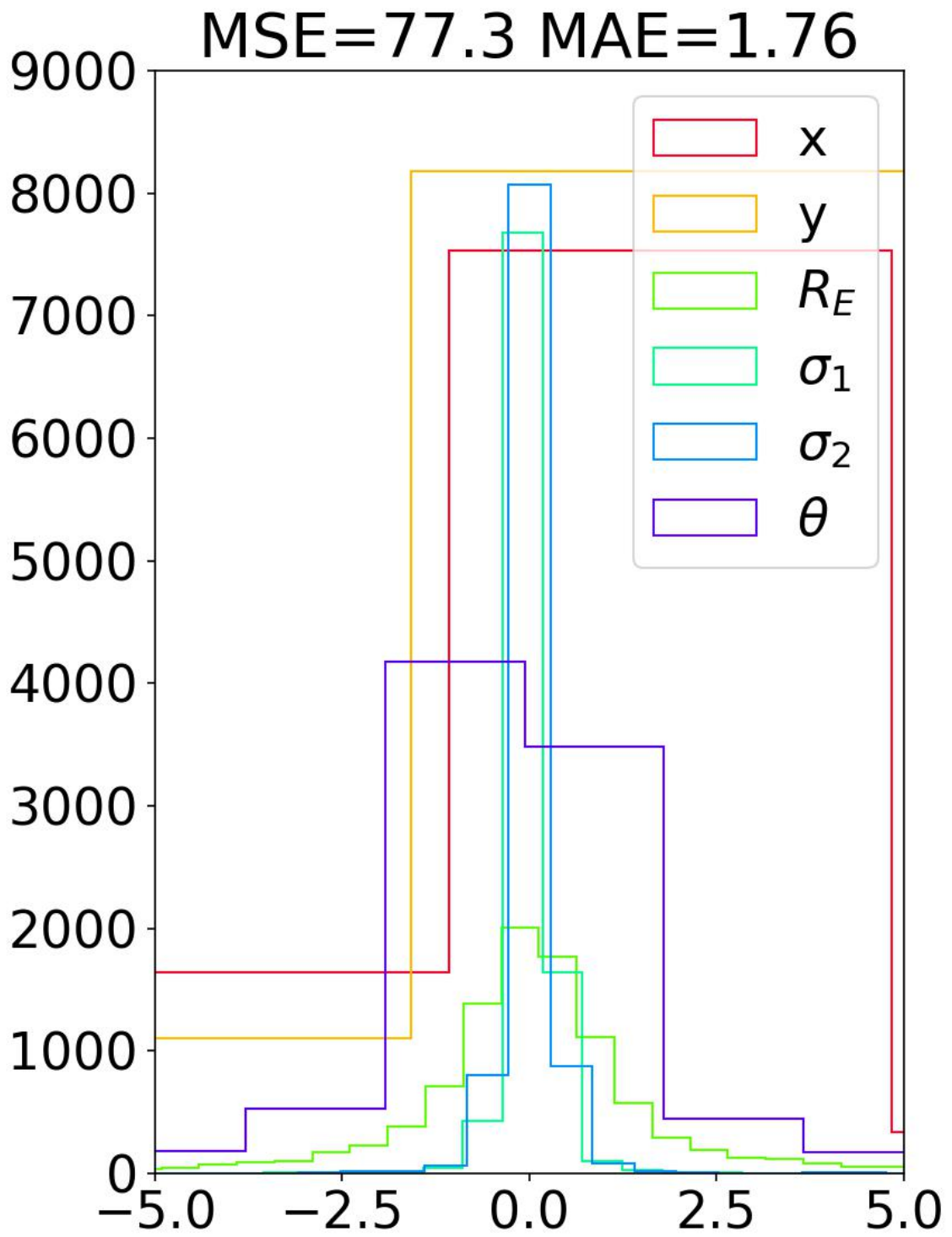


Figure C.26.: ConvNeXt histogram with 0.0001 learning rate

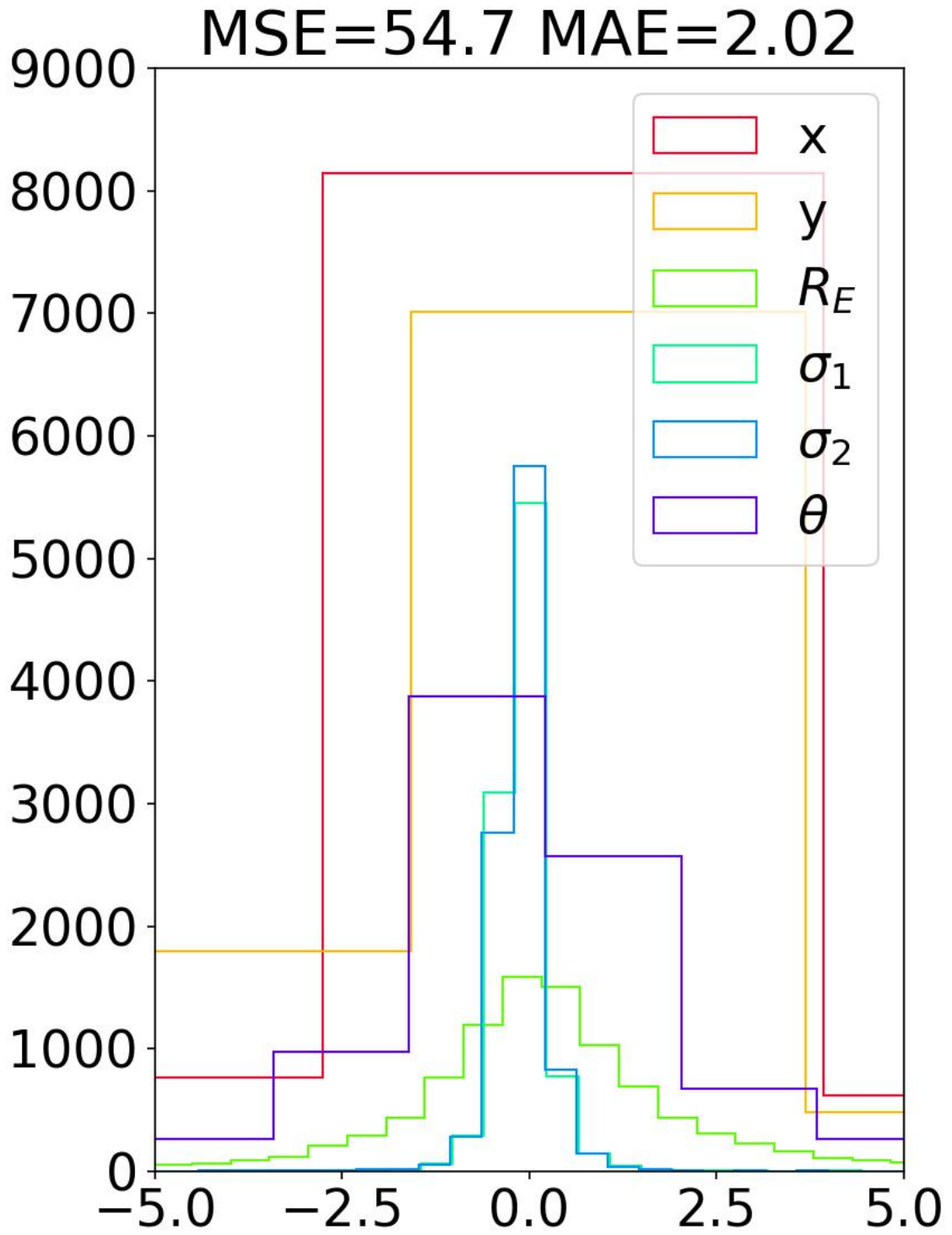


Figure C.27.: DenseNet histogram with 0.0001 learning rate

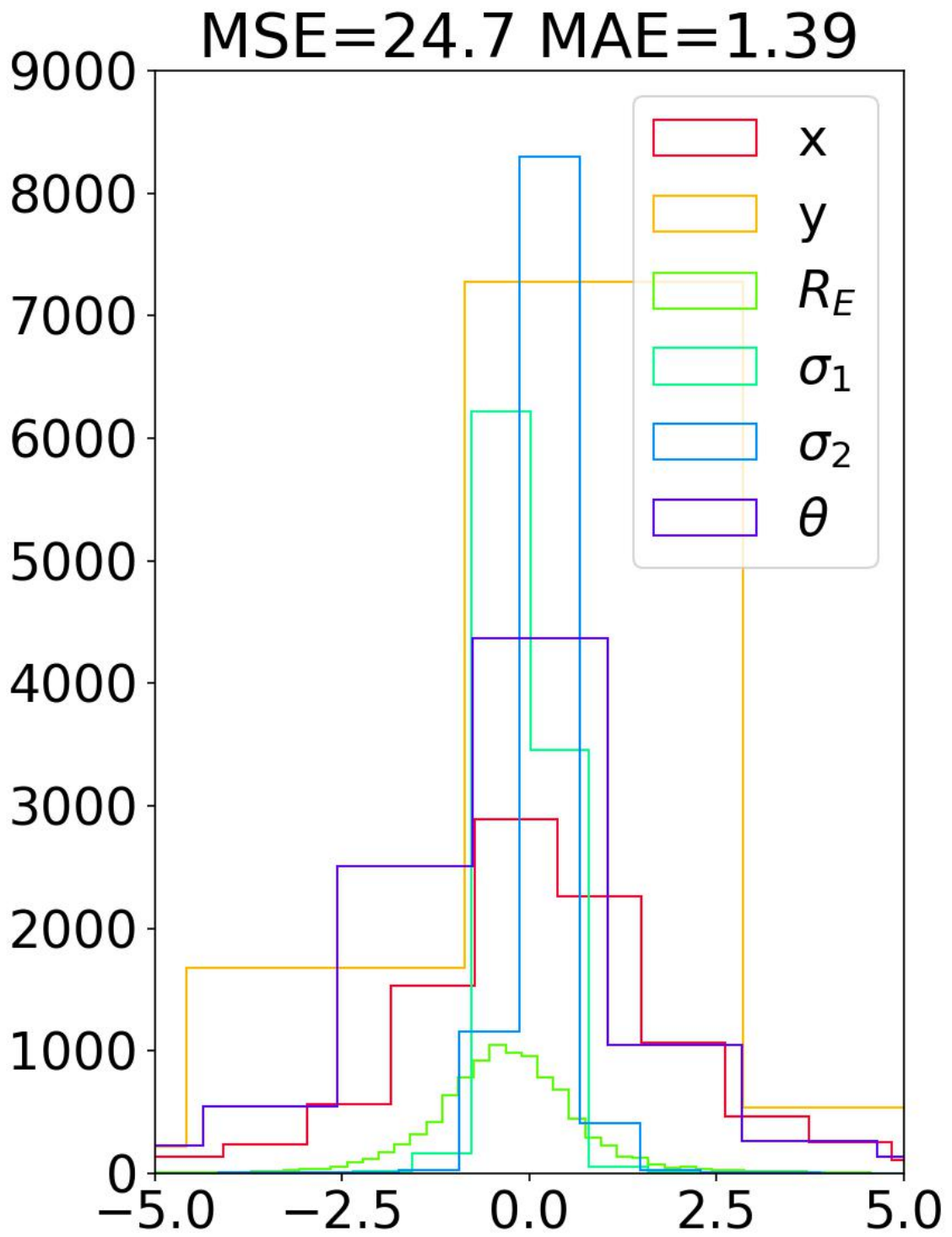


Figure C.28.: EfficientNet histogram with 0.0001 learning rate

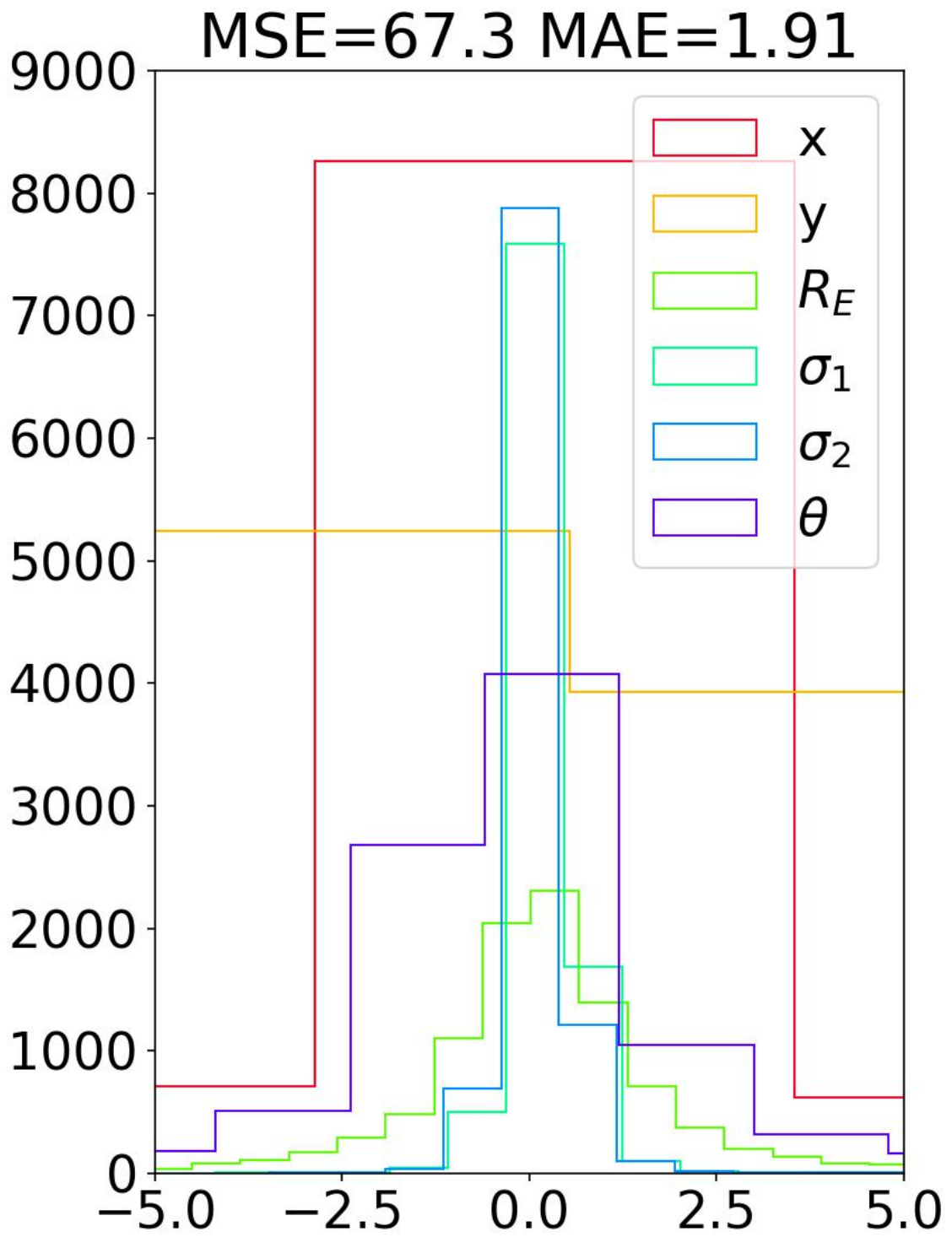


Figure C.29.: EfficientNet-v2.1 histogram with 0.0001 learning rate

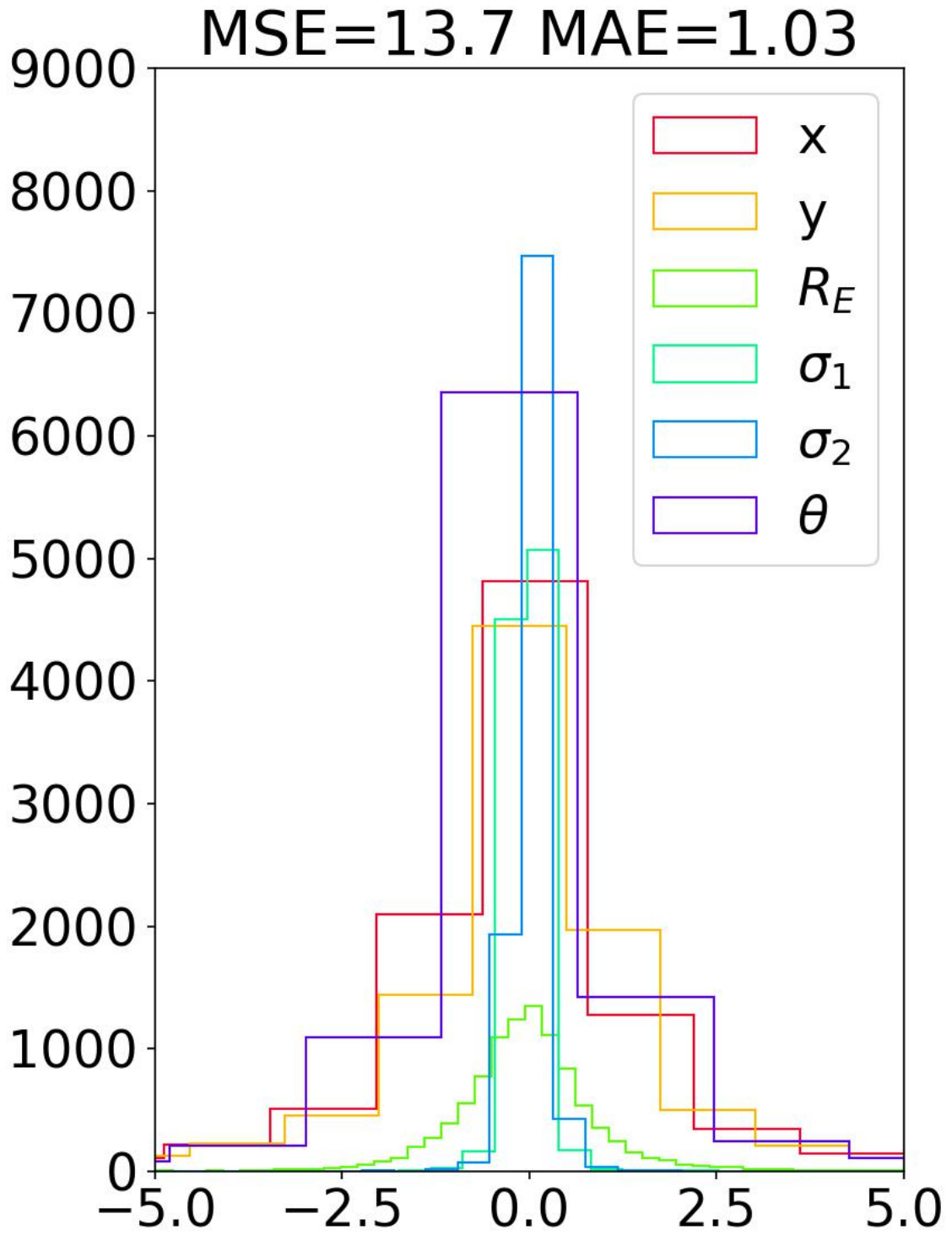


Figure C.30.: Inception-v3 histogram pre-trained with 0.0001 learning rate

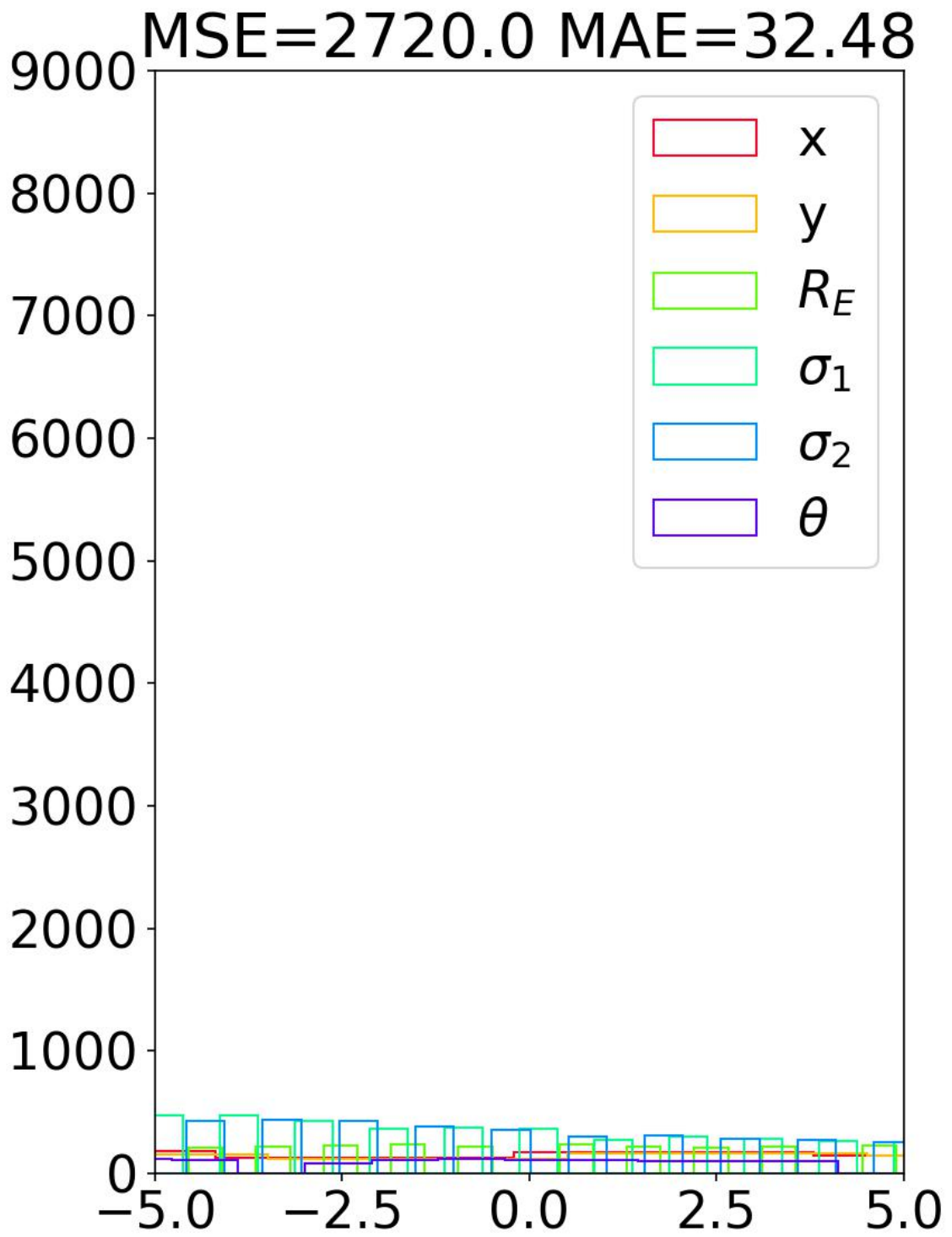


Figure C.31.: MnasNet histogram with 0.0001 learning rate

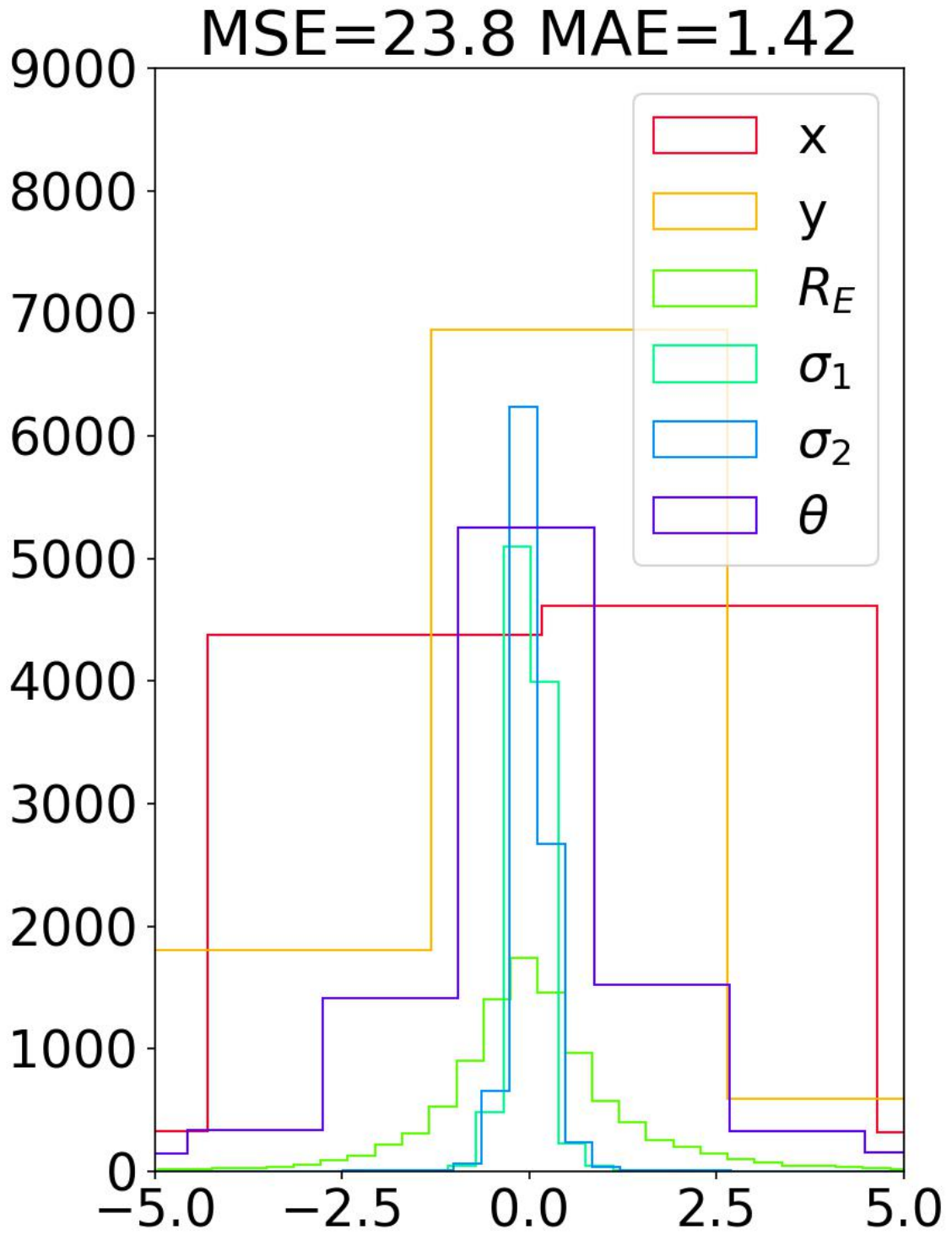


Figure C.32.: ResNet histogram with 0.0001 learning rate

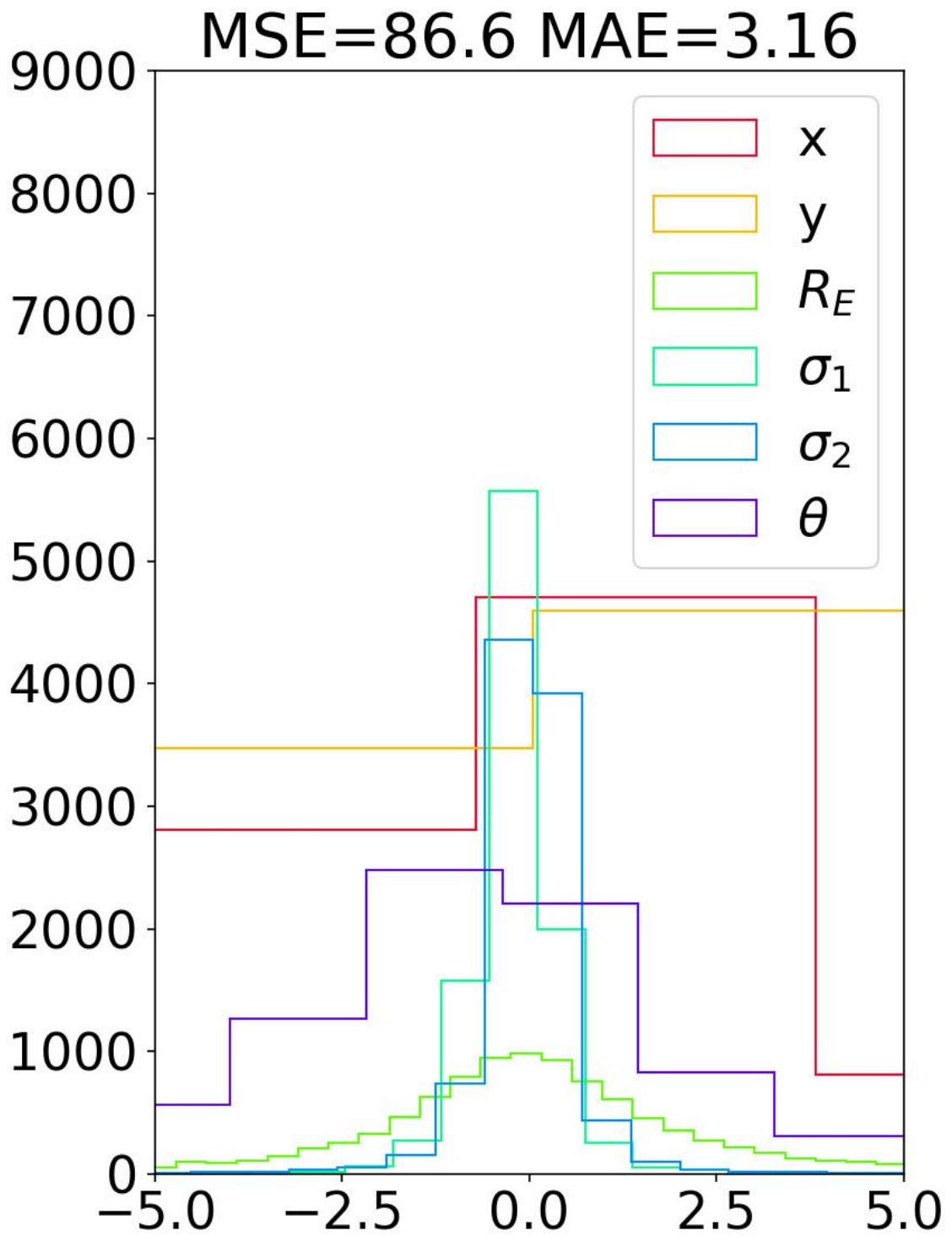


Figure C.33.: SqueezeNet histogram with 0.0001 learning rate and extra layers



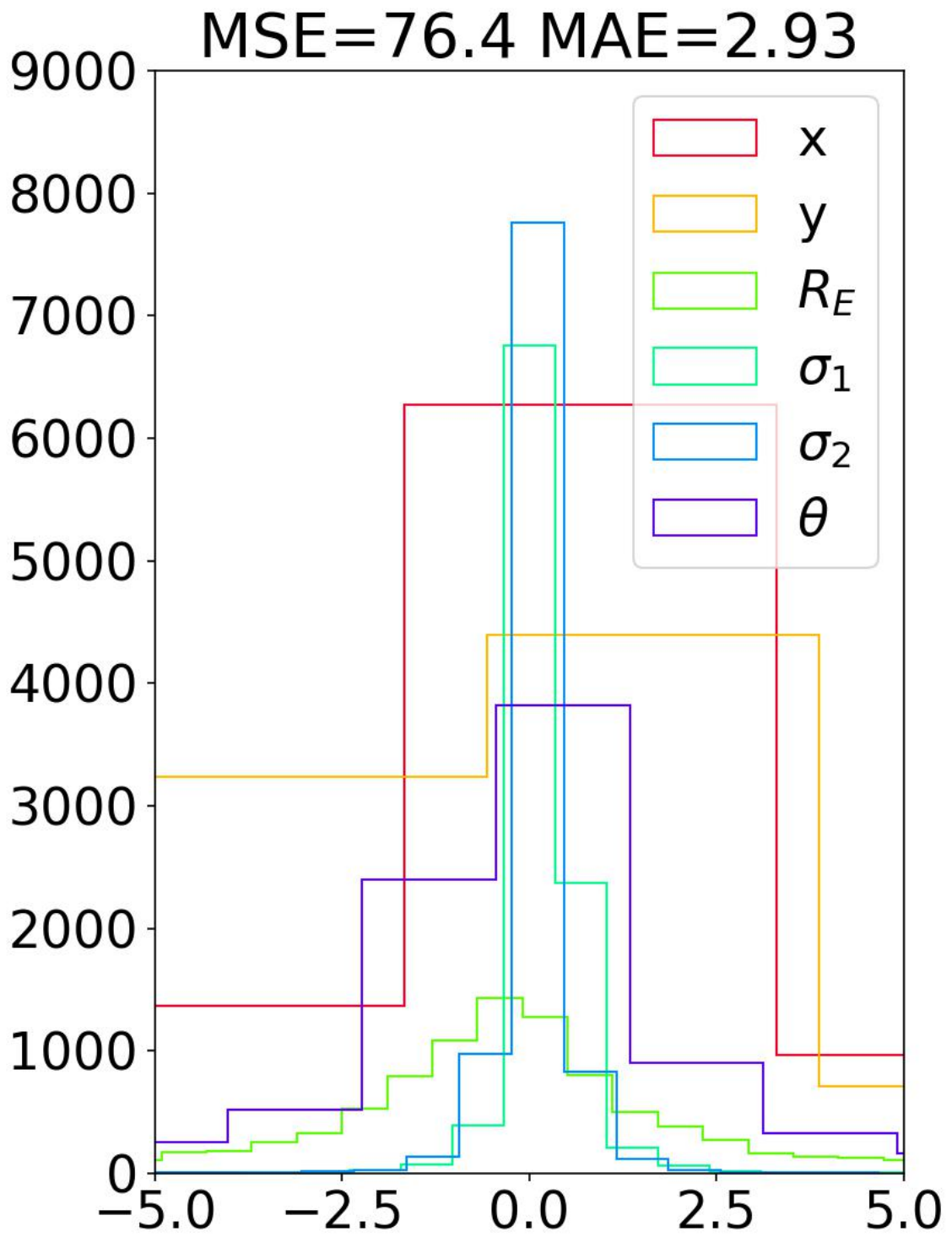


Figure C.34.: Swin-v2.b histogram with 0.0001 learning rate

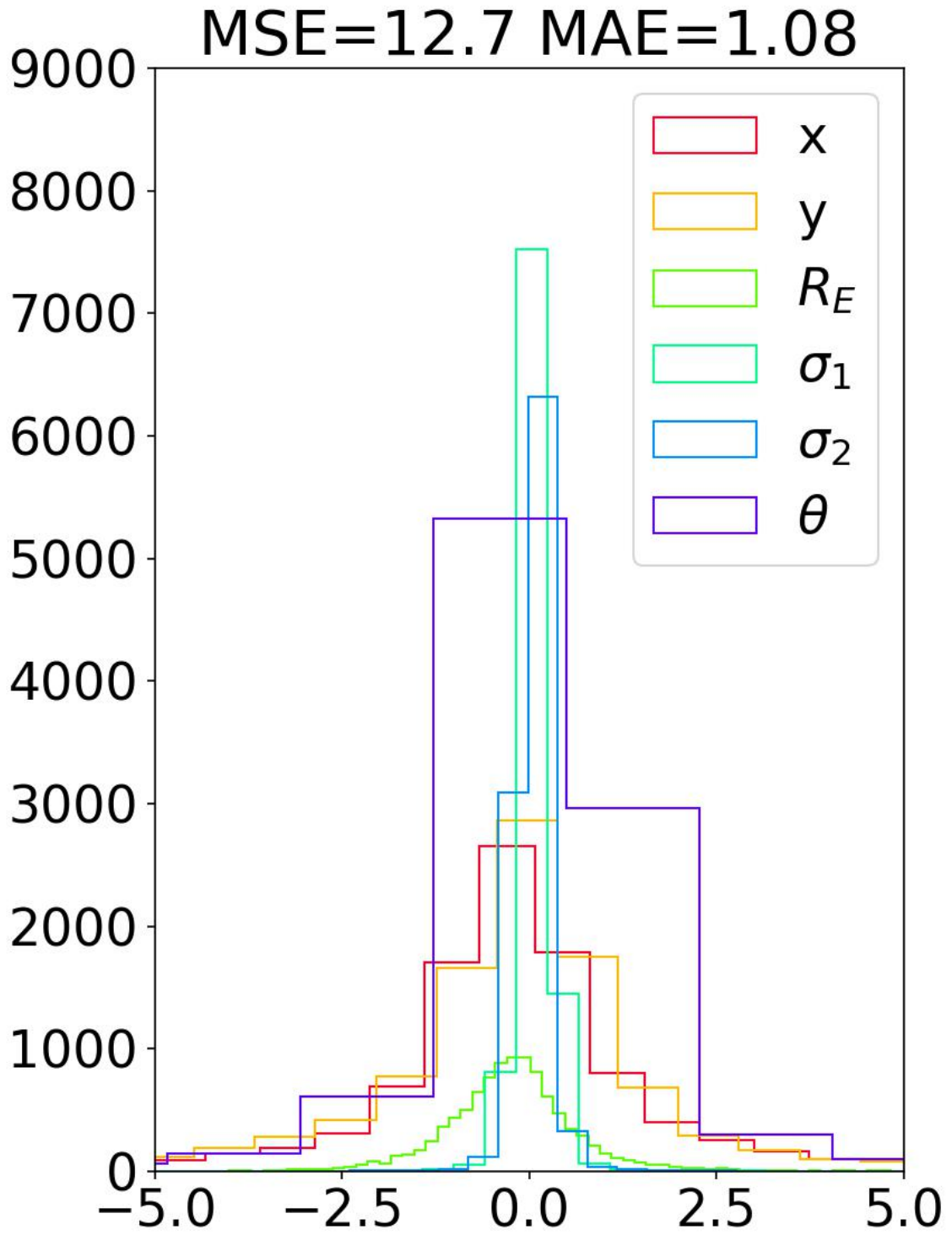


Figure C.35.: VGG-19\_BN histogram with 0.0001 learning rate

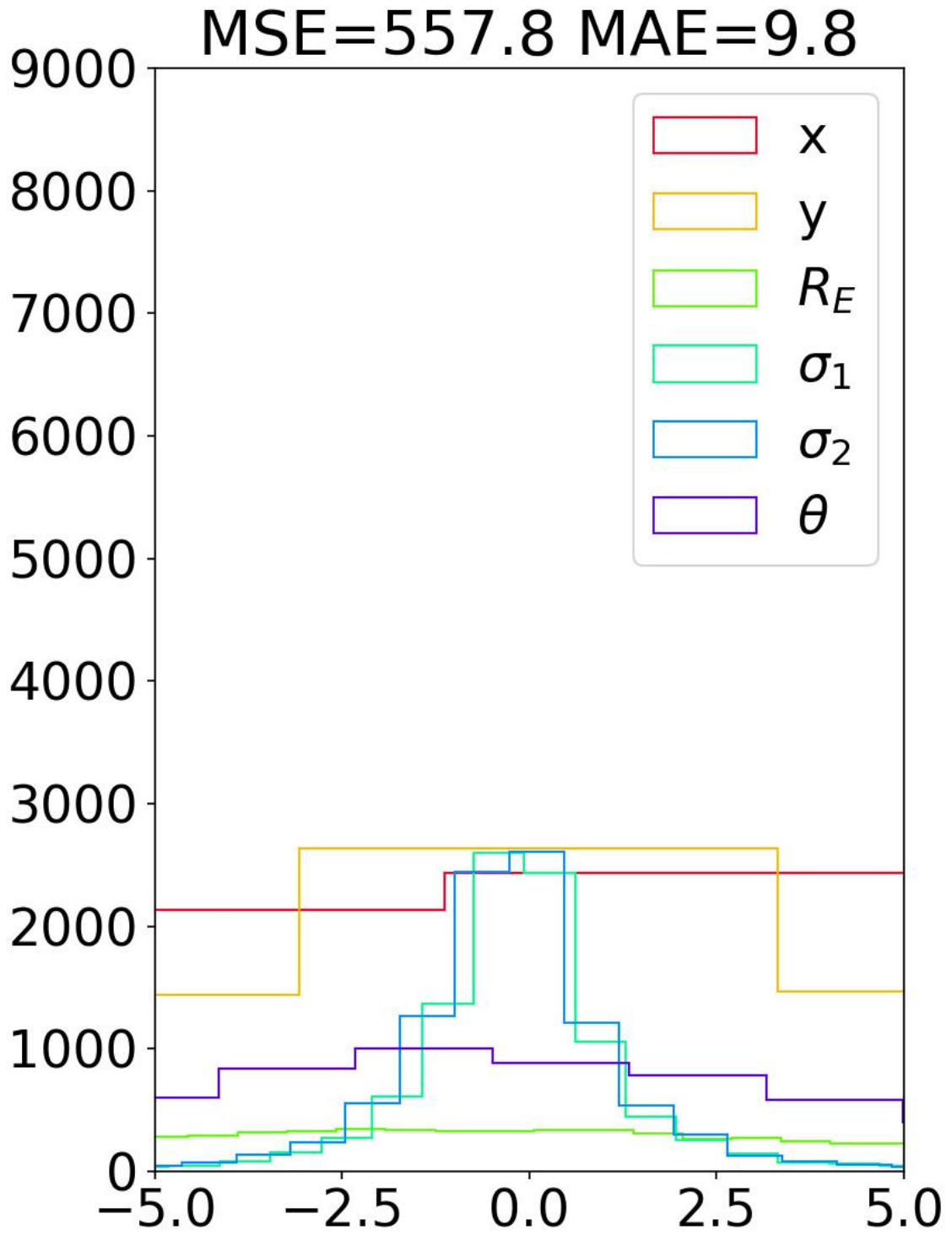


Figure C.36.: ViT histogram with 0.0001 learning rate

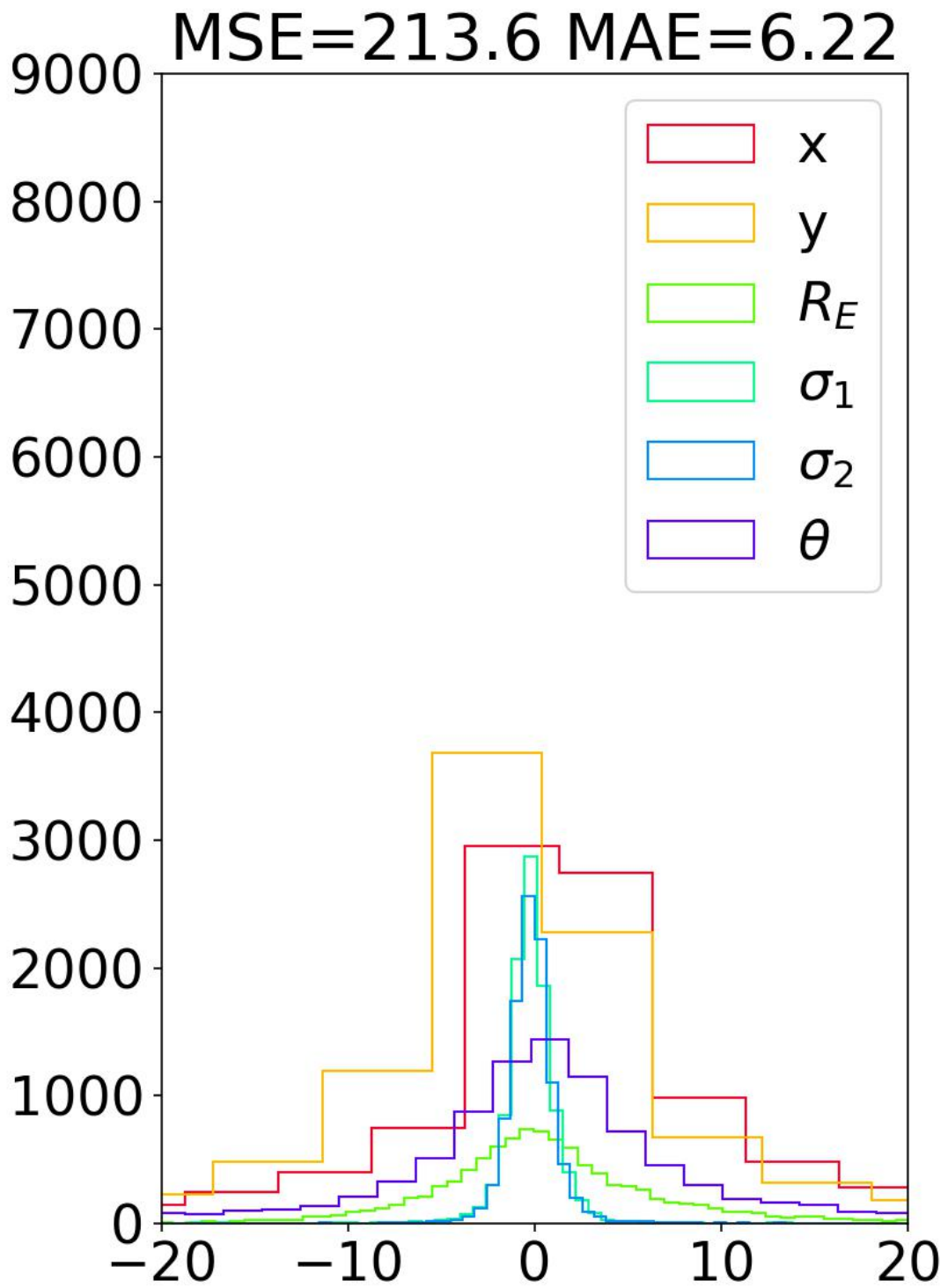


Figure C.37.: AlexNet histogram with 0.001 learning rate

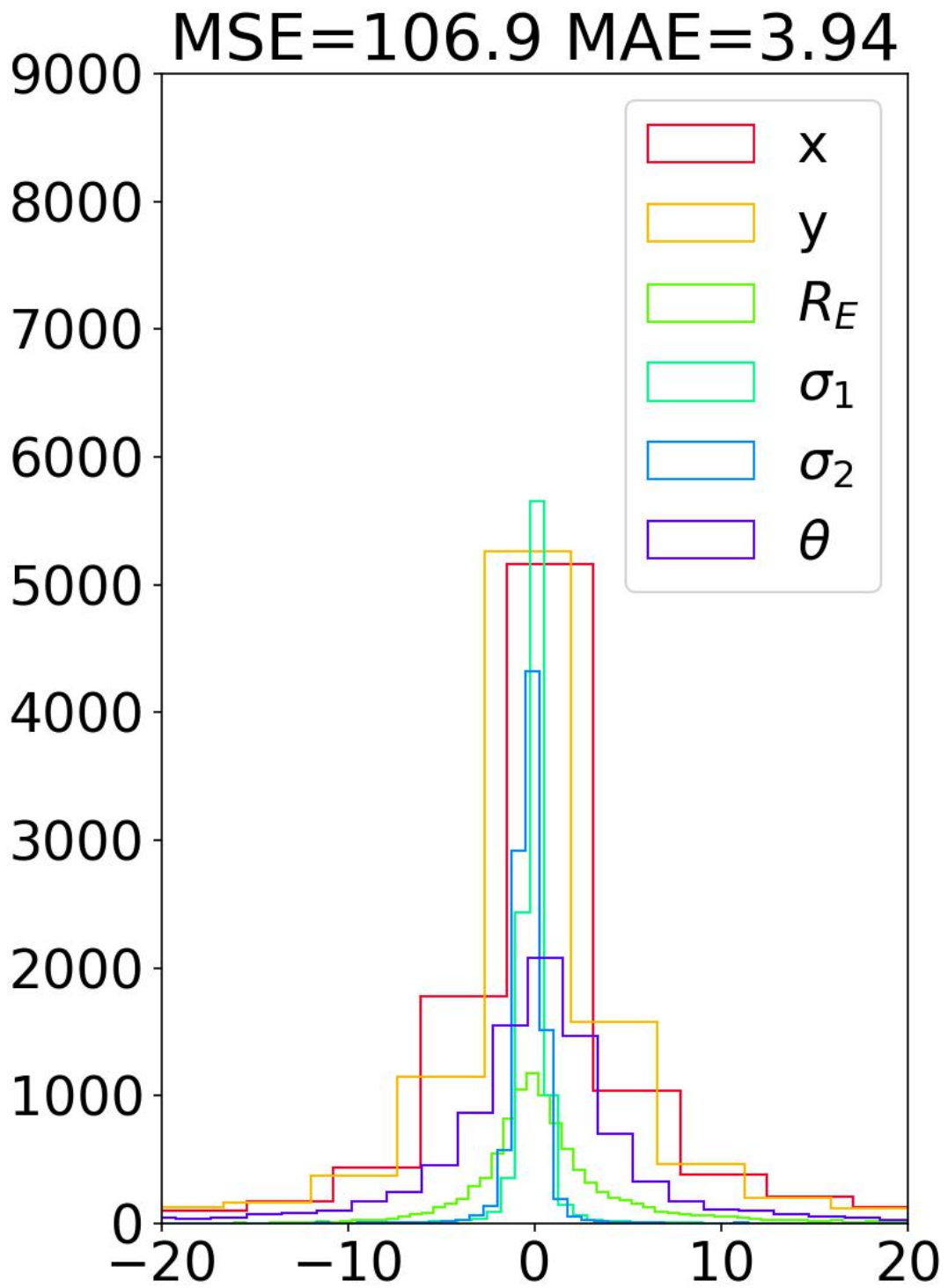


Figure C.38.: AlexNet histogram with 0.0001 learning rate

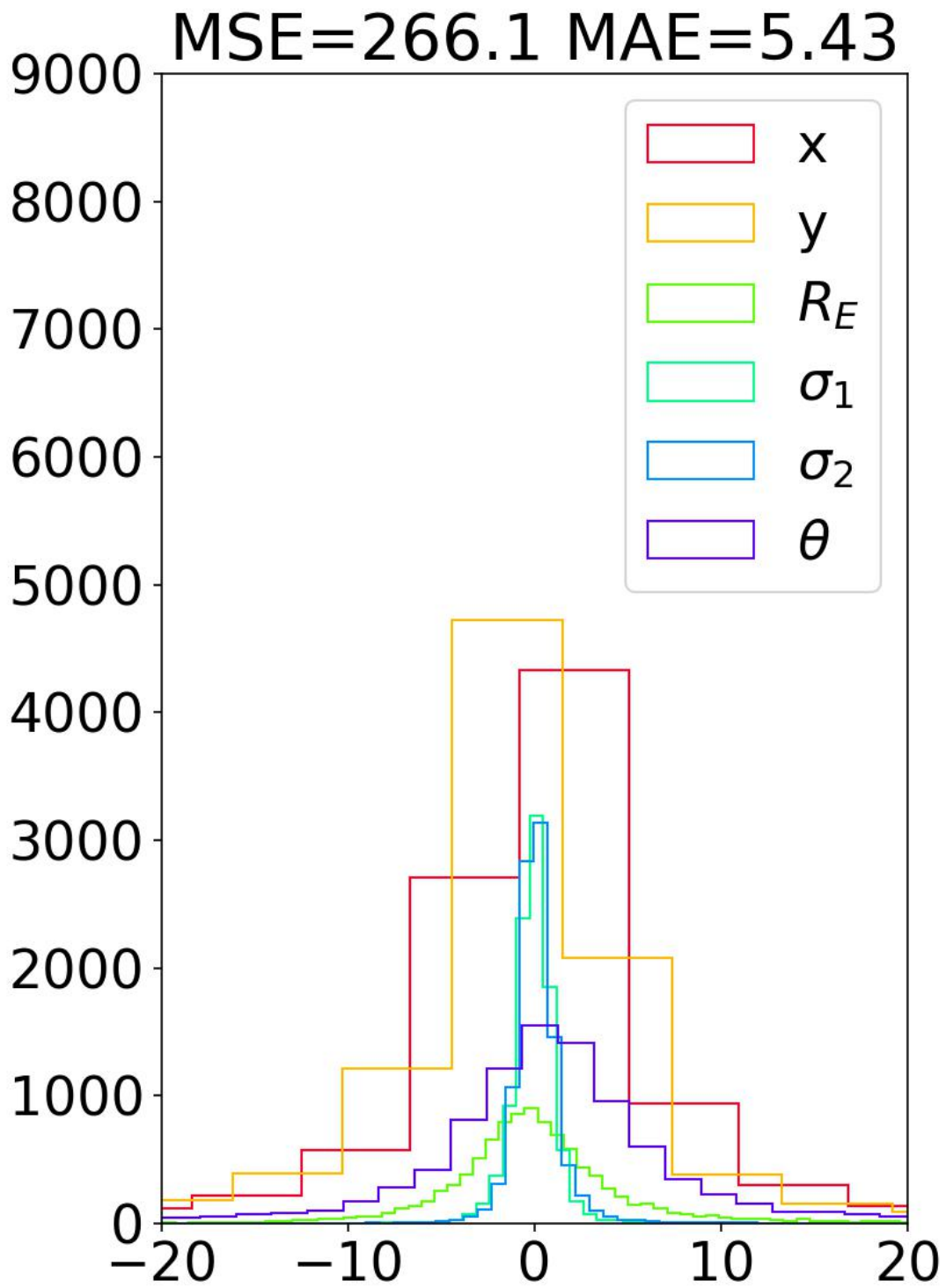


Figure C.39.: AlexNet histogram with 0.001 learning rate and extra layers

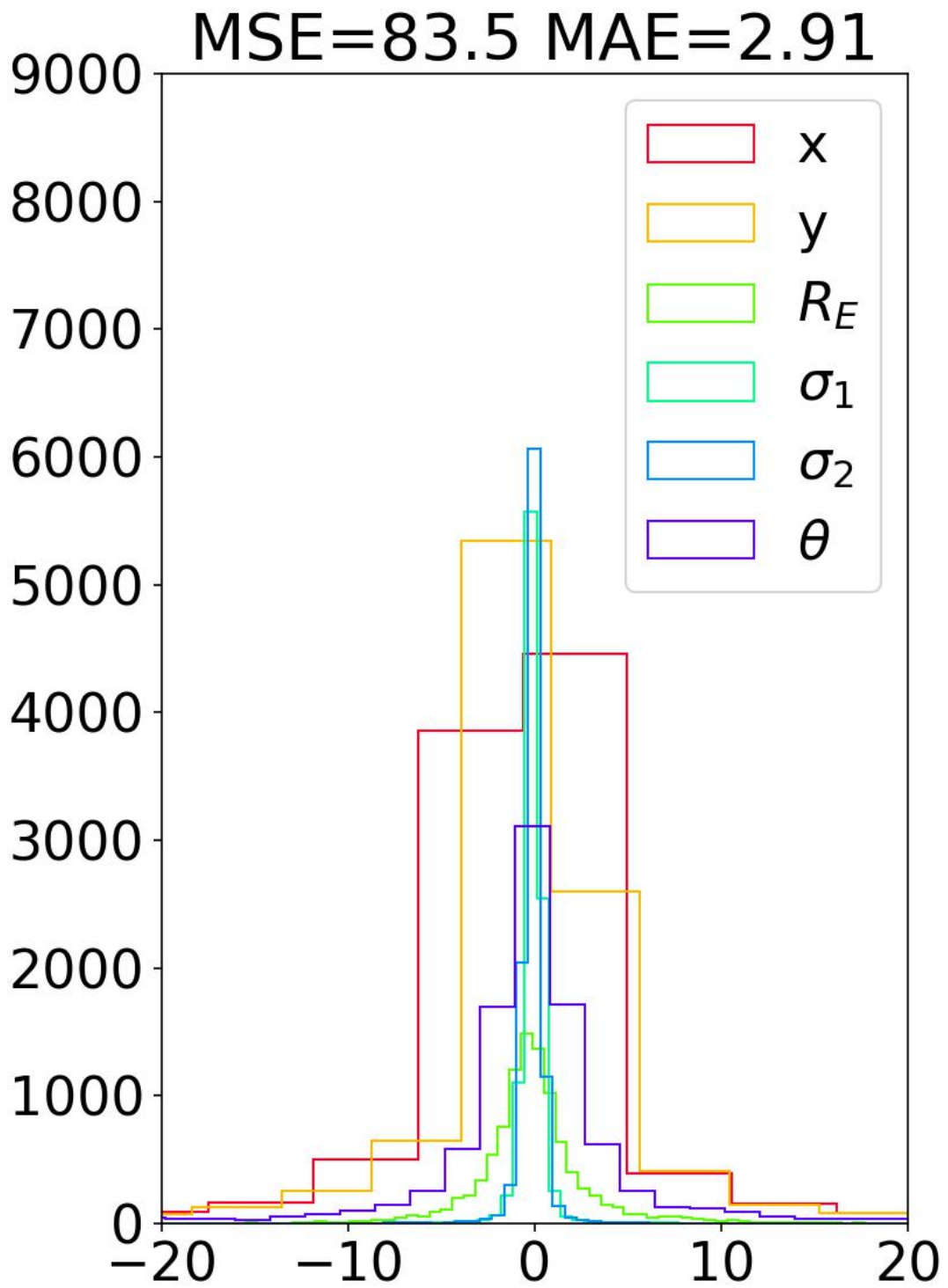


Figure C.40.: AlexNet histogram with 0.0001 learning rate and extra layers

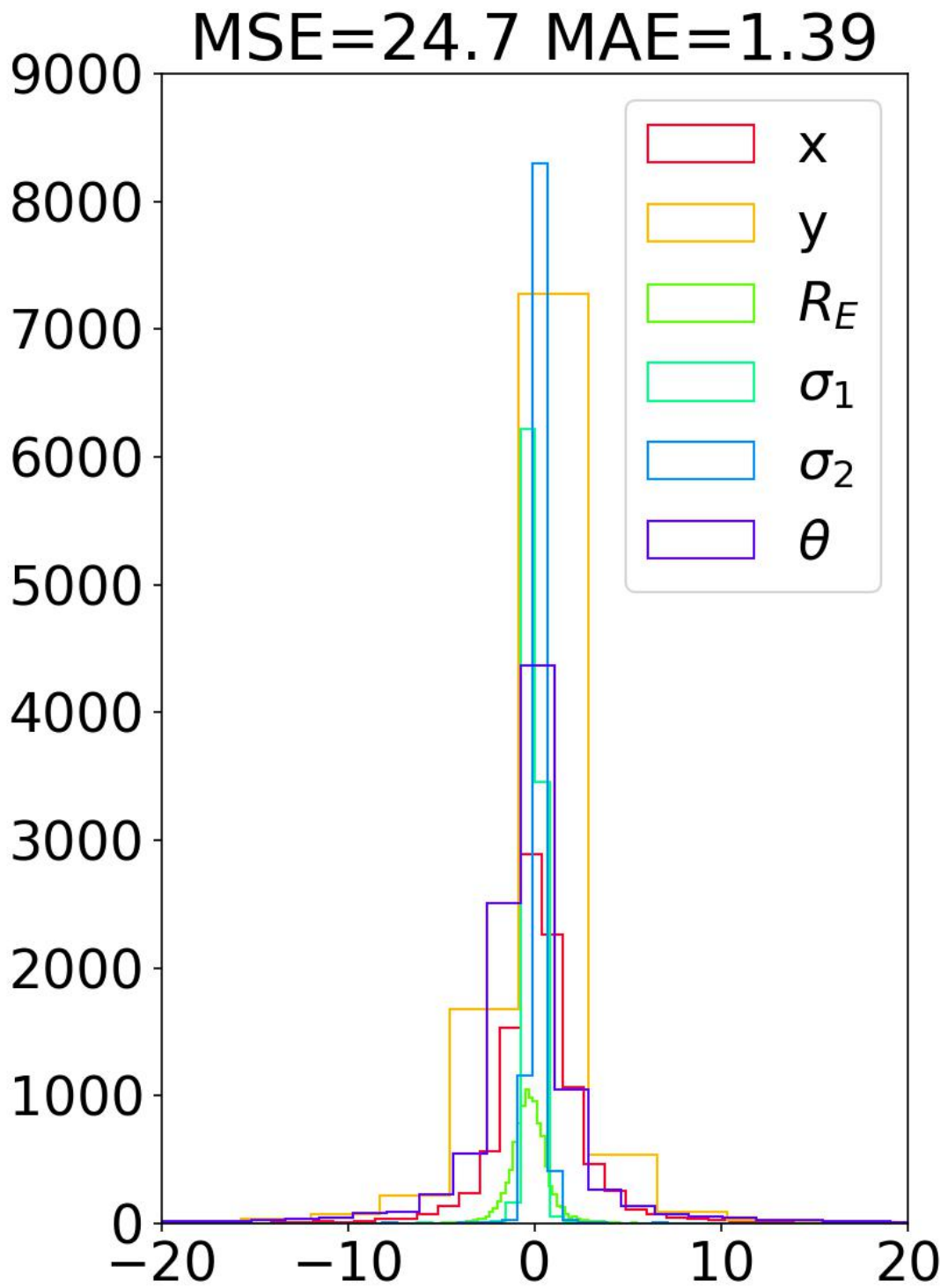


Figure C.41.: EfficientNet-B7 histogram with 0.0001 learning rate



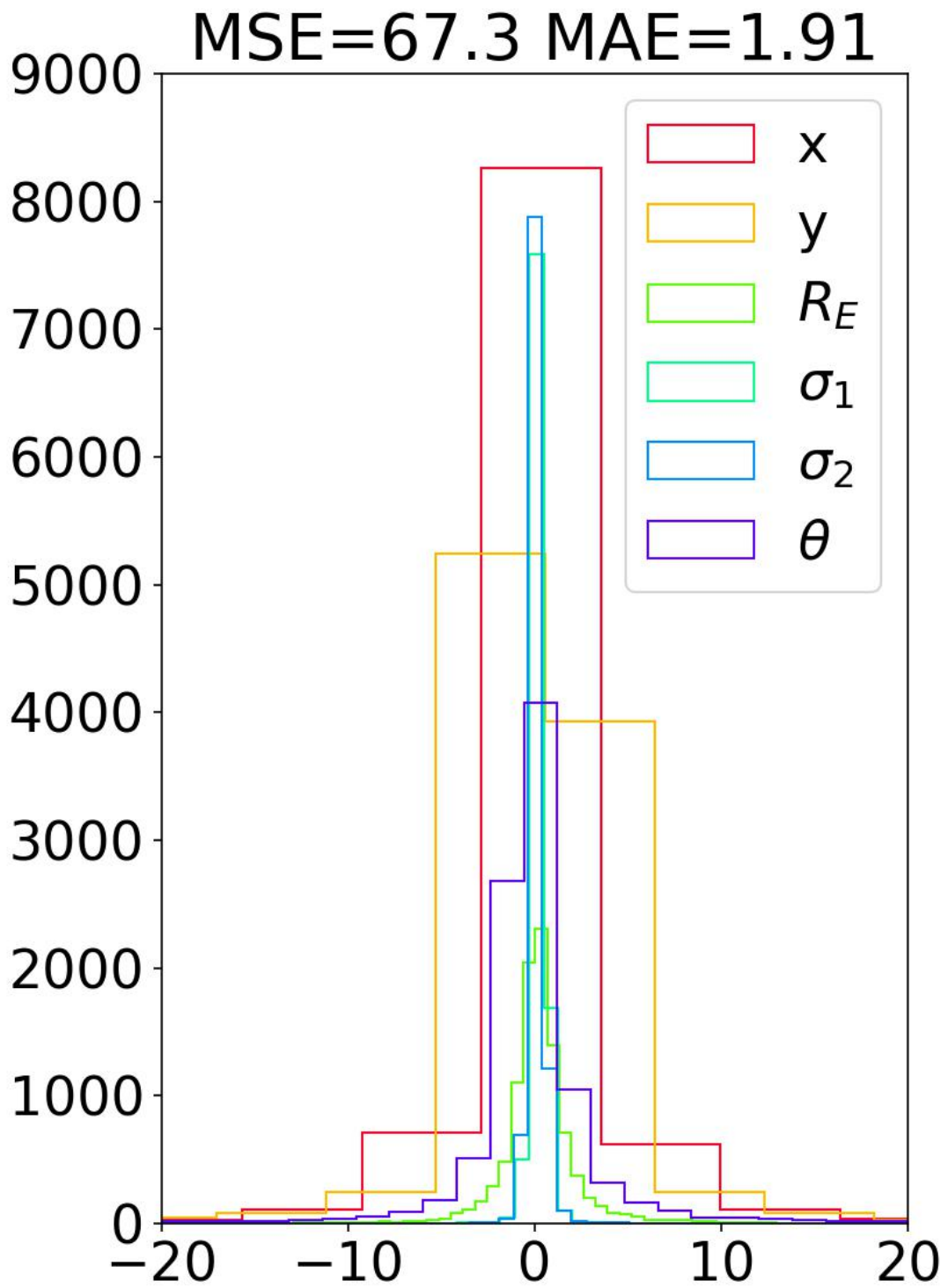


Figure C.42.: EfficientNet-v2.1 histogram with 0.0001 learning rate

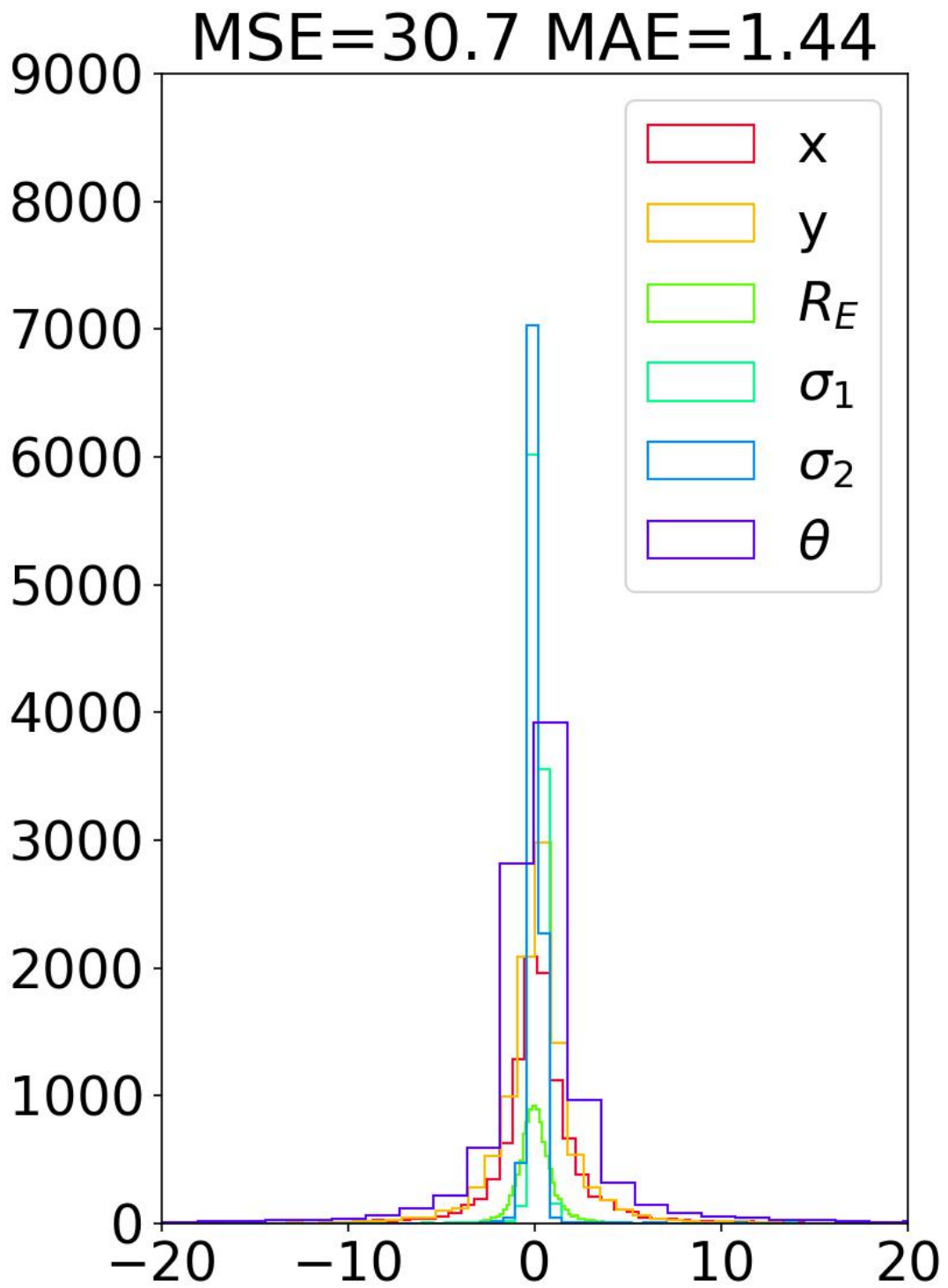


Figure C.43.: EfficientNet histogram with 0.0001 learning rate

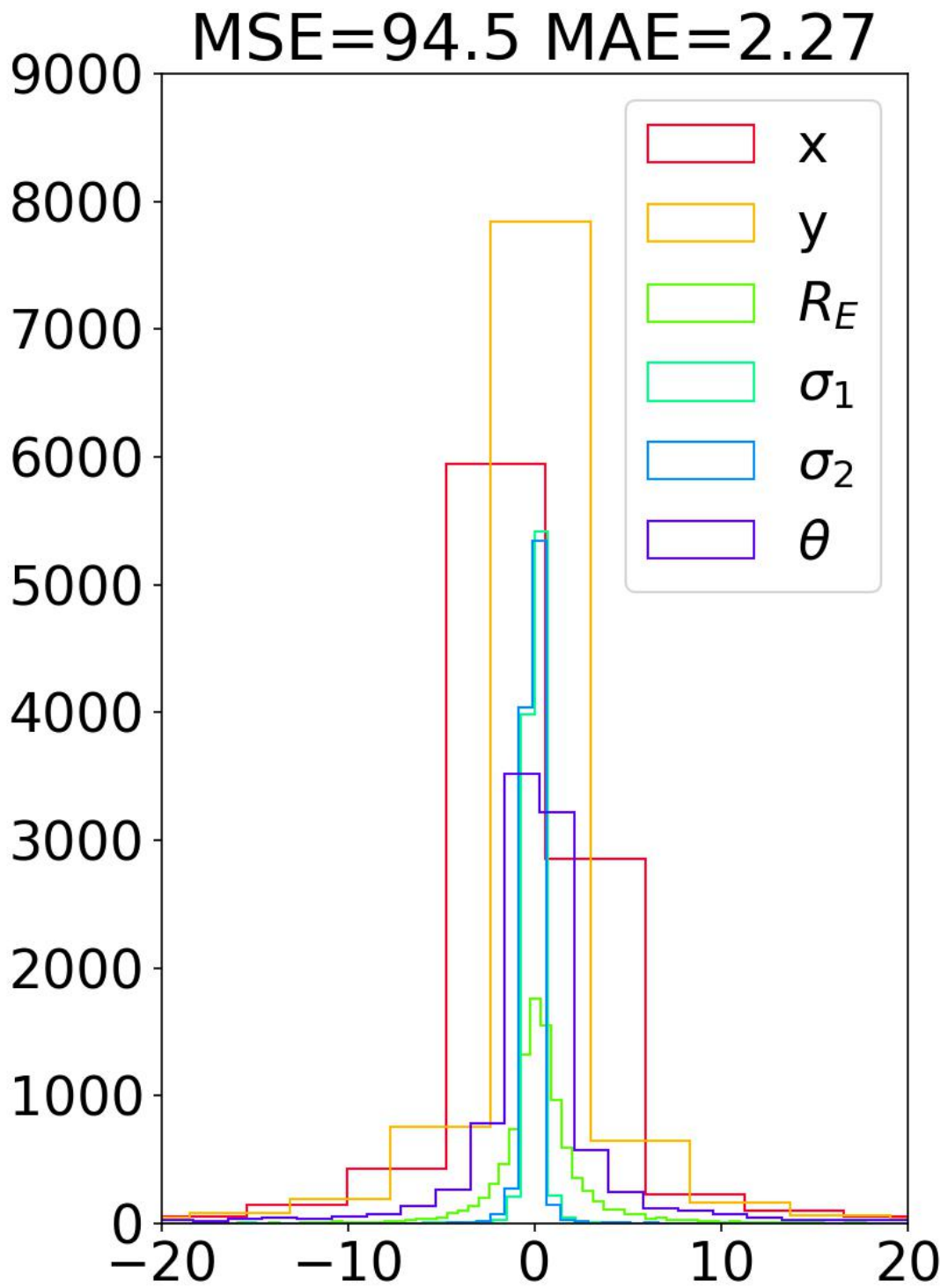


Figure C.44.: Inception-v3 histogram with 0.001 learning rate

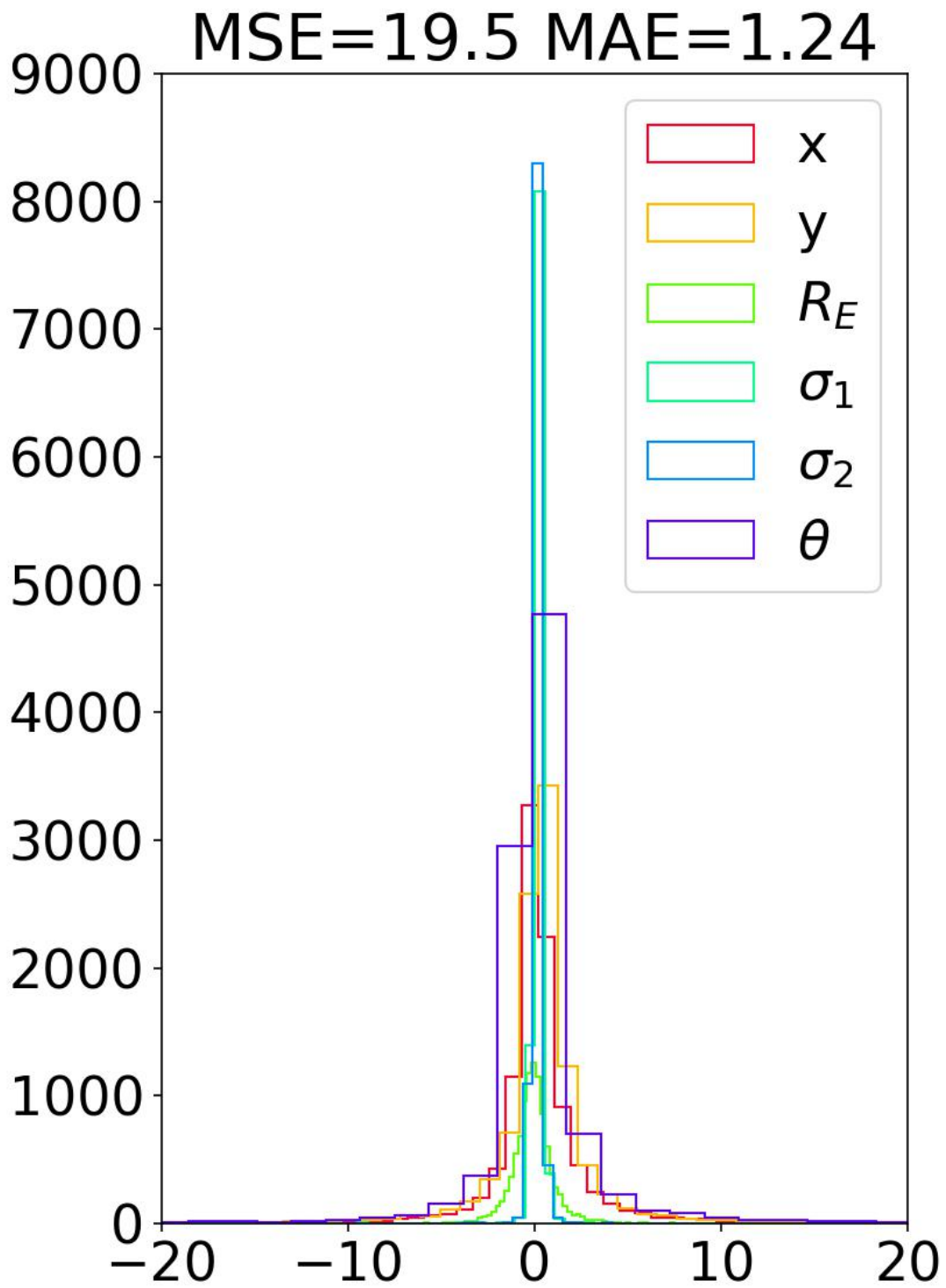


Figure C.45.: Inception-v3 histogram with 0.0001 learning rate.

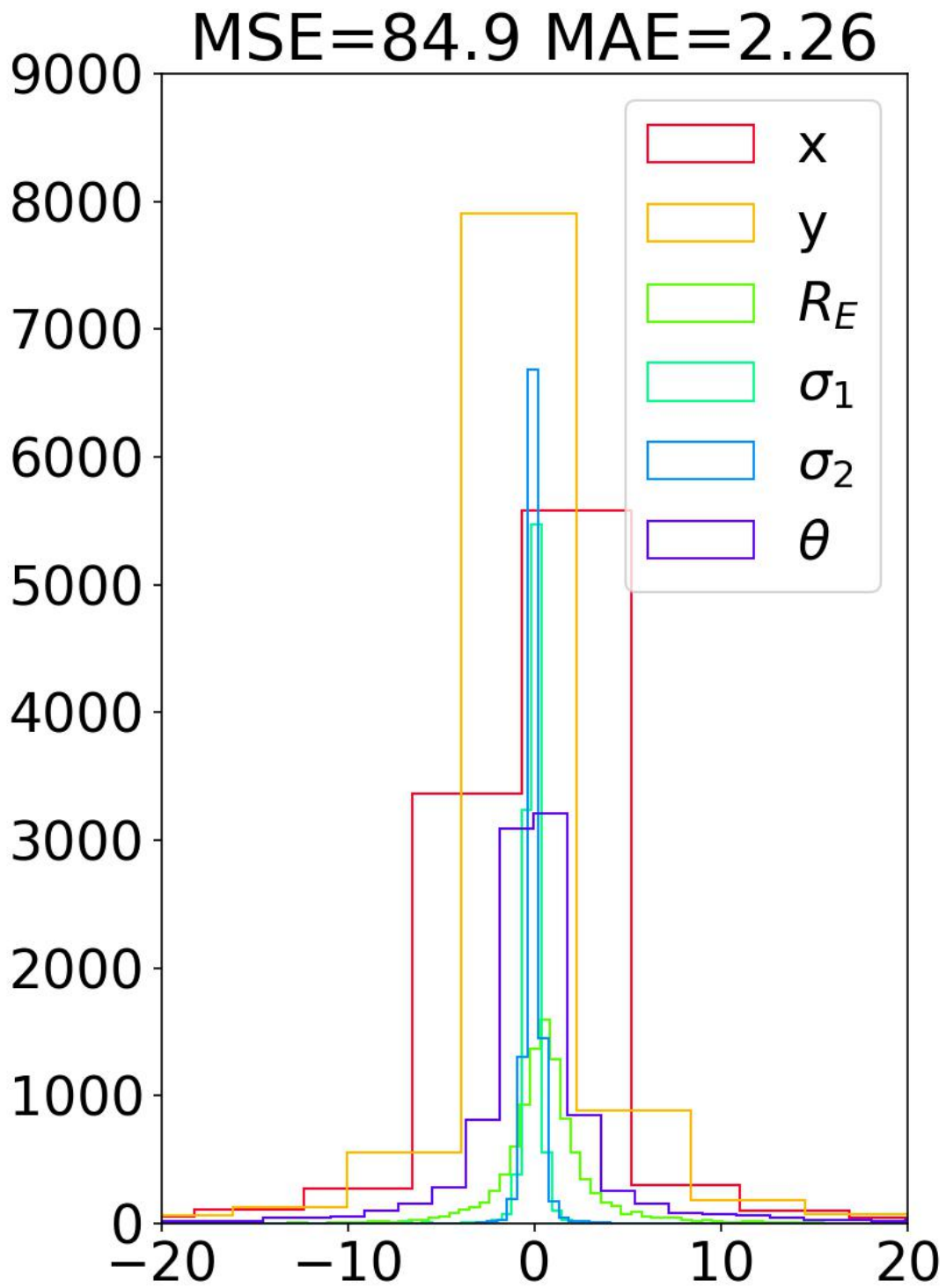


Figure C.46.: Inception-v3 histogram pre-trained with 0.001 learning rate.

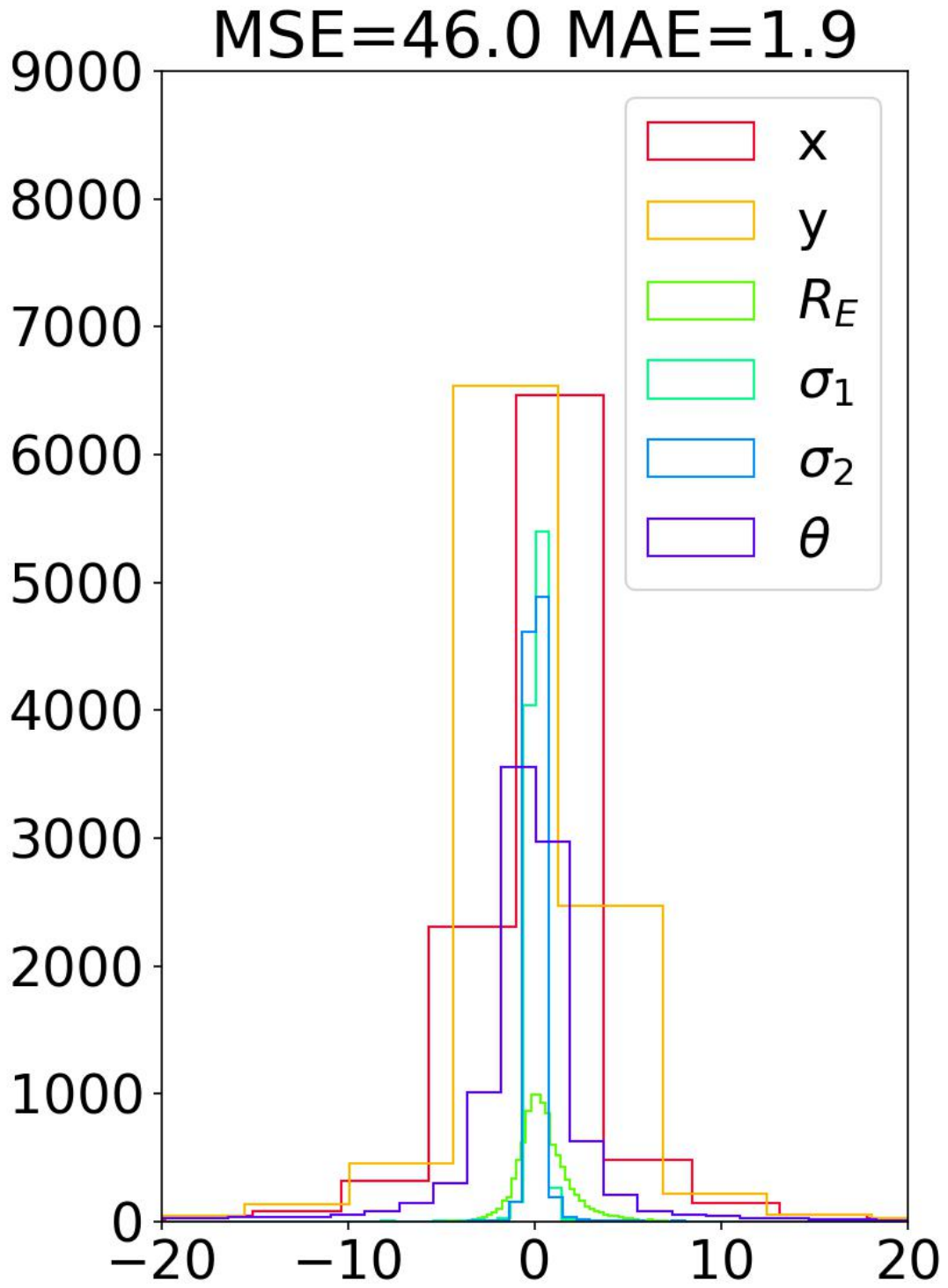


Figure C.47.: Inception-v3 histogram with 0.001 learning rate and extra layers.

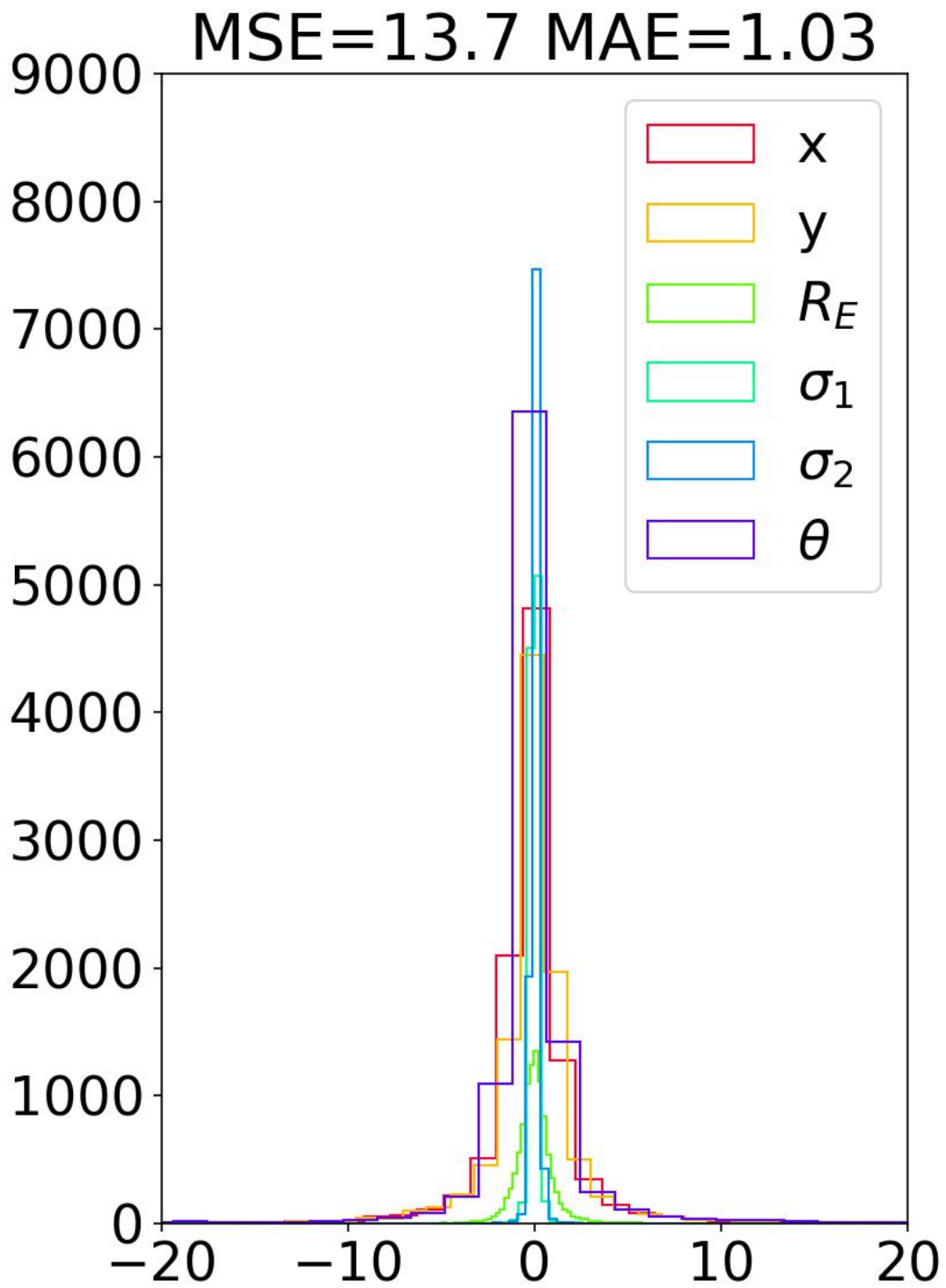


Figure C.48.: Inception-v3 histogram pre-trained with 0.0001 learning rate.

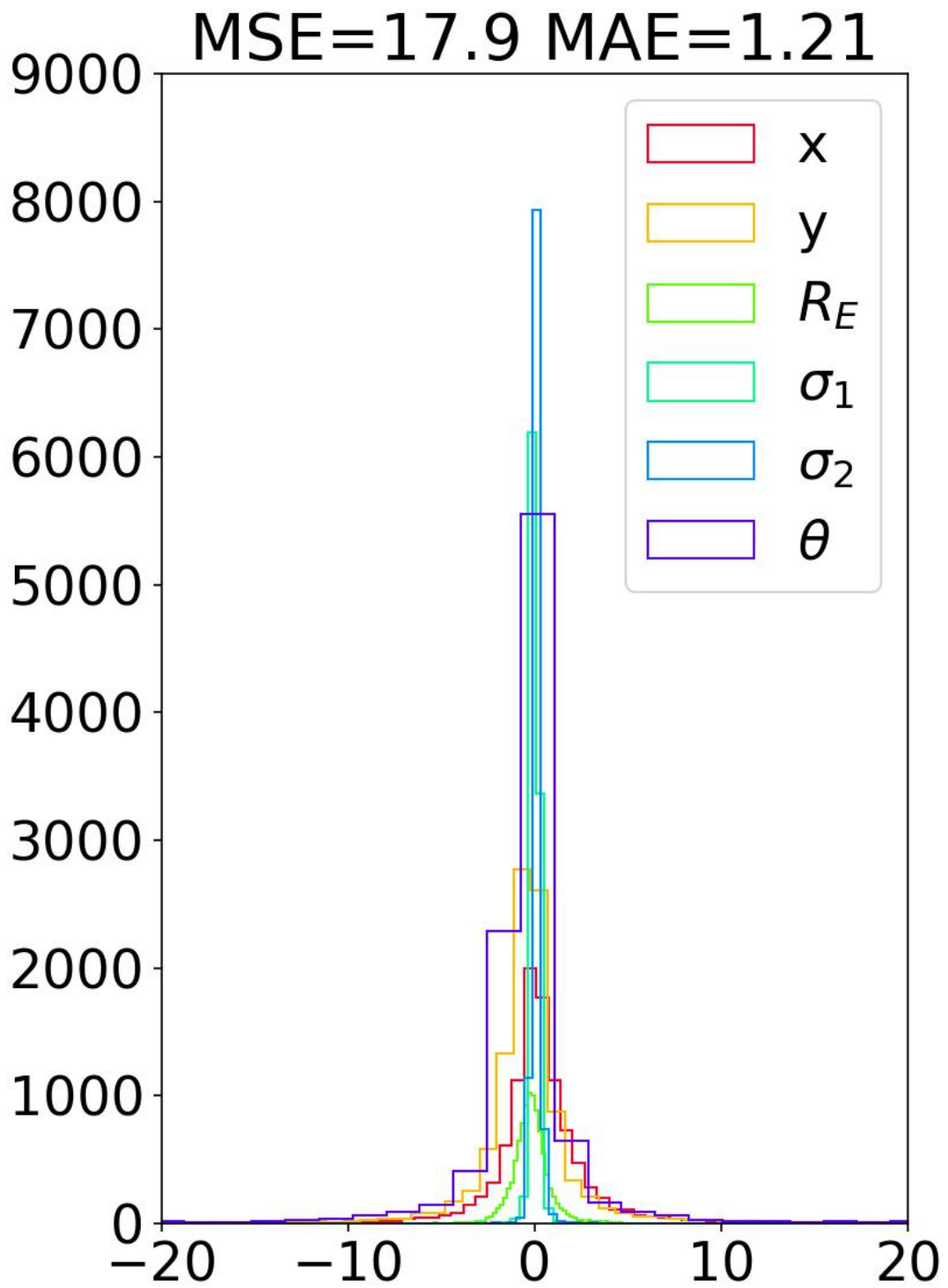


Figure C.49.: Inception-v3 histogram with 0.0001 learning rate and extra layers.



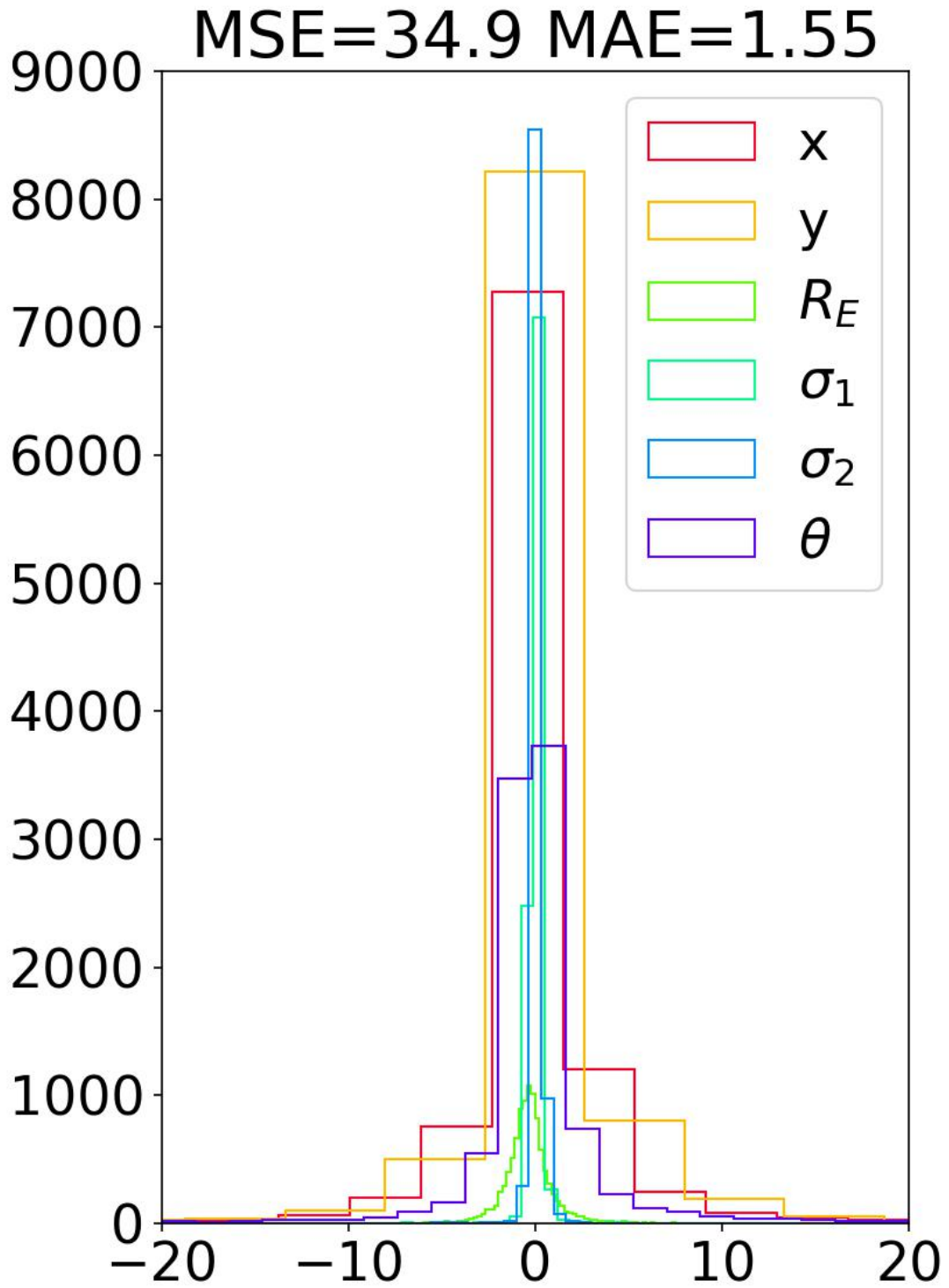


Figure C.50.: Inception-v3 histogram pre-trained with 0.00001 learning rate.

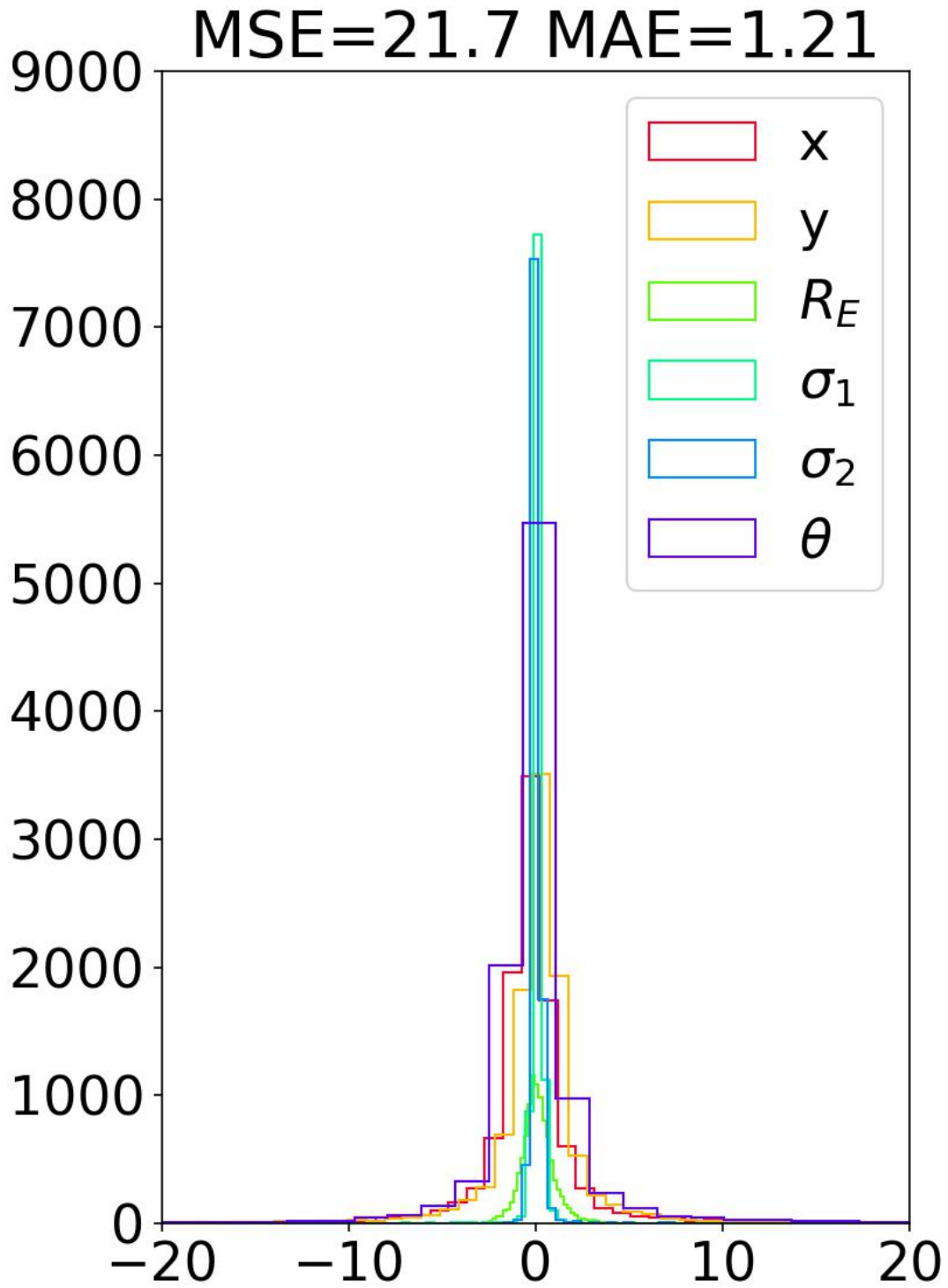


Figure C.51.: Inception-v3 histogram pre-trained with 0.0001 learning rate attempt 2.

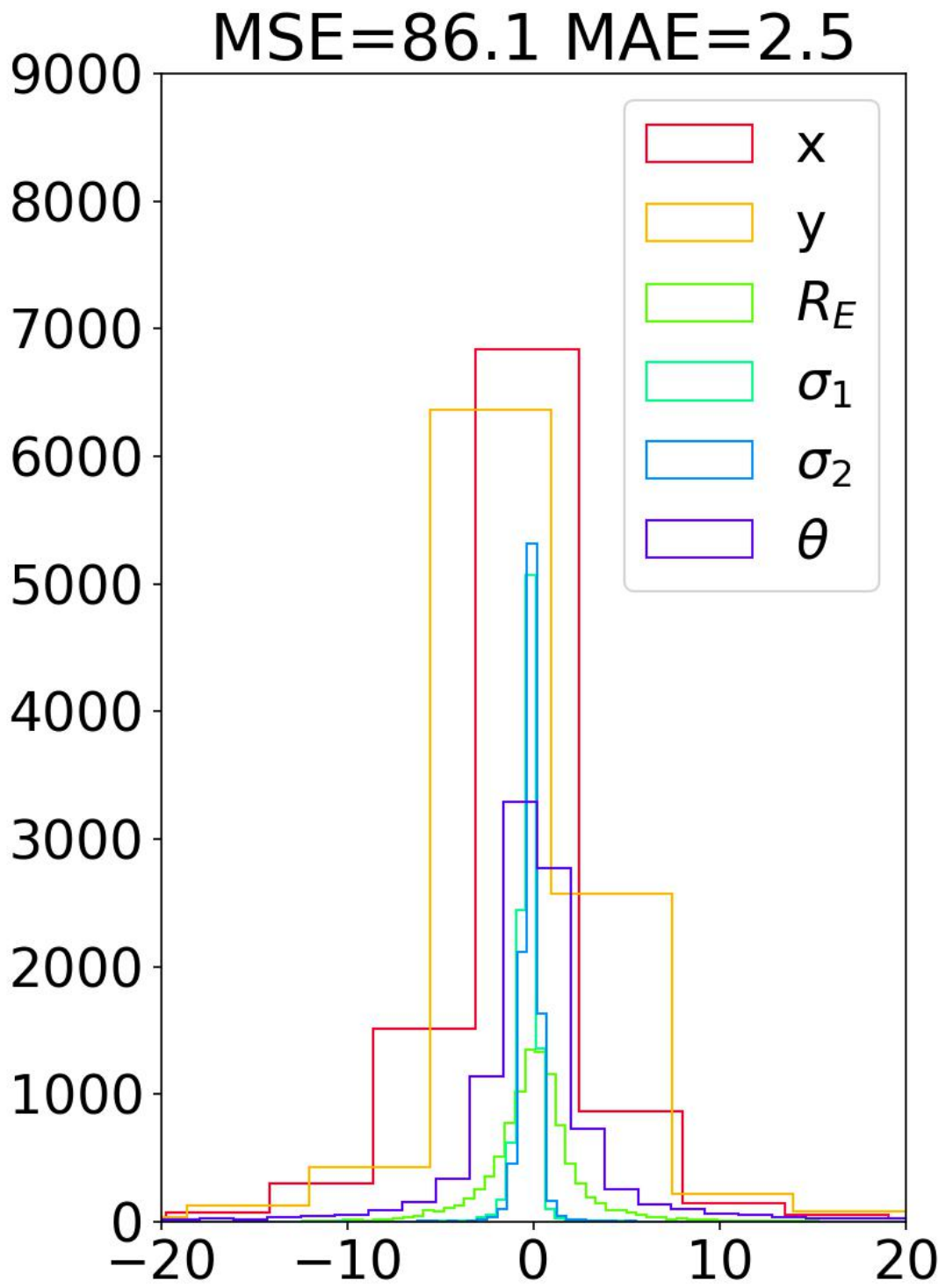


Figure C.52.: Inception-v3 histogram pre-trained with 0.001 learning rate and extra layers.

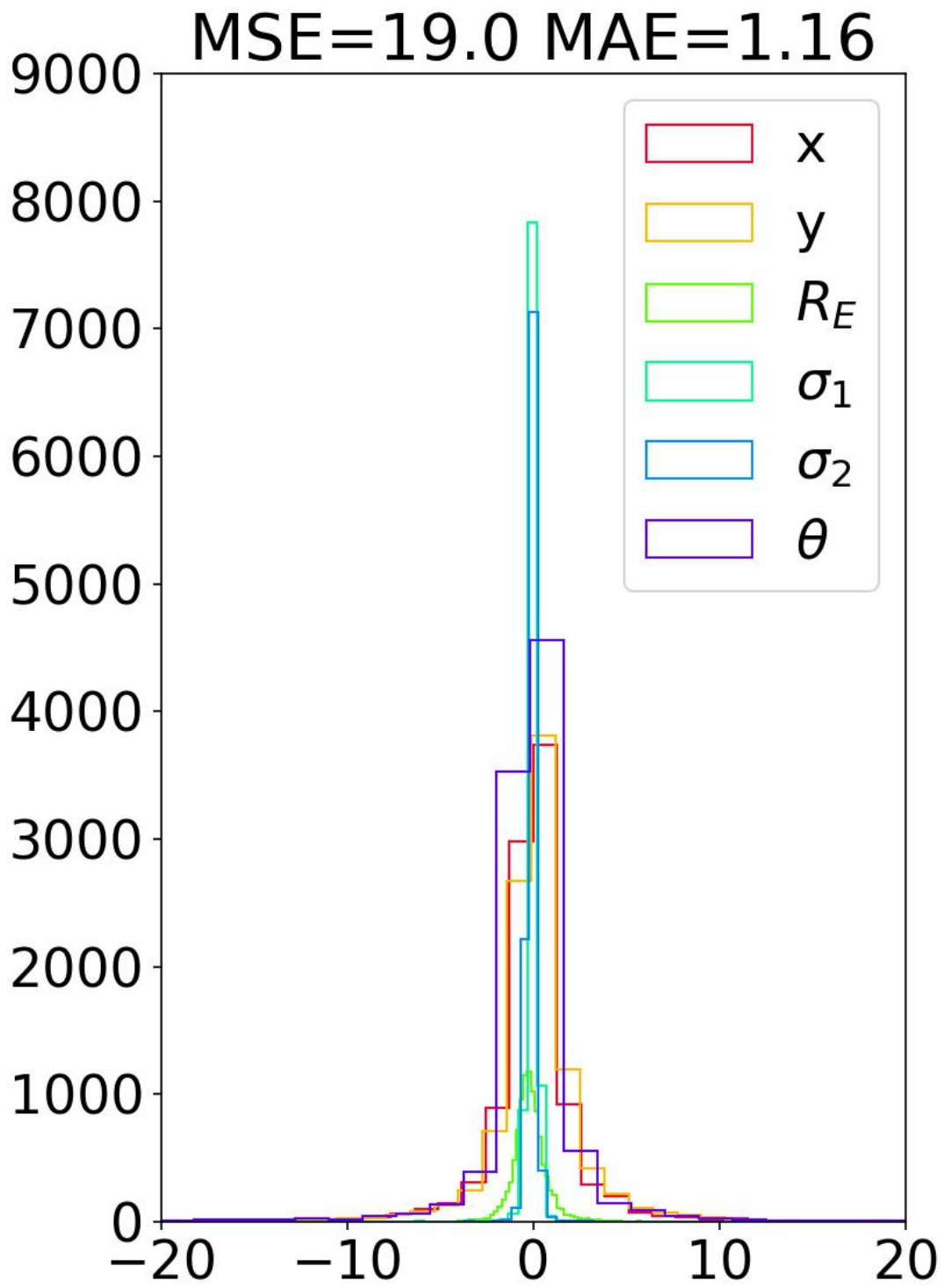


Figure C.53.: Inception-v3 histogram pre-trained with 0.0001 learning rate and extra layers.

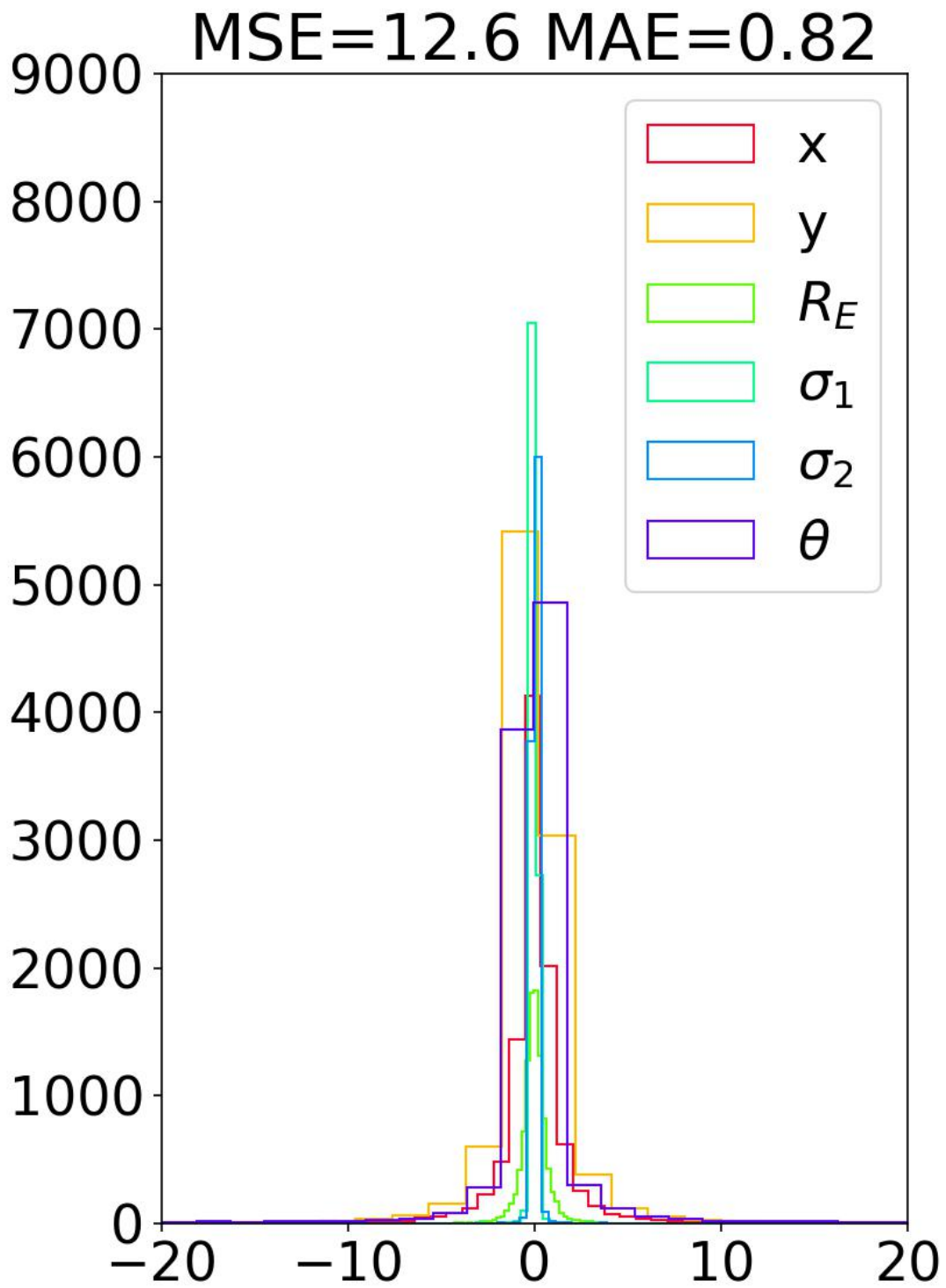


Figure C.54.: Inception-v3 histogram pre-trained with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epochs.

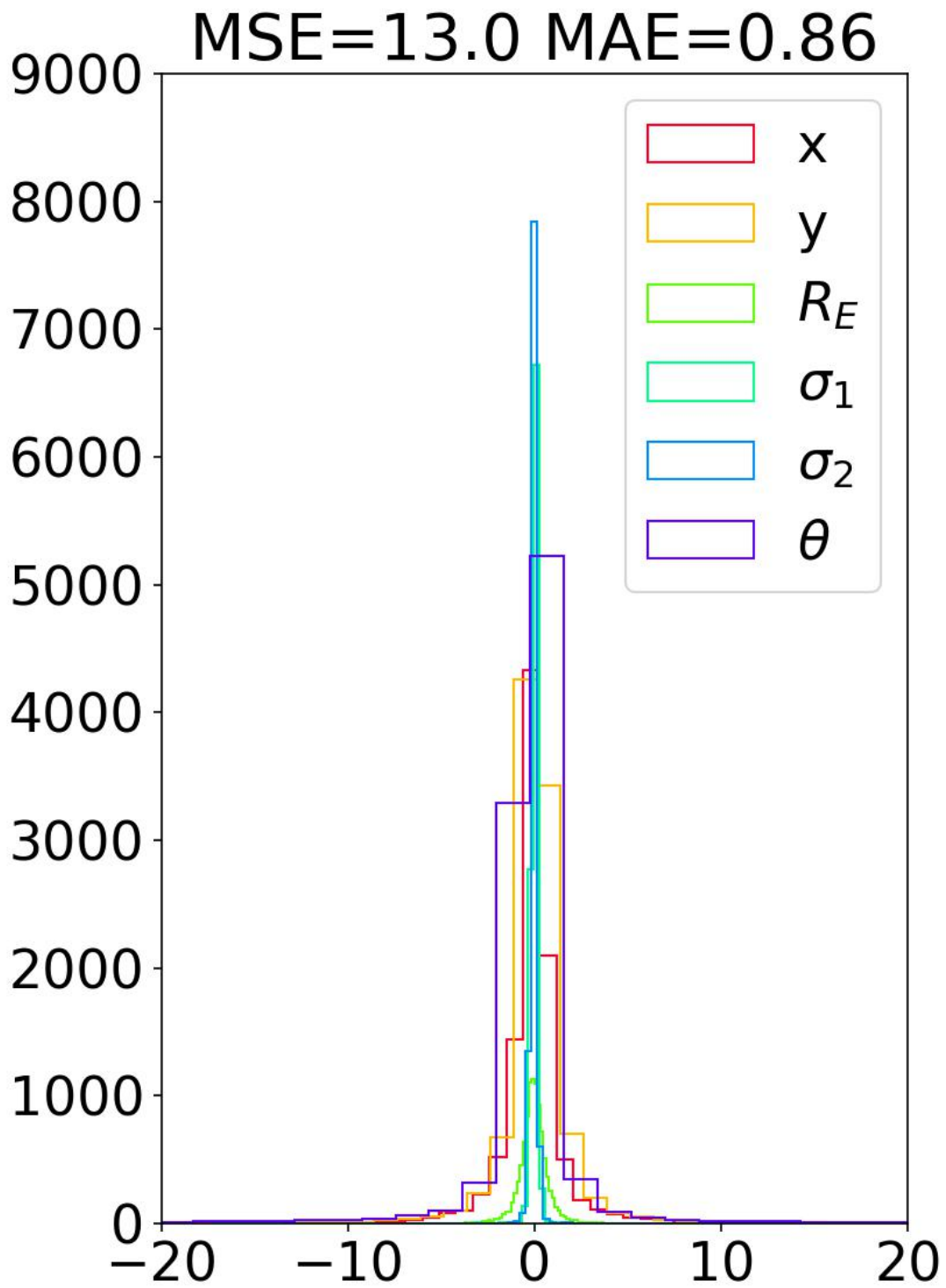


Figure C.55.: Inception-v3 histogram with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epochs, all with extra layers.

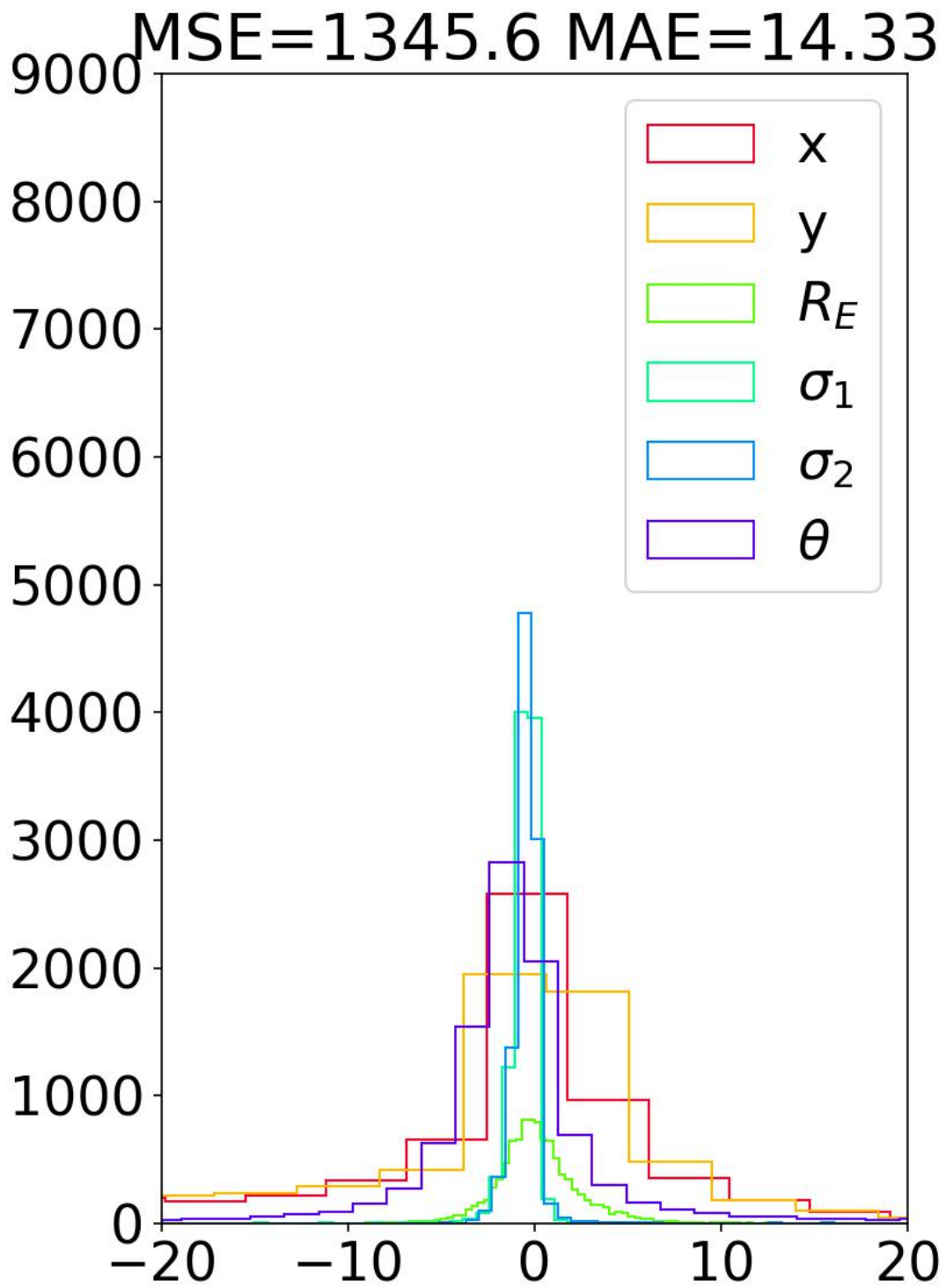


Figure C.56.: SqueezeNet histogram with 0.0001 learning rate.

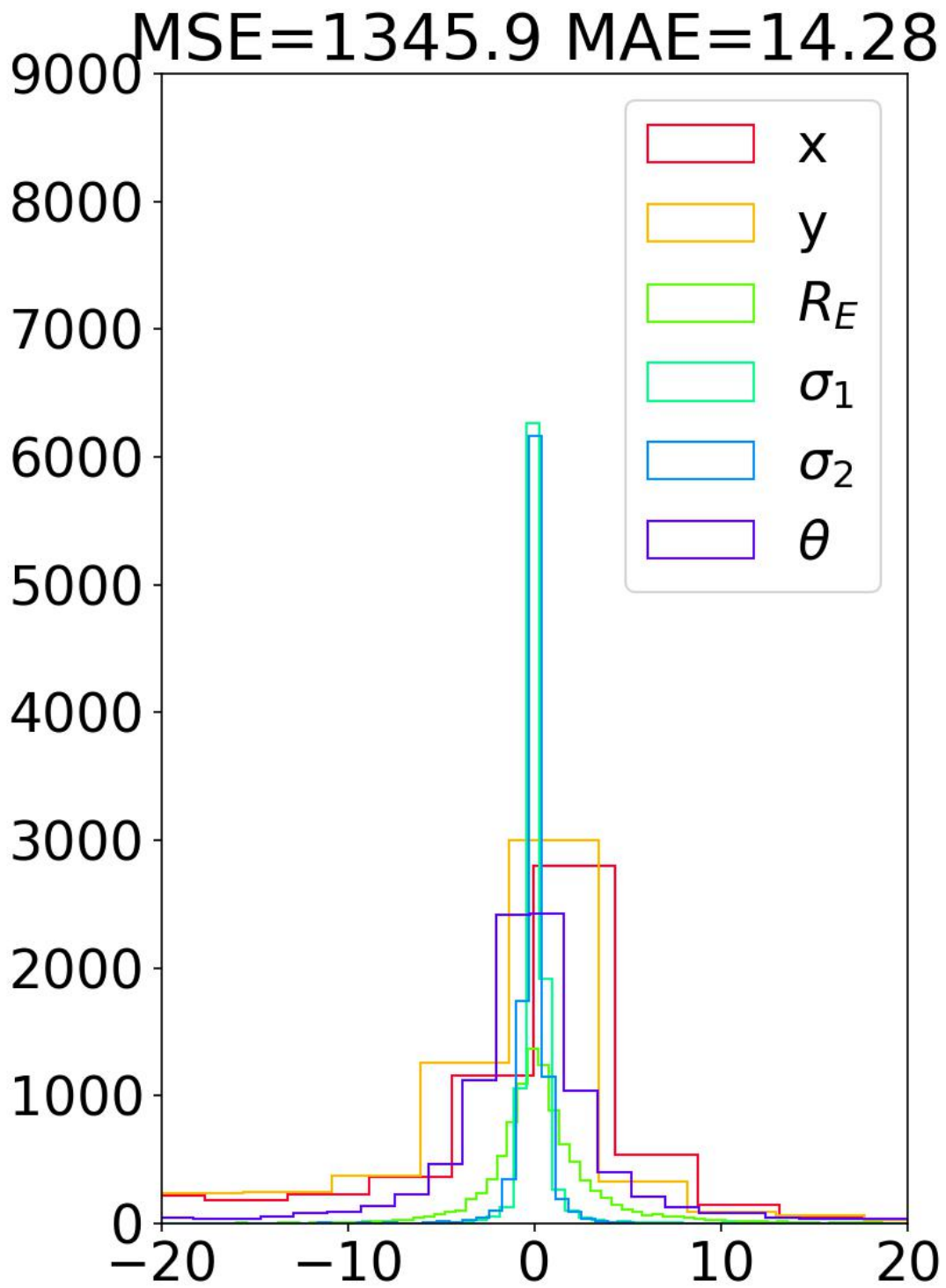


Figure C.57.: SqueezeNet histogram pre-trained with 0.0001 learning rate.



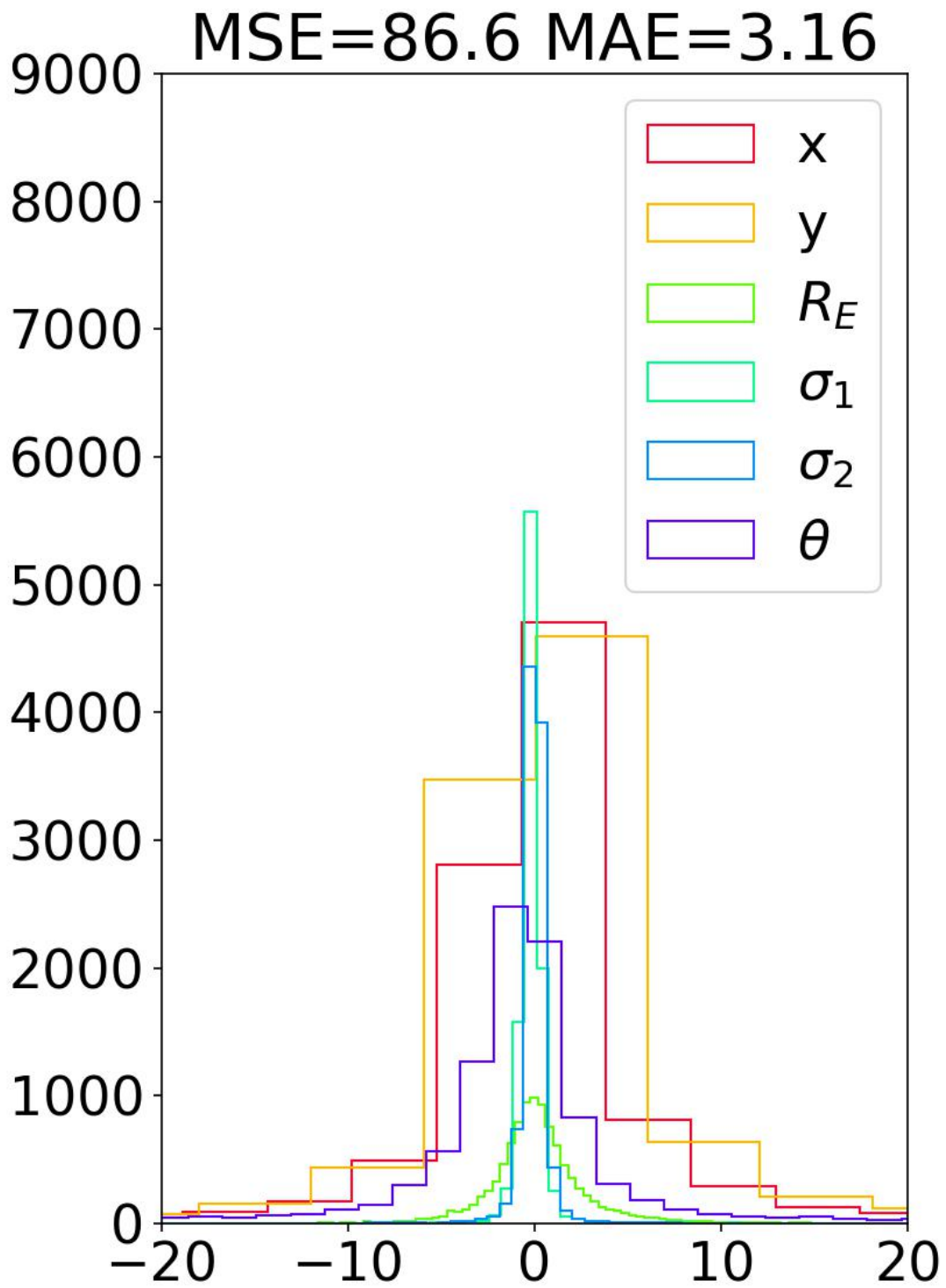


Figure C.58.: SqueezeNet histogram with 0.0001 learning rate and extra layers.

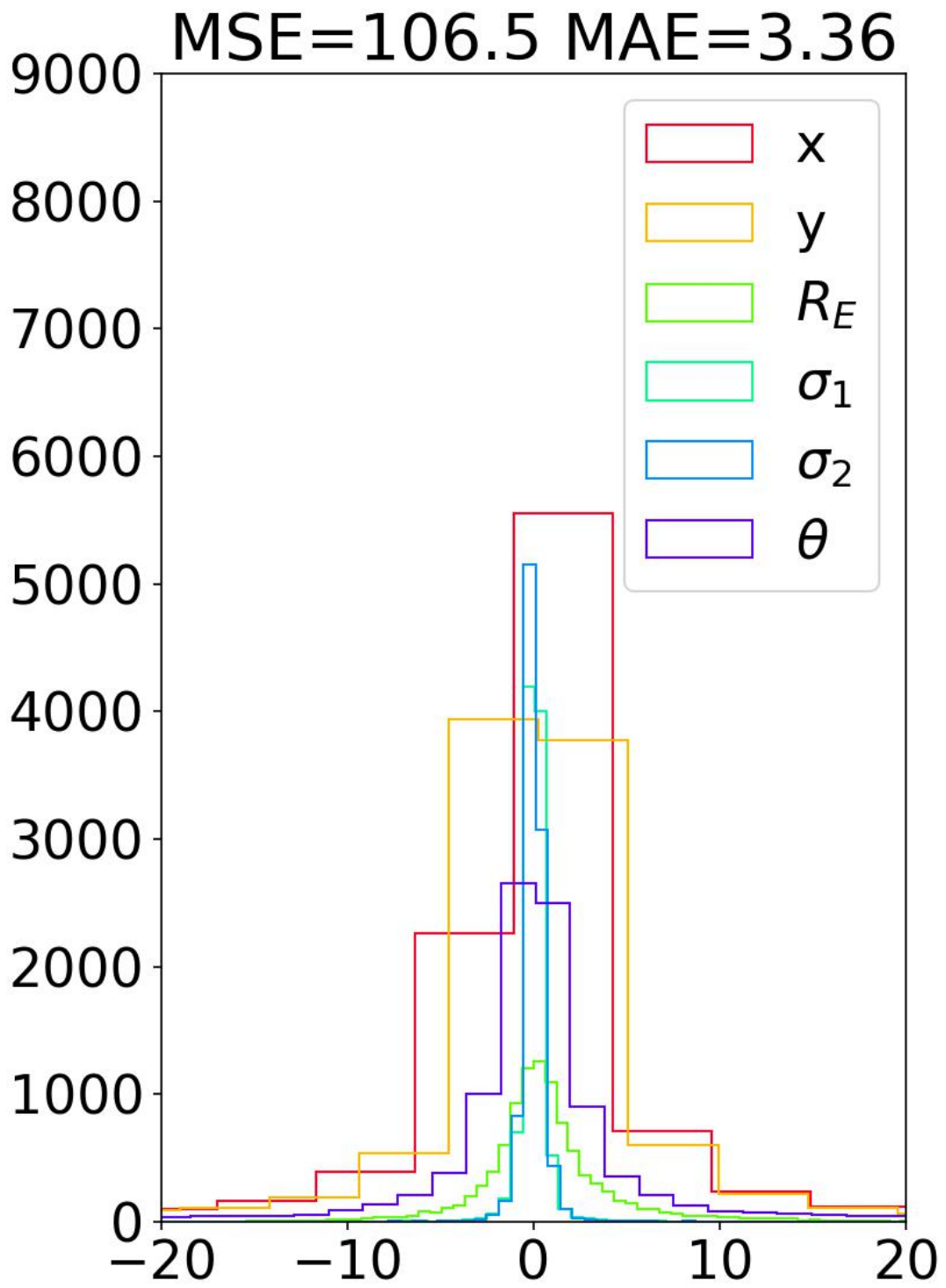


Figure C.59.: SqueezeNet histogram pre-trained with 0.0001 learning rate and extra layers.

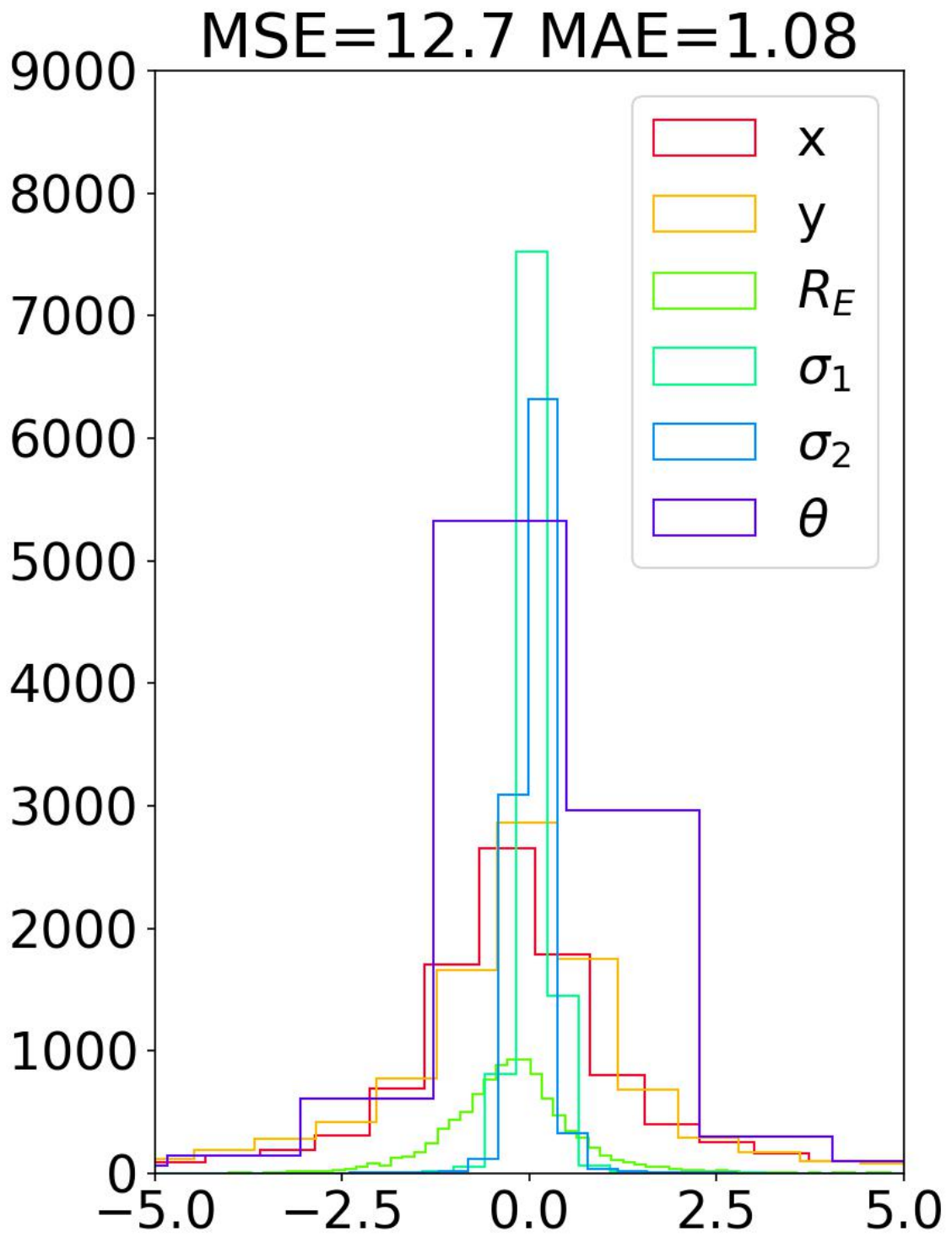


Figure C.60.: VGG-19\_BN histogram with 0.0001 learning rate.

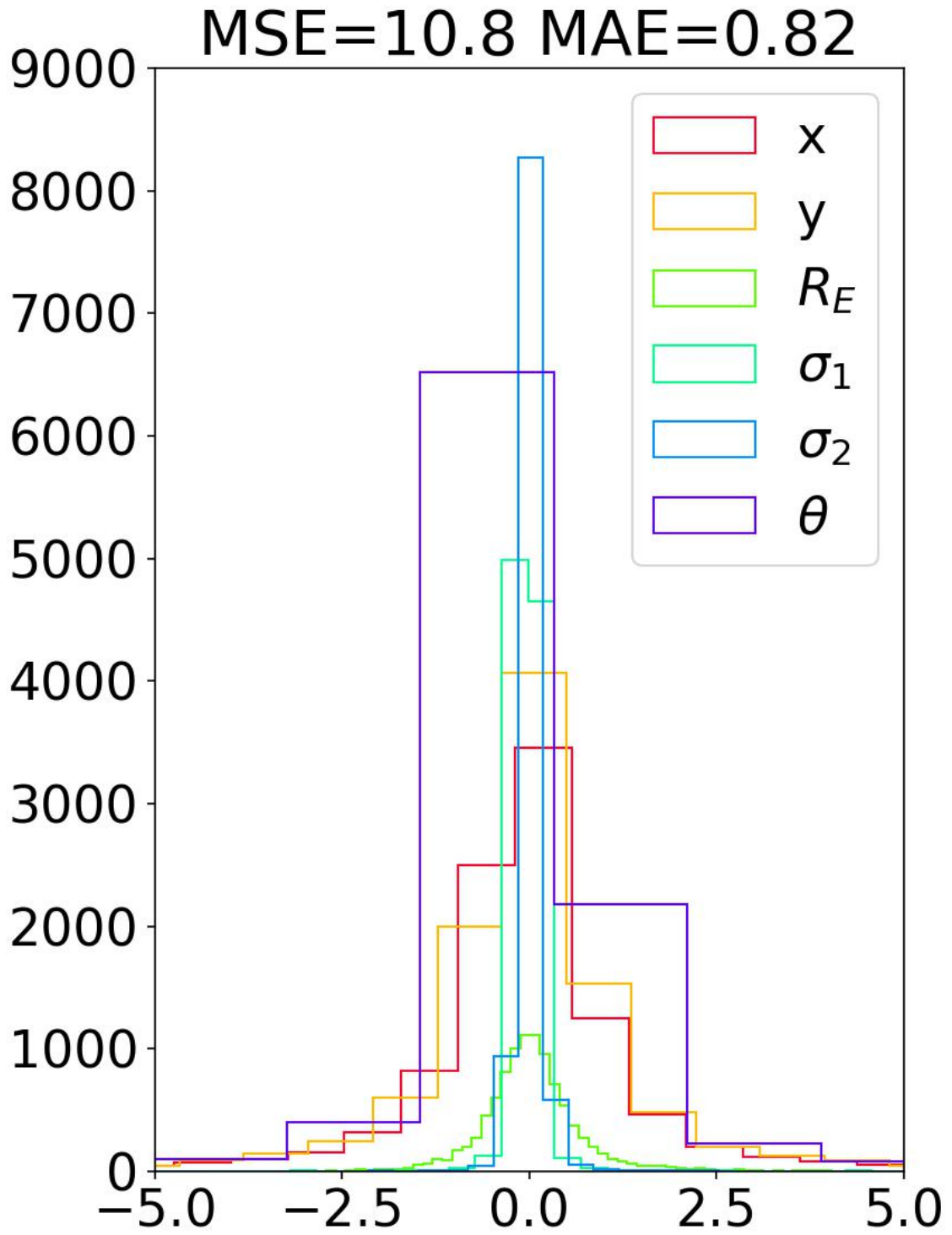


Figure C.61.: VGG-19\_BN histogram with 0.0001 learning rate for 100 epochs, and 0.00001 learning rate for next 100 epoch.

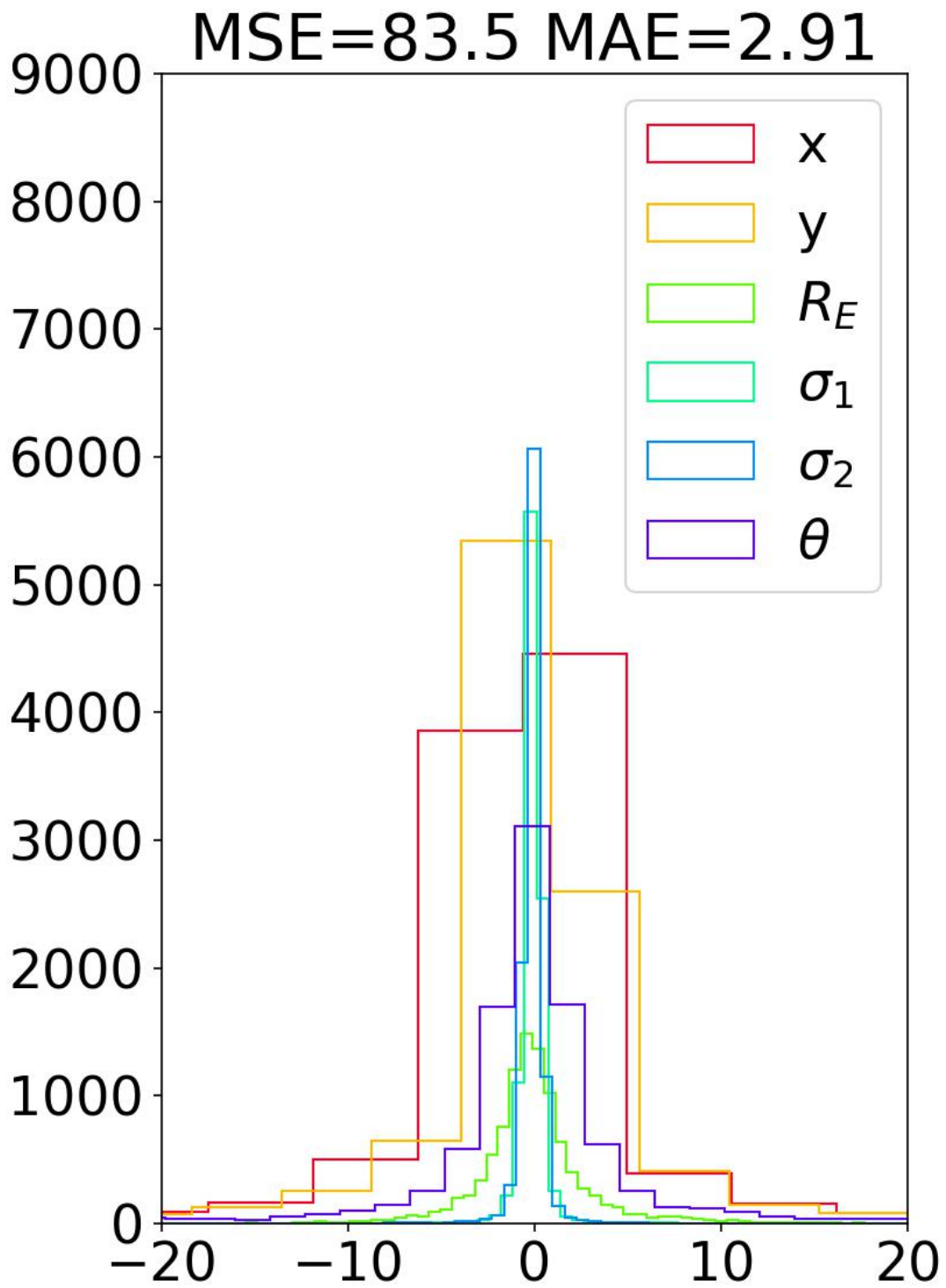


Figure C.62.: AlexNet histogram pre-trained with 0.0001 learning rate.

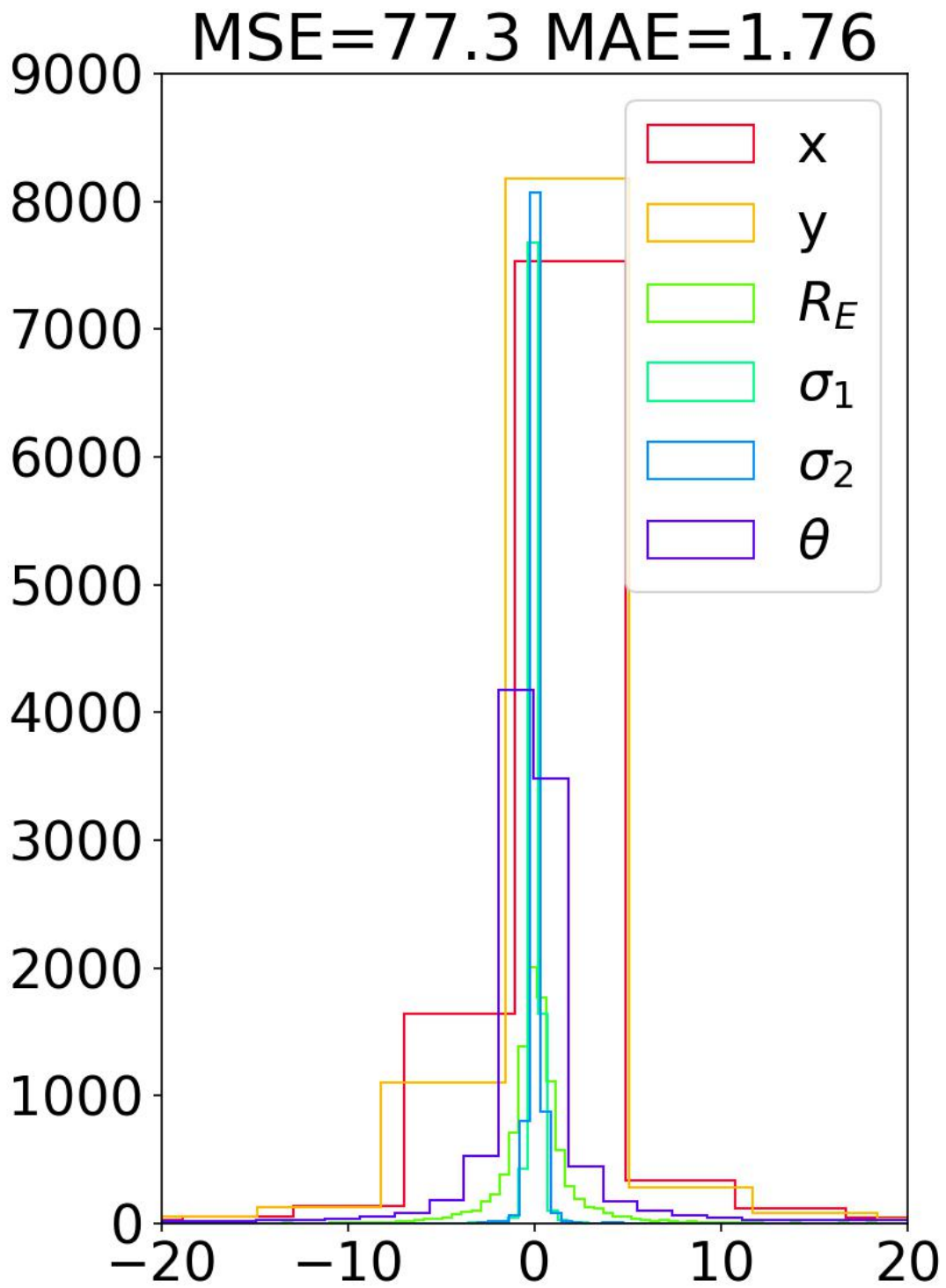


Figure C.63.: ConvNeXt histogram with 0.0001 learning rate.

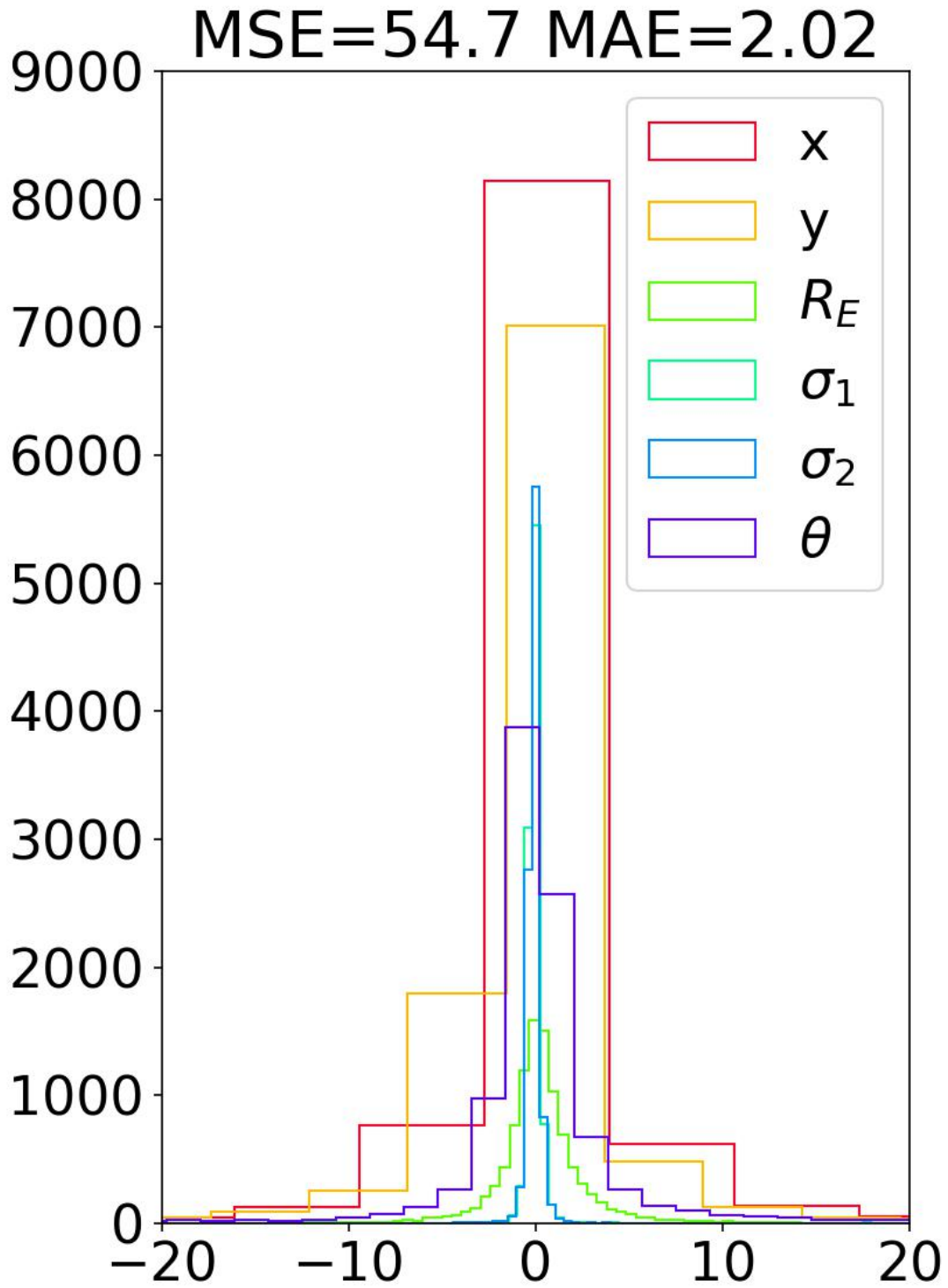


Figure C.64.: DenseNet histogram with 0.0001 learning rate.

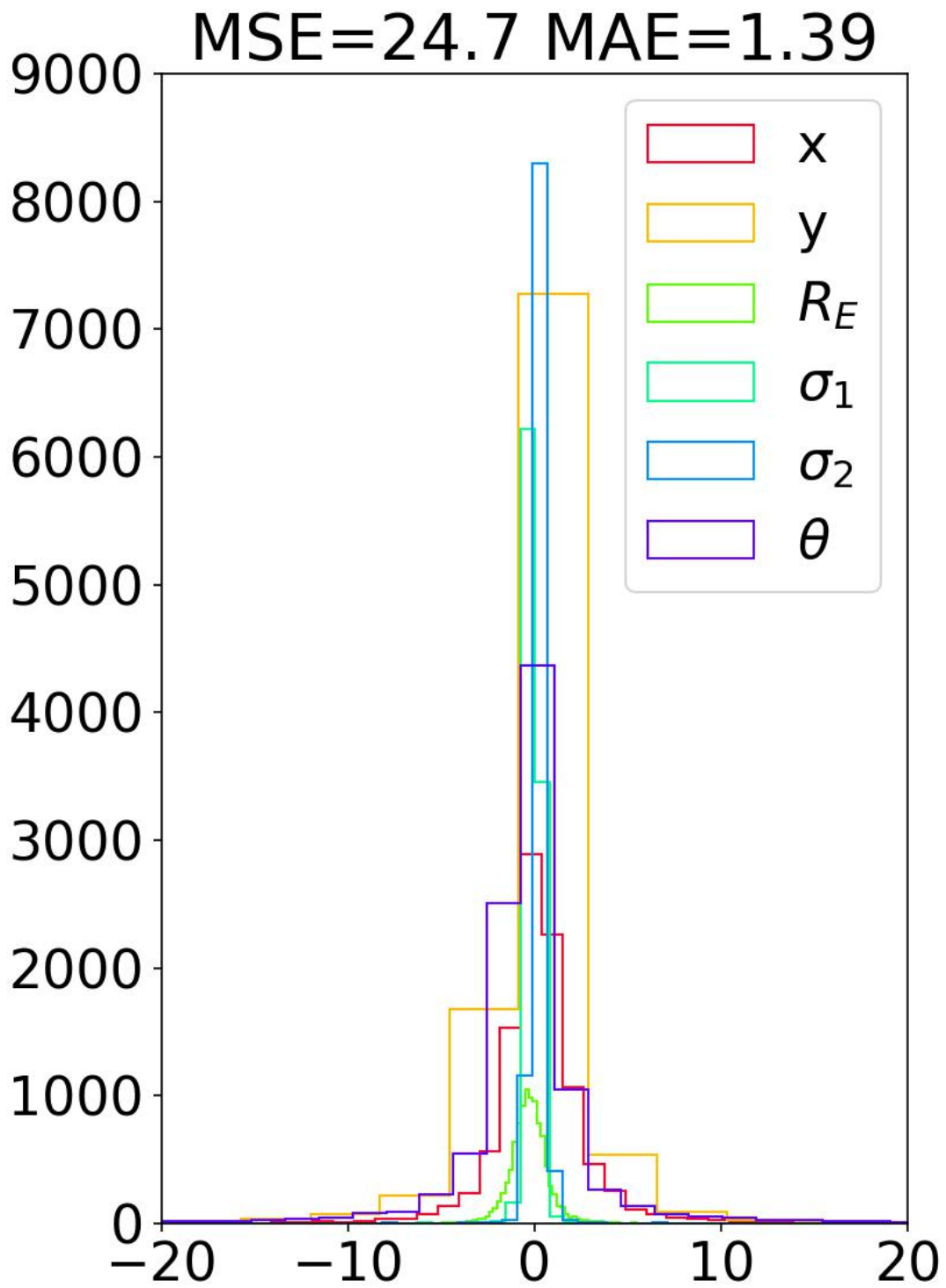


Figure C.65.: EfficientNet-B7 histogram with 0.0001 learning rate.



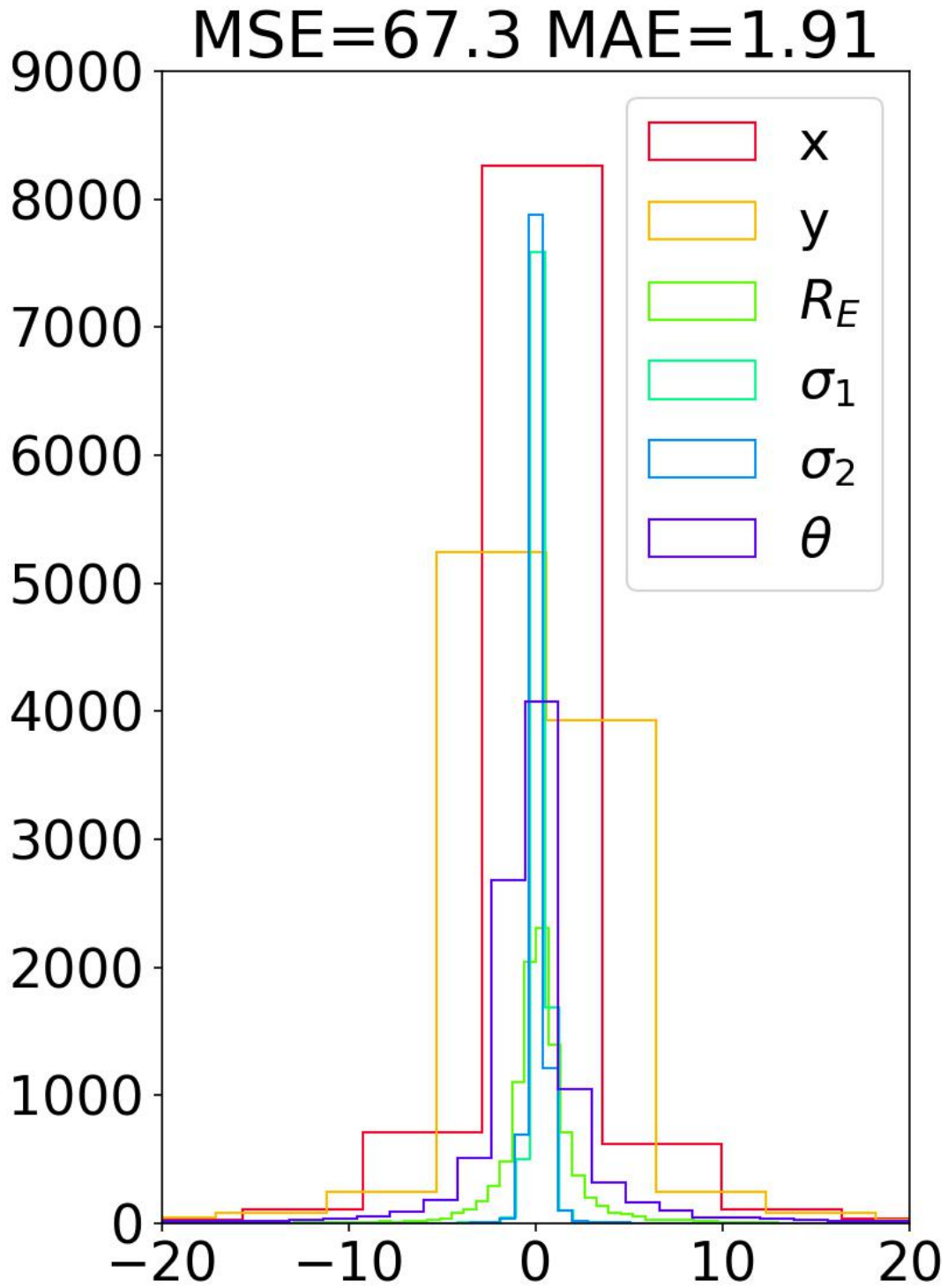


Figure C.66.: EfficientNet-v2.1 histogram with 0.0001 learning rate.

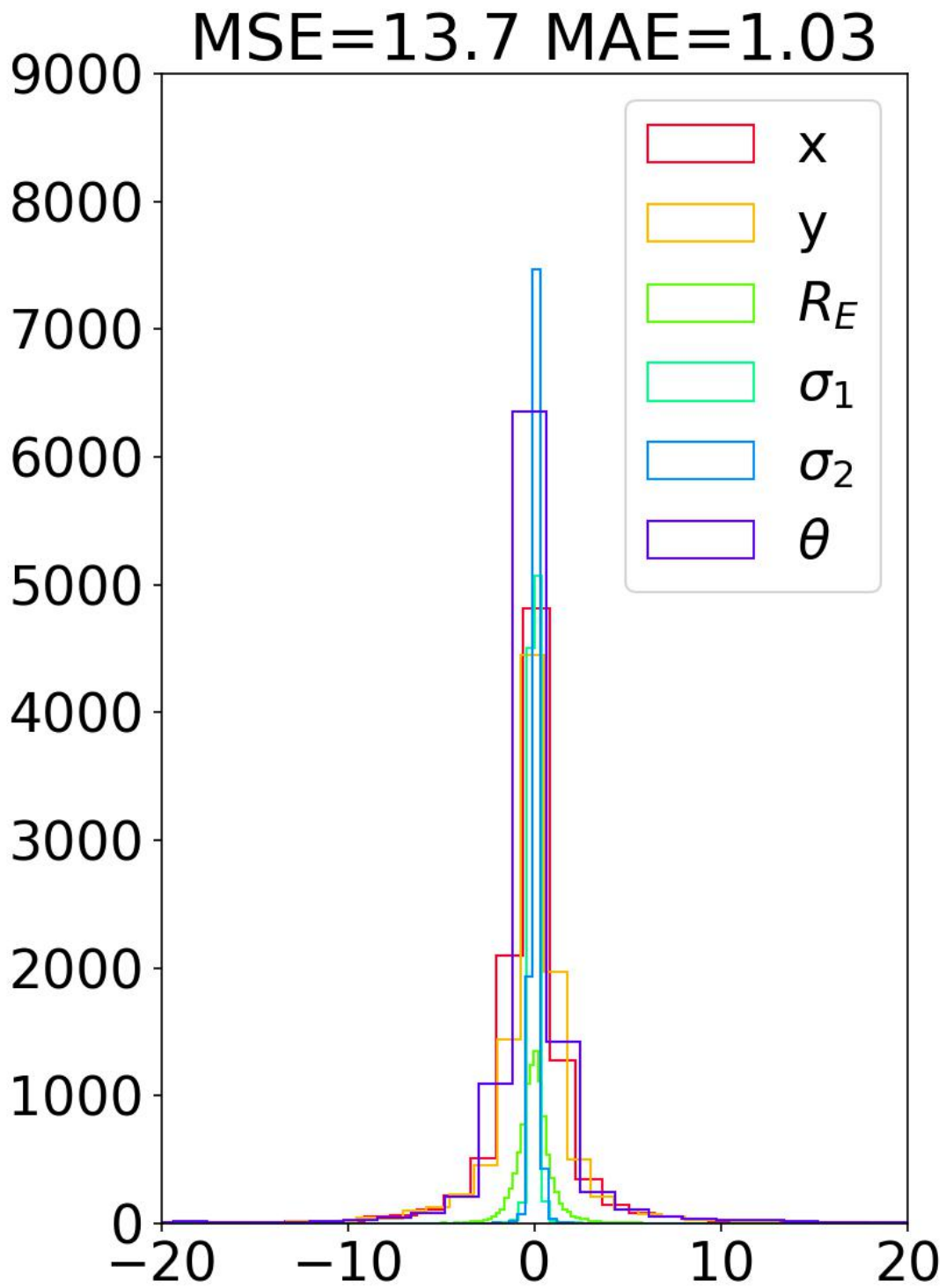


Figure C.67.: Iception-v3 histogram pre-trained with 0.0001 learning rate.

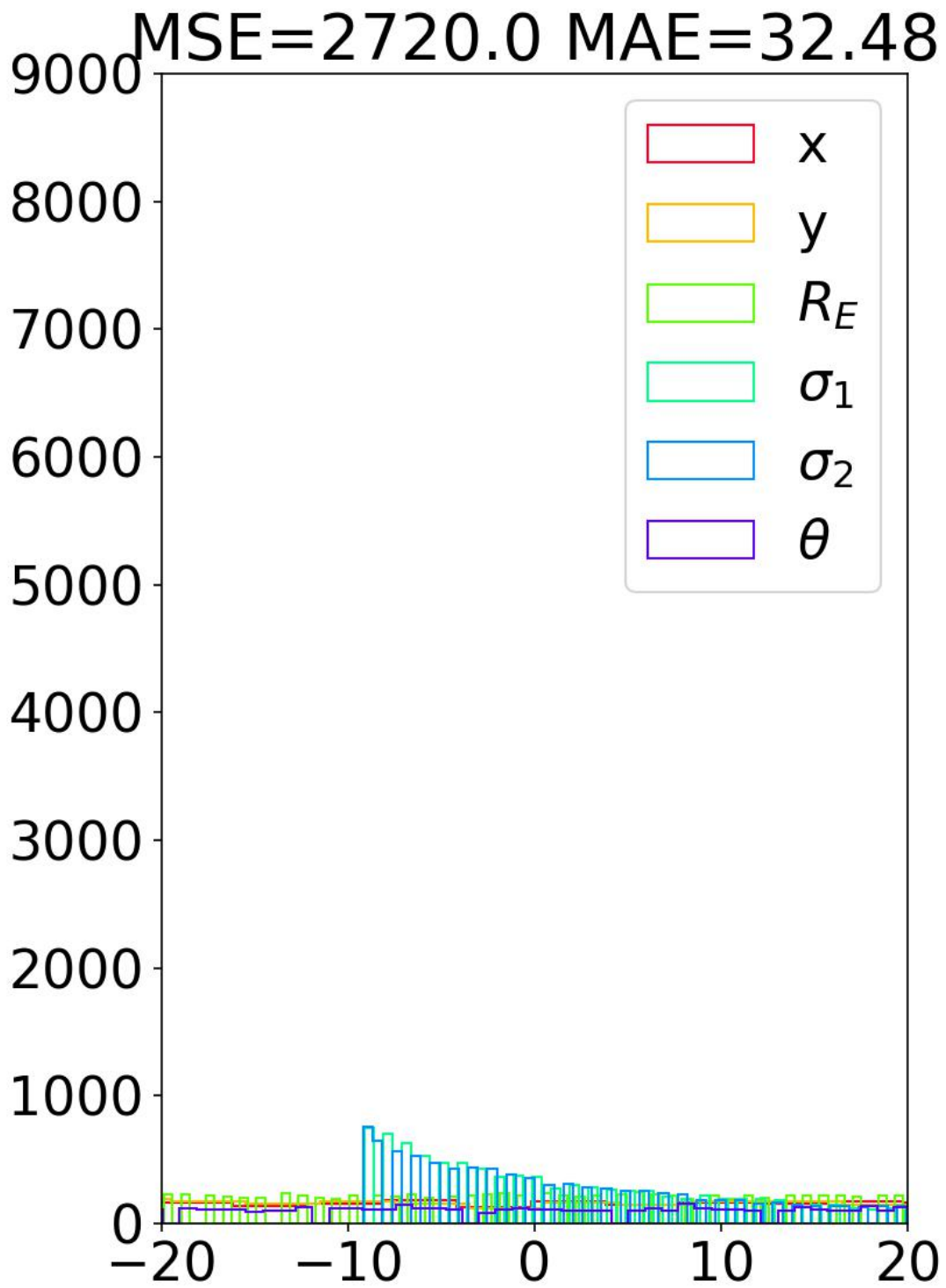


Figure C.68.: MnasNet histogram with 0.0001 learning rate.

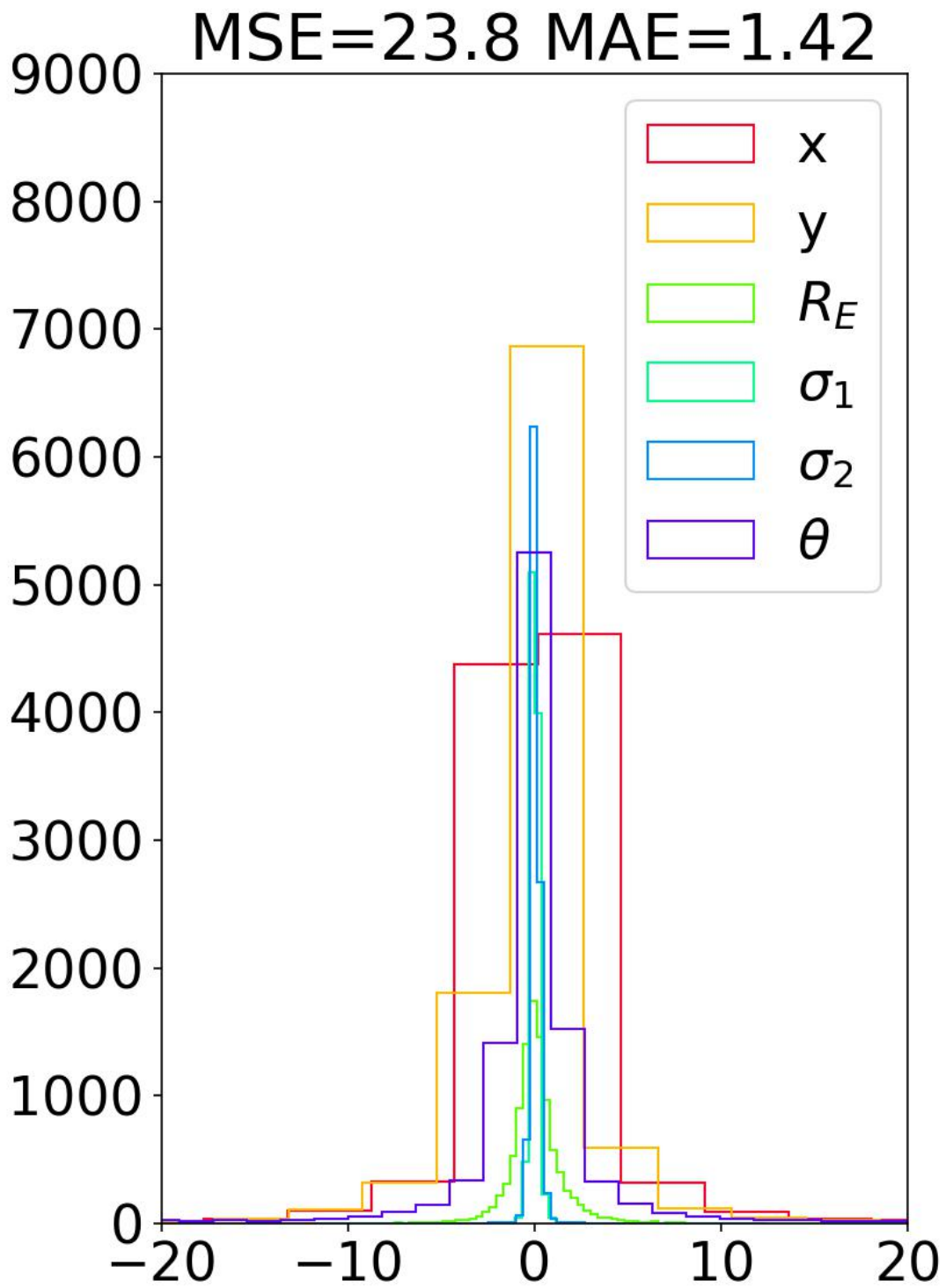


Figure C.69.: ResNet-152 histogram with 0.0001 learning rate.

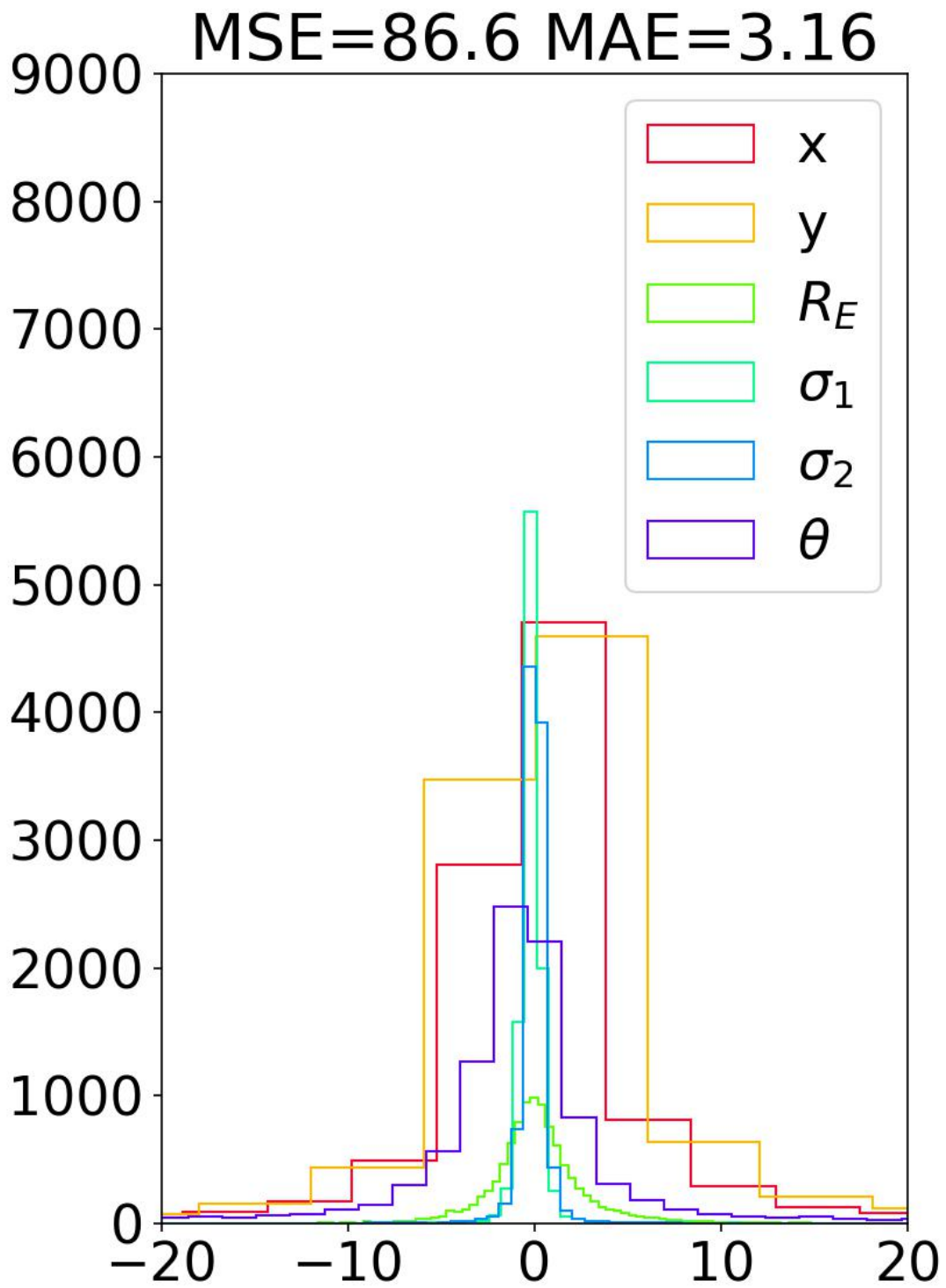


Figure C.70.: SqueezeNet histogram with 0.0001 learning rate and extra layers.

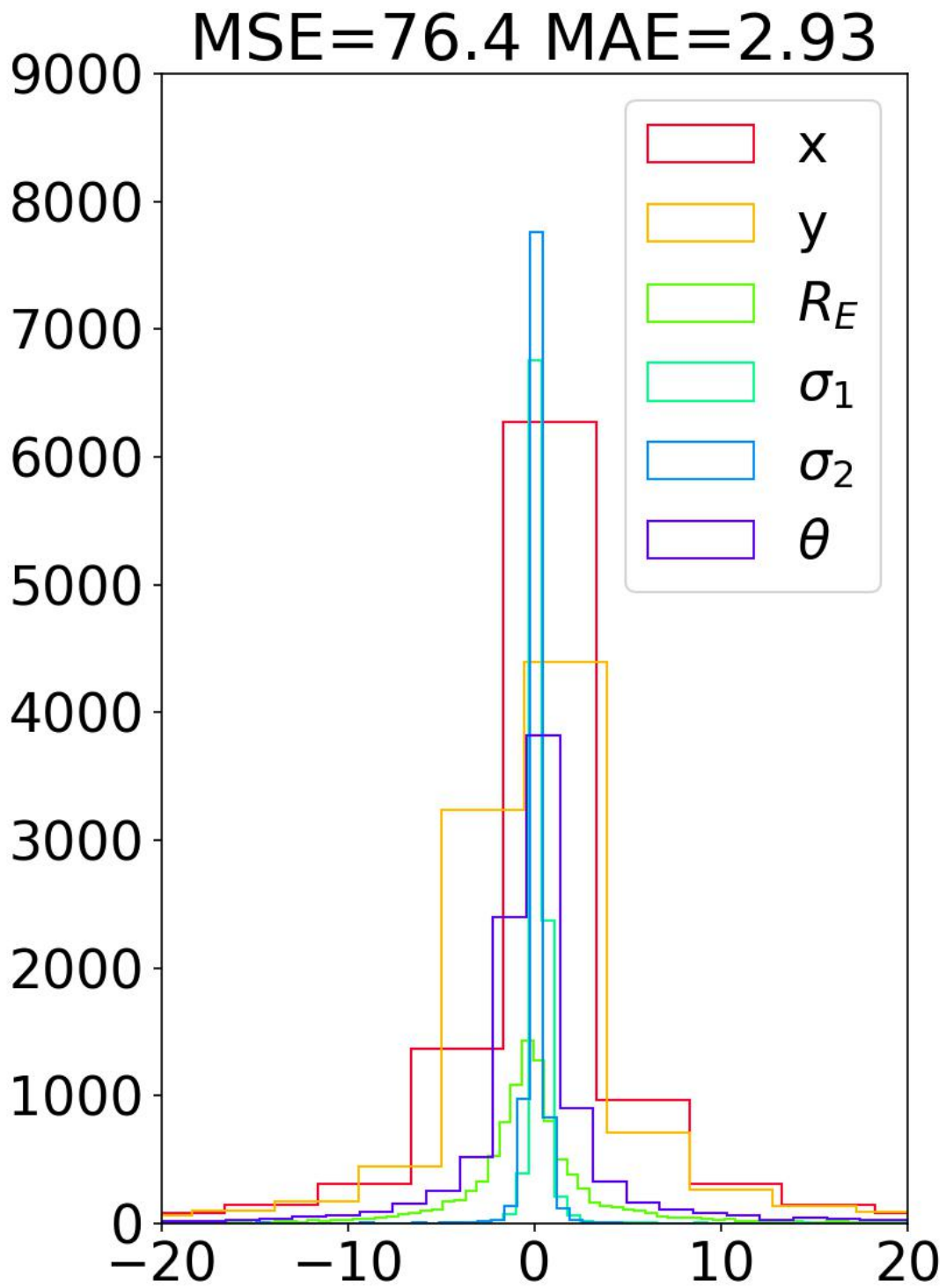


Figure C.71.: Swin histogram with 0.0001 learning rate.

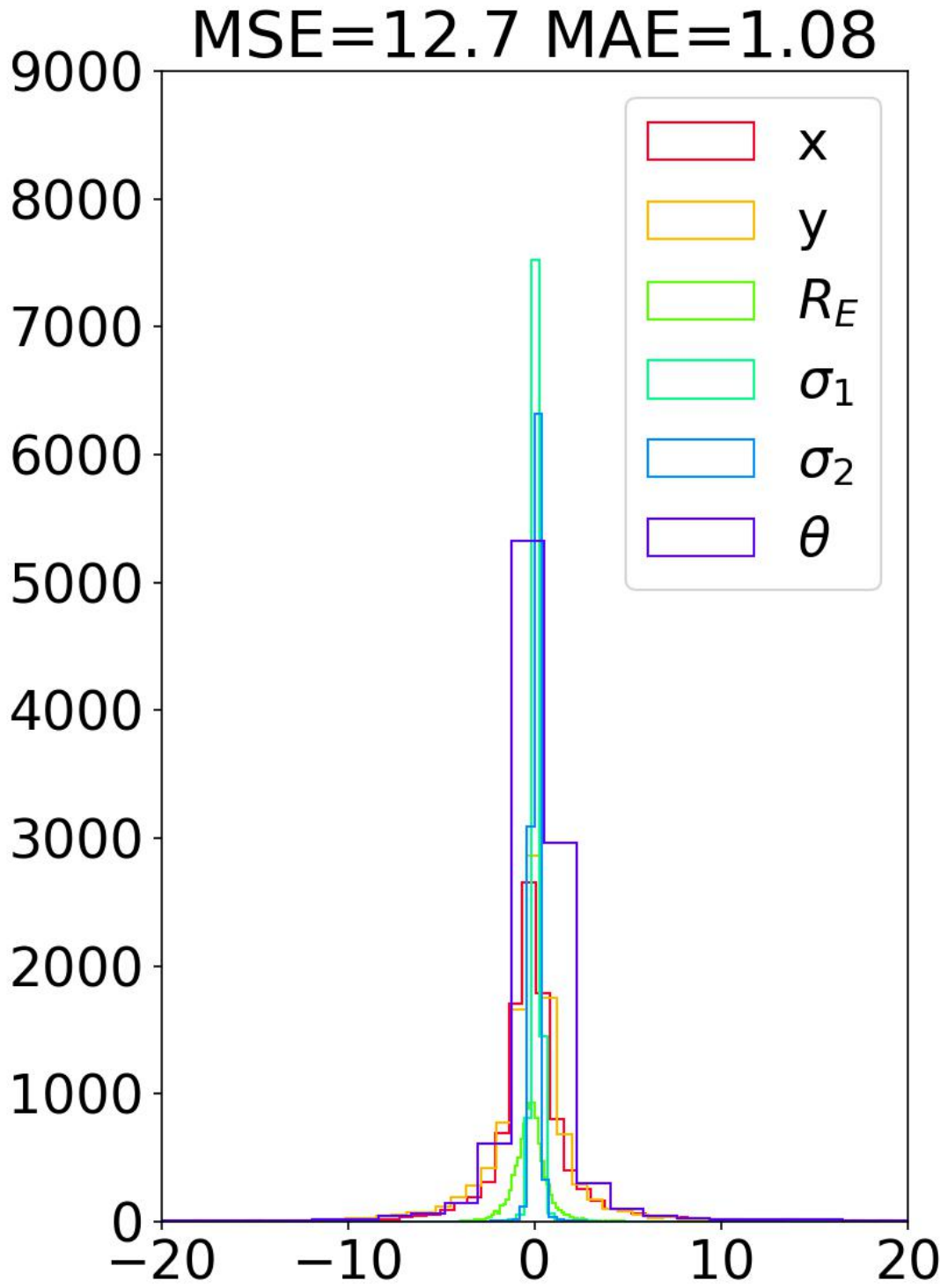


Figure C.72.: VGG-19\_BN histogram with 0.0001 learning rate.

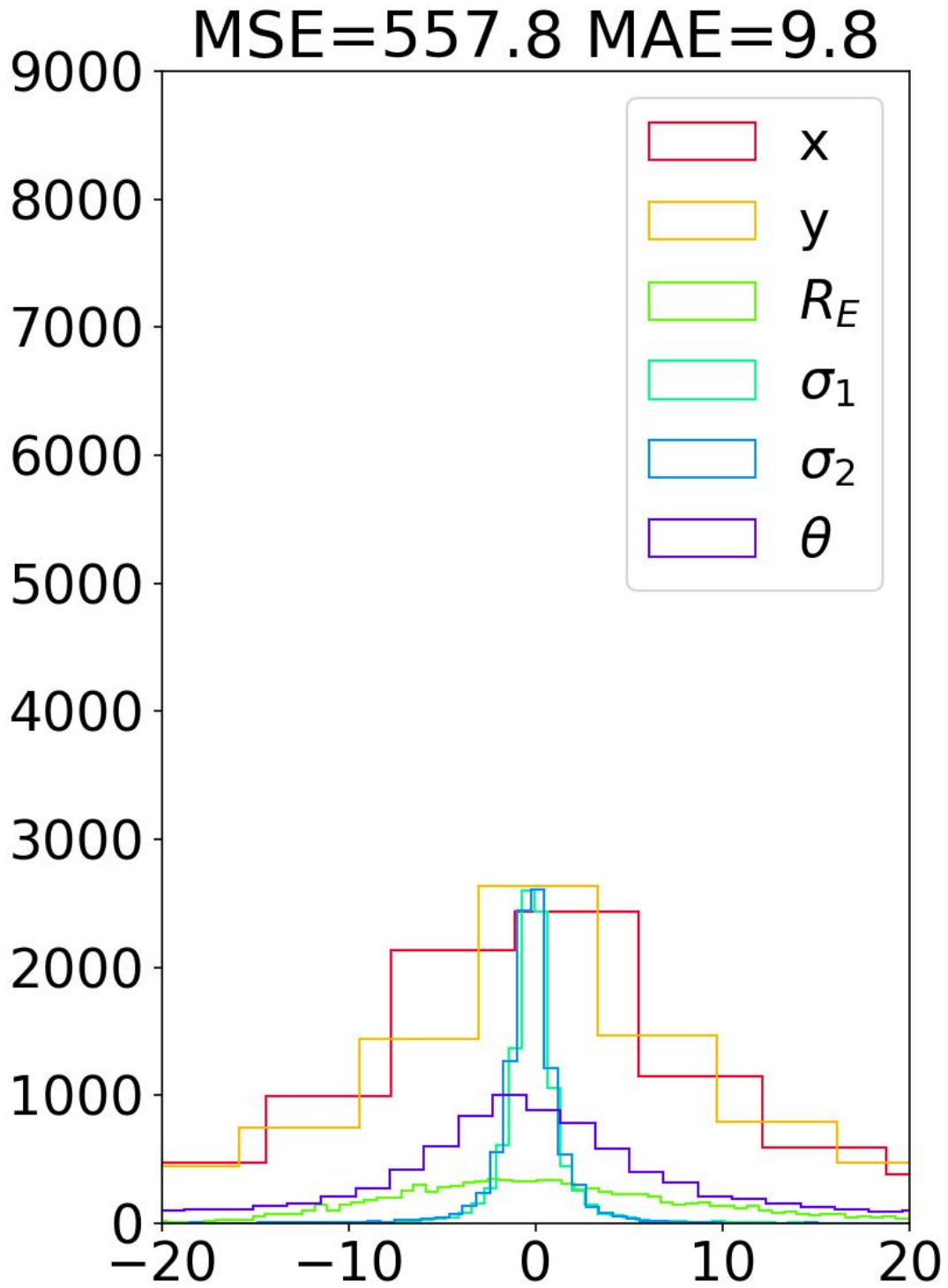


Figure C.73.: ViT histogram with 0.0001 learning rate.



## **D. Pre-project report**

This appendix contains the obligatory pre-project report.

# PRE-PROJECT REPORT

FOR BACHELOR THESIS



# NTNU

Kunnskap for en bedre verden

TITLE:

Intergalactic Machine Vision

CANDIDATE NUMBER(S):

KRISTIAN  
MODESTAS

DATE:

**12.01.2023**

COURSE CODE:

**IELEA2920**

SUBJECT:

**Bachelor thesis**

DOCUMENT ACCESS:

- Open -

FIELD OF STUDY:

**AUTOMATION AND ROBOTICS**

PAGES/  
ATTACHMENTS:

/

BIBL. NR:

- Not applicable -

CLIENT(S)/SUPERVISOR(S):

Hans Georg Schaathun  
Ben David Normann

TASK/SUMMARY:

This is a pre-project report for a planned bachelor thesis. The goal of the thesis is to continue the work on dark matter research using machine learning. Dark matter affects light through gravity, and this bending of the light can be detected.

**Postadresse**

Høgskolen i Ålesund  
N-6025 Ålesund  
Norway

**Besøksadresse**

Larsgårdsvegen 2  
**Internett**  
www.hials.no

**Telefon**

70 16 12 00

**Epostadresse**

[postmottak@hials.no](mailto:postmottak@hials.no)

**Telefax**

70 16 13 00

**Bankkonto**

7694 05 00636

**Foretaksregisteret**

NO 971 572 140

## INNHOOLD

<b>1 INNLEDNING</b> .....	<b>3</b>
<b>2 BEGREPER</b> .....	<b>3</b>
<b>3 PROSJEKTORGANISASJON</b> .....	<b>3</b>
3.1 PROSJEKTGRUPPE.....	3
3.2 STYRINGSGRUPPE (VEILEDER OG KONTAKTPERSON OPPDRAGSGIVER) .....	4
<b>4 AVTALER</b> .....	<b>4</b>
4.1 AVTALE MED OPPDRAGSGIVER .....	4
4.2 ARBEIDSTED OG RESSURSER.....	4
4.3 GRUPPENORMER – SAMARBEIDSREGLER – HOLDNINGER .....	4
<b>5 PROSJEKTBESKRIVELSE</b> .....	<b>4</b>
5.1 PROBLEMSTILLING - MÅLSETTING - HENSIKT .....	4
5.2 KRAV TIL LØSNING ELLER PROSJEKTRESULTAT – SPESIFIKASJON .....	4
5.3 PLANLAGT FRAMGANGSMÅTE(R) FOR UTVIKLINGSARBEIDET – METODE(R) .....	4
5.4 INFORMASJONSINNSAMLING – UTFØRT OG PLANLAGT .....	5
5.5 VURDERING – ANALYSE AV RISIKO .....	5
5.6 HOVEDAKTIVITETER I VIDERE ARBEID .....	5
5.7 FRAMDRIFTSPLAN – STYRING AV PROSJEKTET .....	5
5.8 BESLUTNINGER – BESLUTNINGSPROCESS .....	6
<b>6 DOKUMENTASJON</b> .....	<b>6</b>
6.1 RAPPORTER OG TEKNISKE DOKUMENTER .....	6
<b>7 PLANLAGTE MØTER OG RAPPORTER</b> .....	<b>6</b>
7.1 MØTER .....	6
7.2 PERIODISKE RAPPORTER.....	6
<b>8 PLANLAGT AVVIKSBEHANDLING</b> .....	<b>6</b>
<b>9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING</b> .....	<b>7</b>
<b>10 REFERANSER</b> .....	<b>7</b>
<b>VEDLEGG</b> .....	<b>7</b>

# 1 INTRODUCTION

The universe at large is filled with dark matter. This impossible-to-see matter is only possible to observe based on its influence of gravity. When light passes by a sufficiently dense mass, gravity causes the light to bend, altering its direction. This is called gravitational lensing. By using images of gravitationally lensed galaxies, it should be possible to un-distort the image, finding the properties of the gravitational lens. This in return allows us to map where the dark matter must be located, allowing us to create a map of the dark matter in the universe.

# 2 TERMS

- Dark matter

Matter that is invisible to us and does not emit radiation. The only solid proof of its existence is that its gravitational effect affects visible normal matter.

- Gravitational lensing

Gravitational lensing is a phenomenon where the light from a distant object, such as a galaxy, is bent by the gravity of a massive object, such as a dark matter cluster, which is closer to us. This distortion of light can be used to study dark matter.

- Machine learning

Machine learning is a type of artificial intelligence that allows computers to learn and improve from experience without being explicitly programmed. It involves feeding a computer enormous amounts of data and using algorithms to identify patterns and make predictions.

# 3 PROJECT ORGANISATION

## 3.1 Project group

Student number(s)
ID #####
ID #####

### 3.1.1 Tasks for the project group - organising.

Project leader 1 – Kristian Lynghjem Vegsund  
Project leader 2 - Modestas Kursevicius

### 3.1.2 Tasks for Project leader 1

- Area of responsibility:
  - o Project organizing
  - o Meetings
- Work assignments-:

- Supervisors dialogue
- Write meeting reports

### 3.1.3 Tasks for Project leader 2

- Area of responsibility:
  - Project plan
  - Machine learning
- Work assignment:
  - Complete documentation
  - Specialize in machine learning

### 3.1.4 Shared responsibilities

## 4 AGREEMENTS

### 4.1 *Agreements with client*

Planned meetings with clients/supervisors every 14 days, starting from 11.01.2023. Supervisors also positive to help at any time outside these meetings.

### 4.2 *Work location and resources*

- Work location:  
NTNU Ålesund
  
- Resources:  
Supervisors:  
Hans Georg Schaathun  
Ben David Normann  
Idun High Performance Computing Group
  
- Confidentiality:  
None, open-source project

### 4.3 *Group norms – rules*

All group members agree that they will:

- Show up at agreed meeting times or contact each other if it is not possible.
- Update each other on their progress.
- Log personal hours and work.
- Be objective and withhold their subjective views.

## 5 PROJECT DESCRIPTION

### 5.1 *Problem - goals - purpose*

This bachelor thesis has a goal of continuing research on machine learning as a mean to detect dark matter. Dark matter is not visible to us in any way and is only detected through its gravitational effect. When light particles travel next to dark matter, its gravity can alter the lights trajectory, akin to bending the light.

The goal of the thesis is to implement machine learning to detect gravitationally lensed galaxies and determine the properties of the dark matter. The algorithm can then be used to undistort the image, outputting an un-distorted image of how the galaxy looks. The goal is to be able to use real images of galaxies as input, to help map dark matter in the universe.

## 5.2 Requirements results – specification

- Machine learning:
  - o Be able to train on generated images.
  - o Be able to recognize and mark gravitational lensing in a picture.
  - o Be able to input a distorted image and return an undistorted one.

## 5.3 Planned course of action – methods

The plan is to implement *agile software methodology*. Everything planned is put in a backlog. Short sprints are then done (usually 1-3 weeks long) to finish certain decided problems. After the sprints, new tasks are chosen, and a new sprint period begins.

## 5.4 Information collection

The group has already a good collection of literature on the topics:

- o Roulette formalism
- o Gravitational lensing
- o Machine learning

Additional resources will be collected if and when needed.

## 5.5 Risk assessment analysis

Consequence/ Probability	Low	Medium	High	Severe
Very unlikely	Green	Green	Green	Yellow
Unlikely	Green	Green	Yellow	Yellow
Likely	Green	Yellow	Yellow	Red
Very likely	Green	Yellow	Red	Dark Red

Figure 5.1 – Risk matrix

### 5.5.1 Lockdown

Probability: Unlikely

Consequence: High – Would force remote work

Yellow (Medium risk)

Measures: Save everything in cloud-based storage. Make everything accessible to all members regardless of physical location. Members are aware meetings and work could be moved to fully digital.

### 5.5.2 Sickness (one or both members)

Probability: Very likely

Consequence: Medium – Would force remote work, but for a limited time

Yellow (Medium risk)

Measures: Save everything in cloud-based storage. Make everything accessible to all members regardless of physical location. Members are aware meetings and work could be moved to fully digital.

### 5.5.3 Hardware problems

Probability: Very unlikely

Consequence: Medium – Would lose access to laptop or Idun cluster. Would lose all local files.

Green (Low risk)

Measures: Idun cluster going down is likely to be temporary. Laptop problems can be circumnavigated by using another computer and be sure everything important is always in cloud storage.

## 5.6 Main activities going forward

Nr.	Main Activity	Responsibility	Cost	Time/scope
A1	Pre-project report	KV/MS	--	5 days
A11	Distributing responsibilities	KV/MS	--	1 hour
A12	Work agreement	KV/MS	--	1 hour
A13	Writing pre-project report	KV/MS	--	3 days
A14	Plan meetings	KV/MS	--	1 hour
A15	Supervisor meetings	KV/MS	--	3 hours
B1	Machine learning	MS/KV	--	3 months
B11	Research of machine learning libraries for python	MS/KV	--	1 month
B12	Research of new machine learning structures	MS/KV	--	1 month
B13	Making a neural network capable of classifying gravitational lenses in a photo	MS/KV	--	1 month
C1	Simulator	MS/KV	--	2 months
C11	Creating a representative database to real data	MS/KV	--	1 month
C12	Try to make simulator with colours	MS/KV	--	1 month
D1	Report	1/2		Finish by 22nd May 2023

## ***5.7 Progress plan – planning***

### **5.7.1 Main plan**

Since this project is open ended it is hard to have any certainty on our project. First, we will make simulator work on our computers. Then do research in what parameters gravitational lenses are found out in real world. Next try different machine learning structures to fit our purpose better.

Group plans on planning only 1 to 3 weeks ahead. No more since the task is loosely defined. Therefore, the list in 5.6 is only to be taken as a temporary draft.

Milestones:

- Settle on realistic parameter ranges for simulator.
- Find machine learning structure that is more efficient.
- Make a machine learning model that can find gravitational lenses in a picture.
- Add colours to the simulator and simulate redshift.

### **5.7.2 Management aids**

- Report from last year's work.
- Confluence
- Pre-project report
- GitHub
- Library

### **5.7.3 Development aids**

- Physical:
  - Personal computers
  - IDUN high performance computing for training of neural networks
- Programs:
  - GitHub for version control
  - PyCharm for Python programming
  - CLion for C++ programming
  - Qt Creator for GUI programming

### **5.7.4 Intern control – evaluation**

Project will be work on collaborative, working at the same location with supervisor meeting every 2 weeks.

## ***5.8 Decisions – decision making process***

Since it is a two-person group working side by side, discussions will be used to make decisions until both parties are satisfied.

## **6 DOCUMENTATION**

### ***6.1 Reports and technical documents***

- Pre-project report
- Meeting notes
- Bachelors tesis



## **7 PLANNED MEETINGS AND REPORTS**

### ***7.1 Meetings***

#### **7.1.1 Meetings with supervisors**

- Every 14 days since 11.01.2023
- Meetings will be used for discussing work that has been done and needs to be done in future.

#### **7.1.2 Group meetings**

- Group will meet every Wednesday, Thursday, Friday to work on and discuss the project.

### ***7.2 Periodic reports***

#### **7.2.1 Progress reports - milestones**

- Progress reports will be produced every two hundred hours of combined work.
- Meeting notes will be produced after every meeting with supervisors.

## **8 DEVIATION PLAN**

- Responsibility for progress of this project is on both members.
- Deviation is expected but both members need to agree on how that deviation will be done
- If any conflicts occur, we will contact our supervisors on their opinion and move on from that.

## **9 EQUIPMENT/PREREQUISITES FOR COMPLETION**

- IDUN High Performance Computing Group

# **E. Supervisor meeting notes**

# Meeting notes bachelor project

This document contains the notes taken from each supervisor meeting throughout the bachelor project, as well as any relevant pictures taken from whiteboards/blackboards.

Most of it is written in Norwegian, and is written pretty hastily as to not stop the flow of the meetings. Typos are expected, but the meaning of everything should be clear.

## **Møte 11.01.23 Referat**

- - Skriv innledning tidlig for å begrense/beskrive oppgaven
- Ikke bruk forrige rapport som mal, de var for mange med for stort skop
- Åpen oppgave / opp til oss hva vi vil gjøre
- Simulatoren er under utvikling, dokumentasjon blir for utdatert/unyttig, men skriv gjerne kapitel for hvordan komme i

Plan til neste møte/2 uker:

Skrive forrapport

Skrive ned tanker

Begynne på evt innledning til rapport for tilbakemelding

Skriv lite literatursammendrag fra det vi leser (merk relevante ting)

Neste møte: 25.01.23

## Møte 25.01.23 Referat

- Hans Georg gikk gjennom forprosjektrapport for å forstå fordelingen.
  - Forprosjektrapport var ok, ingen endring nødvendig
  - Tcltk
  - Qt er antageligvis den beste for GUI, men ikke for python (opencv)
  - Bedøm på veiledningsmøter fra backlog hva vi skal gjøre til neste gang
  - Prøv å ha ny prototype klar til neste møte hver gang
  - Felles forventning på hvert møte om hvor vi er om 14 dager (eller annet intervall)
- 
- Ben david notes: teori, historisk, helt I starten står det prosjekt overview (figur 1.1), fysikksiden er på SIE.
  - Resten av notaten: kapitel 3 forklarer roulette-modellen. Den forsto vi ikke (matematisk).
  - Kapittel 4: Spesifikke utregninge (trenger ikke forstå alt). Hente ut det vi trenger?
  - Vi kan gjennomføre prosjekt basert på simulator allerede lagd. Vi trenger en implementasjon av BDN modeller. Hvis vi vil debugge simulatoren eller videreutvikle eller validere, da må vi forstå BDN notat. Eller implementere elliptiske modeller.
  - Fortell Hans Georg om 4 uker om vi vil gjøre selv eller dytte den på HGS og BDN fordi vi skal fokusere på maskinlæringen.
  - Kapittel 5: Litt utdatert skrevet av HGS. Teknisk dokumentasjon av simulator.
  - Vi kan bruke CosmoSIM som sort boks. Burde det I starten iallefall.
  - Vi kan bruke det lette I notatet I introduksjonen I hovedoppgaven.
  - Les iallefall til og med 3.2 om ikke lenger.
  - Gjennomgang av figur 3.2 på tavle
  - PM modell finner  $x, y, R_E, \sigma, \chi$
  - Det er implementert betingelser for hva parametre kan være ( $R > R_E$  bl.a.). Ferdig for PM model, ikke for SIS/SIE. Hvor spredt kan massen være før det blir en vesentlig forskjell fra PM model. Med en klynge kan det samla massen være stor. En punkt-sky model burde kunne være realistisk nok. Hvis den er lett å regne på (og finnes) kan den være nyttig.
  - Kapittel 4.1.2: Utvidelse fra punktlinse til andre burde være enkelt.
  - Focus more on Neural network that is more accurate rather than efficient

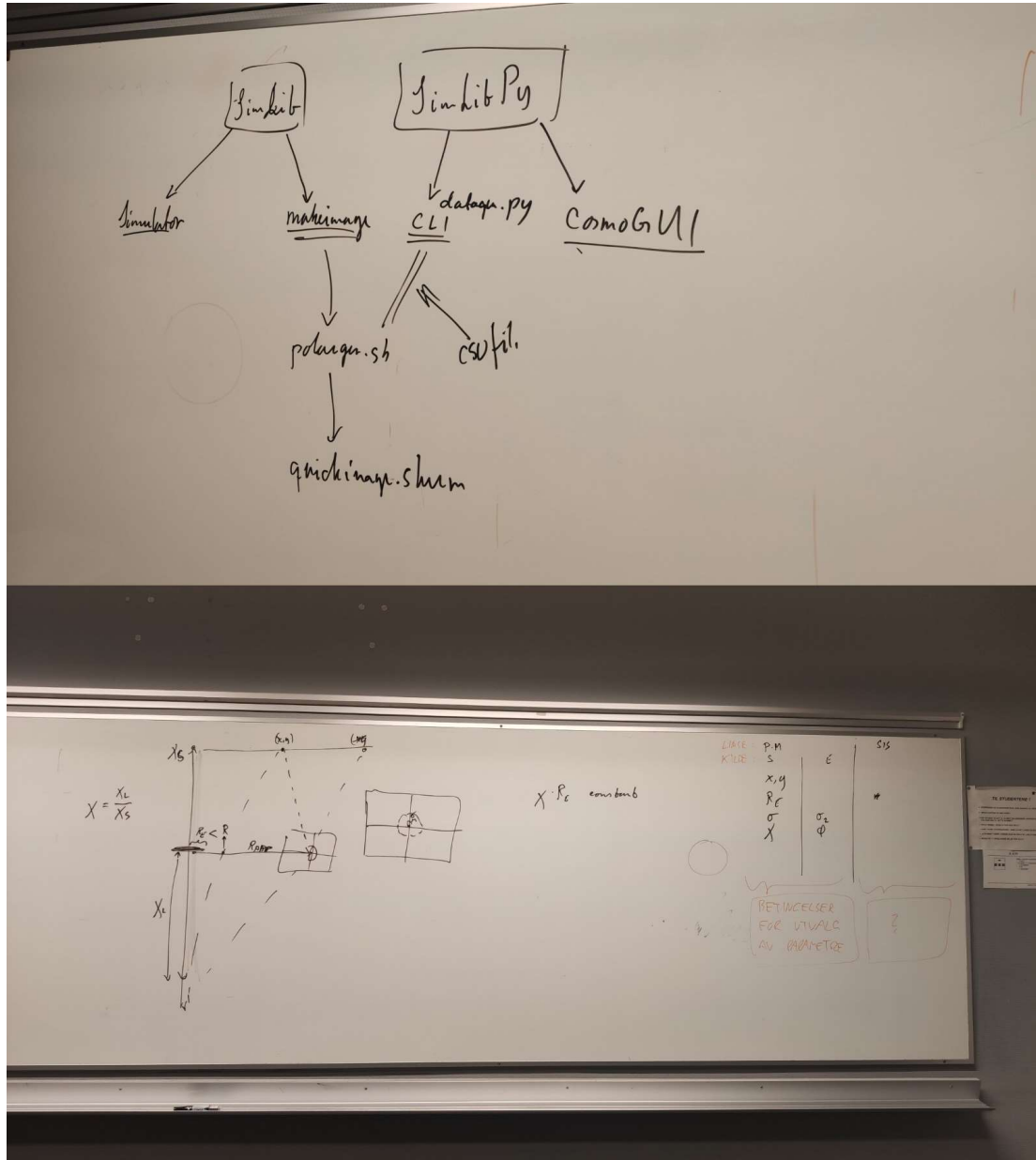
Ben David undervisning/opptatt:

- Mandag: 12-14
- Tirsdag: 8-10 og 16-18
- Onsdag: EiT hele dagen (men kan bookes)
- Torsdag: Fri hele dagen (iallefall til masterstudent kommer)
- Fredag: Permisjon hele dagen
- På campus 7-17 man-fre

Til neste møte:

- Lage en backlog
- Få ting til å kjøre

Bilde:



## Møte 08.02.23

### Spørsmål:

- Hva er forventet av oss på maskinlæringsfronten? Er det nok med at vi forbedrer den/lager vår egen?
- Svar: Hvis vi skal lage en god avhandling, skal vi forkuserer på evaluering av hva vi gjøre (hva er bra her). Hva er gjort nesten ingenting? Studentene I fjor fikk now til å virke. De dokumenterte ikke hvorfor de valgte hva de gjode. Vi må finne ut om det er den beste eller om der er bdre alternatic. De fjorde en feil; De genererte bildet, de visste hvor midten av bildet er. Du får veldig mye informasjon av hvor den ligger I forhold til senter av bildet. I realiteten vet du ikke det. Informasjonen om dette burde vært tatt ut av bildet, og sentrering burde gjøres. Centerimage.py regner gj snittlig av alle pixler og vekter med lysintensiteten. Sort = 0 vekt, Hvit = 1 vekt. Den beregner og translaterer slik at den senterer bildet.

Hvilket nettverk er best? Begrunne Begrunne Begrunne

Sentrerte bilde: Hvilke parameter kan vi estimere.

- Hvordan kan vi bevise at bilde med same forhold mellom E\_R og Chi er same?
- Svar: Hvis vi kan konstruere dataset som beviser vi har pratisk talt like bilder, har vi godt bevis. God strategi å statistisk vurdere datasettet for å bevise det.
  - Hivlke parameter er ubestemmelig?
- Vi for artifakter I bilde generering vårt. Hva det betyr? Har dere vært bort i det?
- Svar: HGS rettet feilen sist uke, men husker ikke hva som var feilen. I simulatoren hadde noen glemt å initialisere midlertidig bilde med 0. Når den overskriden bilde regner den ut akkurat den regionen rundt inni roulettesirkelen. Gammel bildedata lå under ny.
- 
- Svar:

Hans Georg tips om hva vi skal gjøre først:

- Sentrerte bilder: Hvilke parametre kan vi estimere?
- Er noen parametre ubestemmelige fra datasettet? (ML eller statistikk)
  - Regn ut bildedifferanse (absoluttverdi / kvadrert)
  -

### Notater:

- Last ned develop-brach, ikke master
- Cosmosimpy lager python-ting i c++. Den lager en klassefasade for å sette alle parameter I modellen og velge hvilken du bruker. Slik bruker python mindre kraft.
- Cosmosim init.py importerer cosmosimpy
- Hvis c++ kode virker, generer den cosmosimpy
- Init tråder (multithread) for at GUI ikke skal henge seg opp I real time sim.
-

- 
- 
- Å kunne argumentere for at programmet vårt er unikt eller så bra som mulig er bra.
- 
- Verdt å lese og finne ut hva forfattere hadde i tankene i papers på NN (hva var de laget for?)
- Målet er ikke å ha nyeste NN/ML, men heller ha noen som er godt beskrevet, så kan vi gjøre en vurdering for hva våre problem har til felles til de nettverkene og sammenligne. Prøve forstå så mye som mulig, teste hypoteser.
- Grave i empiriske data?
- Burde ha med en nettverksdefinisjon(?). Hvis vi finner ett nettverk (ferdig eller kodet selv), ha med referanser så leser kan slå opp definisjon,. Hvis vi lager selv burde hele dokumentasjon følge med. Legg med python kode i vedlegg + grov beskrivelse i teksten.
- Andre måten er å lage en figur som beskriver lagene. Det er mye arbeid å få til, men ser bedre ut.
- Definisjon av nettverket må være med.

Vurderingsgrunnlag: Hvor mye nytt ligger i teksten? Hvor godt presenterer teksten denne? Bedre å ikke skrive for mye/overfladisk. Vi burde hatt litt råd fra Ottar. Alltid en risiko å avvike fra tradisjon. Hvis sensor forventer samme som alle andre, er det risiko for å bli trekket for å skrive kort men bra. Hvis vi kan argumentere bra, trenger den ikke være lang. Ikke drukne resultat i tekst. Innledningskapitel veldig mange som skriver samme som året før. Det irriterer ottar. Mange skriver om hvert bibliotek/library. Unødvendig. Skriv heller om hvorfor du velger de om det sto mellom flere. Ikke skriv for mye om slike ting. Metodekapitel (material and methods): Hvordan henter man data og gjøre statistisk analyse om den. Vi har ikke så mye å si på den. Det blir ofte langt for andre, men ikke nødvendigvis for oss. Ottar er ingeniør. Hvis sensor er usikker på kriterier spør de Ottar. Kan få tilbakemelding fra medstudenter på om det er leselig. Bytt gjerne rapporter med en annen gruppe for å gi gjensidig tilbakemelding. Tydelig med referansene for å gjøre det mer leselig. Avslutt gjerne med åpne spørsmål?



## Møte 22.02.23

### Spørsmål & svar:

- Vi sammenlignet bilder med samme  $\chi/E_R$  forhold, og ca 40-50% av bildene er identiske, mens de resterende er under 1 pixel forskyvet til siden. Er dette nok for ML? Er dette nok til å si at de er samme bildene?
  - For 8-bit bilde: 255-pixel span – For å tolke: Se på hvilken avstand vi får når vi regner avstand mellom bilder. To bilde, det nærmest linsen det minste. Hvis kilden er i origo, så vil
  - Dem slyber it fra s,a,e akse fra origo, når bildet er i origo skjer ikke dette. Da vil den dra bildet ut langs x akse. Hvis bildet ikke er i origo vil det antageligvis ikke skje.
  - - Sjekk om forskjellen er større med lave og høye verdier av  $\chi$  (nært 0 og 100). Hans georg tipper det er numerisk feil. Modellen er ikke nøyaktig hvis vi får små tall (f eks  $\chi = 1$ ) å gå høy avrundingsfeil.
  - Opencv bruker brøkdeler av pixler, de interpolerer, men avrundingsfeil kan føre til at de hopper over en pixel.
  - Vi kan regne euklids avstand mellom 2 bilder og kvantifisere avstanden. Da kan vi med 8bit bilde vise at største lysfeil er 1 enhet. At ingen pixler avviker med mer enn 1 eller 2 i lysstyrke. Kan også sjekke hvor ofte store feil oppstår.
  - Flat sky har en tilnærming, så avrundingsfeil skjer. Det er en avrundingsfeil fra flat-sky approximation. Vi kan ikke unngå avrundingsfeil.
  - Hva skjer i lavere oppløsning? Blir det større forskjell?
  - Kan bruke redshift for å finne avstand for å finne  $\chi$
- Test med  $\chi$  konstant (50 f eks) og kjør ML med det for å sjekke om det blir bra.

# Møte 08.03.23

## Notater:

- Vi må holde mye konstant, og variere få parameter/element
  - Hvis vi varierer alt vet vi ikke hvorfor, så test få nye parameter om gangen
- Velg referansepunkt
  - Finne noe som har ok nøyaktighet
    - Antageligvis AlexNet
    - Kan spisse søket til å forbedre etter utgangspunkt er greit
- Kan være interessant å se hvorfor efficientnet er tregt når det skal være raskt, kanskje tyngre referansepunkt
  
- **AlexNet referanse:**
  - Velge optimaliseringsalgoritme
  - Velge kostfunksjon
  - Hvor mange epoker?
  - Kjøretid? (inference og training)
  - Nøyaktighet (feil per parameter, gj snitt og varians)
  - DOKUMENTER ALT DOKUMENTER ALT
  - DOKUMENTER ALT DOKUMENTER ALT
  - DOKUMENTER ALT DOKUMENTER ALT
- Når vi har referanse, kan vi vurdere hvor mange ting vi har mulighet til å kjøre resten av prosjektet. Dokumenter dette og begrunn i rapporten.
  
- 300k bilder for å trene er veldig mye, fordi vi har få parameter så bildene har ikke mye variasjon.
- Bildene må være sentrert
  - Linsen må flyttes til midten av det forvrengte bildet (se tavlebildet)
    - Dette er antageligvis gjort (ser slik ut)
    - Må være gjort for trening på empirisk data
- Prøv med 300k bilder eller mindre for å se om det gjør en forskjell hvis det ikke tar for lang tid
  
- **Adversarial training:**
  - Begynne med tilfeldige sett
    - Ta bilder som gir dårlige resultat for å lage nytt sett med kun de
      - Tren på nytt med bildene som gir dårlig resultat
- Kan sammenligne estimat -> simulator -> fasit med input (se tavlebilde)
  - Kan trene noen epoker til med sett av de som ga dårlig resultat etter vanlig trening.
- Sett tilbake i treningsmodus etter test automatisk.

- 300k bilder i 50 epoker -> ett par epoker med de få dårlige bildene vil antageligvis gi lite resultat
- Adversarial training må sjekkes ut videre, veldig relevant og **nytt**
- Hva gjør du så? Mate tilbake adversarial training eller bruke fasiten til å forbedre estimat? Åpent.
  
- Masterstudent skal ikke begynne før august
  
- Må ha en del nokså ferdigstilte sider før påske for tilbakemeldinger. Det tar litt tid og tar noen runder fram og tilbake.
  
- Ikke modifier nettverk før vi har funnet ut hva som er bra og ikke og hvorfor. Da kan vi bruke tid på slikt.
  
- Anbefalt å gjøre ting med SIS også, ikke bare punktmasse.
  - Kan lønne seg fordi det gir oss flere sjanser til å finne noe interessant
    - Vi kan ikke bare se på punktlinse og se på hva som er best, fordi hvis det ikke er noe spennende der står vi fast
    - Hvis vi har SIS også, er det flere ubesvarte spørsmål som gir mer interessante problemer og løsninger:
      - Hvorfor er SIS vanskeligere enn punktlinse? Etc. etc. etc.
  
- Det viktige er å sette opp grunnlaget skikkelig (referanse).
  
- Mer informasjon når man trunkerer(?)
  
- Klarer vi kartlegge den mørke materien? Hva tar oss lengst mot det endelige målet?

## Møte 22.03.23

Kapitel for rapport: "What batch size should we use?"

- Two experiments: different batch size and everything else the same
- Observations
- Interpretation
- Use the result to shape the rest of the projects parameters

Det er noe med algoritmen som gjør at batch size spiller en rolle. Godt å skrive om i rapporten.

HGS tipper at med små nettverk har ikke batch size så mye å si, men det blir vesentlig ved større nettverk.

- Trenger antageligvis ett kritisk minste antall batches (700 batch size med 3000 bilder gir muligens ikke nok batches). Den gjør en normalisering, der den kanskje mister noe.

HGS enig i at vi skal fokusere mye på å skrive fram til påske.

Ødelagte roulette bilder:

- $R_E < ||\chi||$
- Chi er avstanden fra origo+y til linsen
- 
- $R_E > \sigma + ||\tilde{n}|| = \text{actualAbs} = \text{sqrt}(x^2+y^2)$
- $R_E > \sigma X + ||\tilde{n}|| X$
- $\tilde{n} = \text{eta}$

Tror vi må bruke cartesian coordinates, men polar er støttet  
Hvis linse kommer for nært E\_R får vi problem(?)

① DRAW  $R_E$  FROM PDF

↓

THIS DETERMINES LOWER BOUND ON  $\eta$

↓

② DRAW  $L_L < \eta_{ACT} < L_U$  CHOOSE

↓

CALC  $\eta_{ACT}$ ,  $\sigma_{max}$

③ DRAW  $\sigma_{min} < \sigma < \sigma_{max}$

CONCLUSION  
DRAW  $R_E, \eta_{ACT}$   
↓  
GET  $\sigma$

$\rightarrow \sigma \chi + \|\bar{\eta}\| \chi$

$d^2 \left[ \frac{1}{(d-\Delta d)^2} + \frac{1}{(d+\Delta d)^2} \right] = 2(d-\Delta d)^2 (d+\Delta d)^2$  What batch size should we use?

$d^2 \left[ \frac{1}{(d-\Delta d)^2} + \frac{1}{(d+\Delta d)^2} \right] \sim 2(d^2 - \Delta d^2)^2$

$2d^2 \Delta d^2 =$

$\frac{1}{(d-\Delta d)^2} + \frac{1}{(d+\Delta d)^2} = \frac{2}{d^2}$

$\frac{d^2 (d-\Delta d)^2}{(d-\Delta d)^2 (d-\Delta d)^2} + \frac{d^2 (d+\Delta d)^2}{(d+\Delta d)^2 (d+\Delta d)^2} =$

$\frac{2d^2}{(d+\Delta d)^2 (d-\Delta d)^2}$

$q = \frac{m}{(d+\Delta d)^2}$   
 $+ \frac{m}{(d-\Delta d)^2}$

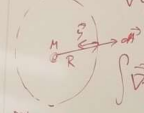
$q' = \frac{2m}{d^2}$

$+ \Delta d^2$  - Too expensive  
different batch sizes  
everything the same


- Observation  
- Interpretation

GAUSS GRAV. LEMMA  
 $\nabla \cdot \vec{g} = -4\pi G \rho$

CONCLUSION  
 DRAW  $R_E, \eta$   
 GET  $\sigma$

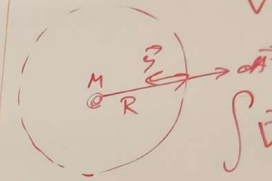

 $\int \vec{\nabla} \cdot \vec{g} dV = -4\pi G \int \rho dV$   
 DIV. TH.  $\Rightarrow \int \vec{g} \cdot d\vec{A} = -4\pi G M$   
 $\Rightarrow \int \rho r^2 d\Omega d\epsilon = -4\pi G M$   
 $\Rightarrow -4\pi R^2 g = -4\pi G M \Rightarrow g = \frac{GM}{R^2}$

$\rho \sim \text{CONST}$   
 $\alpha \sim \frac{1}{r^2}$


 $d^2 \frac{d-d\Delta d}{d^2 [2d^2 - 2d\Delta d - 3a^2]}$   
 $\frac{1}{(d+\Delta d)^2}$   
 $\frac{d^2 (d-d\Delta d)}{(d+\Delta d)^2 (d-d\Delta d)}$

① DRAW  $R_E$  FROM PDF  
 $\Downarrow$   
 THIS DETERMINES LOWER  
 BOUND ON  $\eta$   
 $\Downarrow$   
 ② DRAW  $L < \eta < L_U$  CHOOSE  
 $\Downarrow$   
 CALC  $\sigma_{min}, \sigma_{max}$   
 ③ DRAW  $\sigma_{min} < \sigma < \sigma_{max}$

CONCLUSION  
 DRAW  $R_E, \eta$   
 GET  $\sigma$


 $\nabla \cdot \vec{g} = -4\pi G \rho$   
 $\int \vec{\nabla} \cdot \vec{g} dV =$   
 DIV. TH.  $\Rightarrow \int \vec{g} \cdot d\vec{A} = -4\pi G M$   
 $\Rightarrow \int \rho r^2 d\Omega d\epsilon = -4\pi G M$   
 $\Rightarrow -4\pi R^2 g = -4\pi G M$

CONCLUSION  
 Draw  $R_c$ ,  $\eta$  ACT  
 ↓  
 GET  $\sigma$

$\vec{\eta} = (x, y)$  (R <sup>phi</sup> ~~theta~~)

$\|\vec{\eta}\| = \text{actual ALs}$   
 $= \sqrt{x^2 + y^2}$

$R_c > \sigma \chi + \|\vec{\eta}\| \chi$

$\sigma \sim \text{konst}$   
 $\alpha \sim \frac{1}{\sigma}$

$d^2 \left[ \frac{d-d_0}{d^2} \right]$   
 $d^2 (d^2 - 2d_0)$

$\frac{d^2}{(d+d_0)^2}$

# Møte 12.04.23

Now our work should focus on PsiFunctionSIS generated images not Roulette.

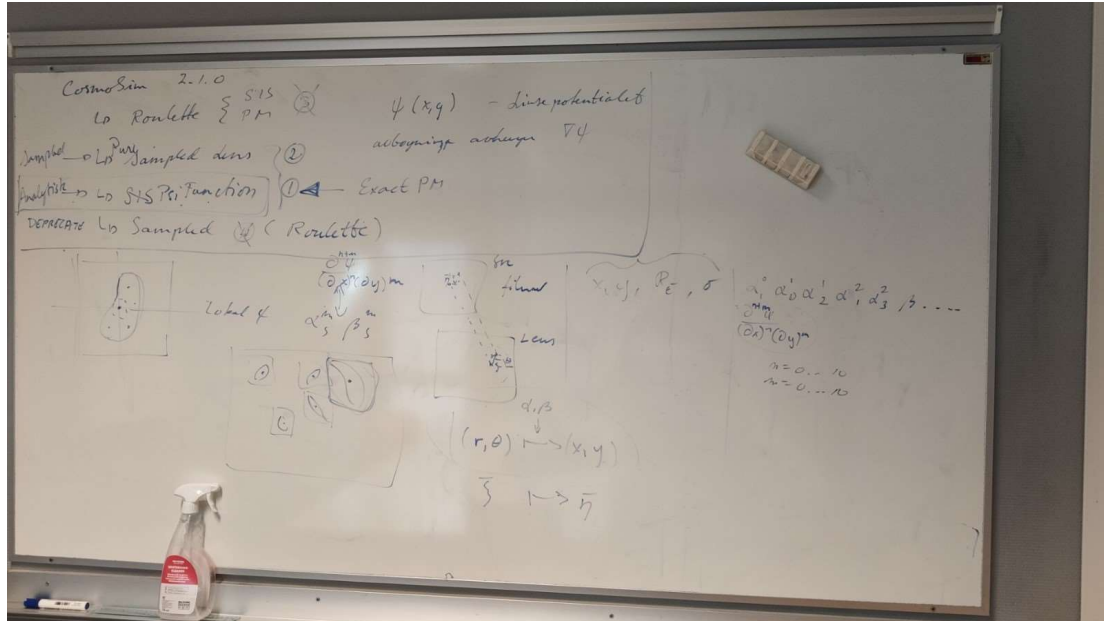
Roulette may be useful in the future but not now.

Make bigger mages and crop them so more of the centered image is used up by galaxy.

Now is time to **focus**, write experiments,

We wrote a script, describe why, how with 3 pages(uffff)

Write about Roulette, what we found works, although we moved away from them.





## Møte 19.04.23

- Sett opp issue på github angående "RuntimeWarning: overflow encountered in scalar add" (at centering "ødelegger" bilder).
- Vi bruker heller roulette enn psifunction pga det er enklere å jobbe rundt problemene
- Vi lagde formel for å sette maks/min R og sigma for å unngå konvergensringen
- HGS klarer ikke lage en robust maske I tide. Tidligere laget han en "directed component", en maske basert på det, som kunne vært brukt. Den burde være implementert I samme modul som centerimage.

## Møte 03.05.23

- Bilder skulle lages med raytracing ikke roulette med amplituder
  - Var ikke tenkt at man skulle generere med mange amplituder med roulette
  - Hvis du insisterer på roulette kan du spytte ut mange nok amplituder
  - Hvis vi lager 100/200/whatever og bare bruker første 5/6/whatever, går det fint? Burde gå fint.
  -
- Vi skal nå gjøre ting raskest mulig, ikke perfekt. Kun rapporten skal være perfekt, men alt annet skal kun være "bra nok".
- IDUN er en del fikling, men verste fiklingen var bibliotek og bygg-systemet som er ferdig.
  - IDUNbuild.sh for å bygge
    - Hvis det fungerer er det lite arbeid, hvis ikke er det mye
    - Største utfordringer er knyttet til module-load
    - Du må eksplisitt laste modulene du vil ha, ellers får du nyeste
    - Enten virker det, eller ikke
  - CosmoML/Tests/LargeSet/ slurm scripts
    - Rediger scripts for å legge de i IDUN kø
    - Når det begynner har vi rett på ressursene vi har bedt om
    - Images.slurm er eksempel på en jobb som generer bilder
      - Linje 2 velger cpu eller gpu
      - 4: maks klokkeid (maks kjøretid før stopp)
      - 6: c12 = antall cpu kjerner
      - 10: endre epost
      - 7: array kjører 100 jobber nummerert fra 0-99, ta bort når eksprementering
      - SBATCH er ting køsystemet leser
      - 14-23: diagnostic
      - 28-: laste moduler
      - Gammel versjon som bruker c++ versjon, er ikke testet på nye python-scriptene på idun
      - 35: kritisk, python-kjøremiljø; inneholder mer enn bare torch (ikke scipy)
      - Må skjønne hvordan module load fungerer, ha kontroll på python-kjøremiljø, skjønne konfigurasjon sbatch
    - **Cartesian120k2.slurm:**
      - **Mest relevant for oss**
      - Tilsvarende machine learning job.
      - 7: hvor mange gpus
      - 8: velger type GPU
      - Kjører ikke simulator, kiun pytorch så mindre module load
      - Klarte ikke bruke pre-installert python
    - Cartesian120k2plot.slurm
      - 33: <<EOF = mindre enn mindre enn EOF (EOF er linje 44)
  - Kjører IDUN-koden gjennom Git.
  - Kan lage egen IDUN-branch om nødvendig. HGS har gitt opp på å bruke 2 like grener samtidig.
  - CMAKE sjekker om pnavn begynner med IDUN, og kjører egen kode hvis det gjør det



## Møte 10.05.23

In 3.4 Hyperparameter optimisation, include more graphics showing the results from changing the hyperparameters if we have them. "Choosing the optimizer" already has a graph, need more like that with results alongside.

Write more about Idun scripts and how much faster we could train with Idun.

Combine figure 3.4 to 3.12 in groups that makes sense, and put them at the end of chapter 3.2 (Right before 3.3 Creating a reference network).

Explain the groups in paragraphs, each paragraph explaining one group (example: "figure 3.6 shows different results with different learning rates. 3.6a has a learning rate of ..." blabla

"Figure 3.14 justifies our choice" in "Loss function choice" is not scientific enough. Explain why figure 3.14 justifies it.

Move AlexNet explanations from 3.3 to theory 2.2 (only the parts that explain it)

import code snippets as code into latex, not screenshots from pycharm (at least no red lines under text....)

Explain simply how formula 3.1 works, and how we found it (trial-and-error but make it sound like it was skills)

Make better name for table 3.1 ("# better than average" is not correct) and fix the text explaining it to reflect what is being shown in table

## Møte 16.05.23

Kode-vedlegg:

Den ene er at siden vi legger ved som vedlegg, og/eller henviser til open-source, så må vi skille mellom det vi har skrevet, det vi har lånt, og det som vi har lånt+modifisert såpass at det ikke kan skilles. Om vi har endret et opensource prosjekt til graden at det ikke kan kjennes igjen, så går det fint men skriv det. Ikke bruk kode fra andre uten at det nevnes.

Når det er som vedlegg har det neppe for seg å kopiere inn kode vi ikke har skrevet selv.

På nettverk vi har tatt og så vidt modifisert: Ta tid i development til å forklare hva vi har endret og hvorfor. Det er åpenbart vi har tatt kode og testet den.

Vi kan gjøre det mer tydelig at vi tar utgangspunkt i koden vi fikk i fjor. Forklare hva vi endrer i development for hver prototype. Dette er programutviklingsprosjekt ikke så mye koding. Vi tar komponenter og lodder sammen.

Kan enten ikke forsøke tippe  $\chi$ , la den være kjent (fast eller vvarierende verdien). Enkleste hvis vi vil tippe andre parameter.

Kan bruke rødshiftanalyse for å vite hvor langt unna linsen er.

Alternativt kan man prøve hypotesen om at man ikke kan predicte  $\chi$  og  $r_e$ . Sjekke at man får samme resultat om ratio er samme.

I fjor fant de ut det var vanskelig å finne de ut uavhengig av hverandre. Det er ikke klart om det er sånn eller om det burde være slik eller om det er slik når  $\chi$  er nær 0 og 1.

Hvis  $\chi$  er konstant kan vi teste uten problem. Hvis  $\chi$  varierer mellom 30-70? klarer vi finne  $e_r$  og  $\chi$  fornuftig eller er de for avhengig av hverandre.

Legg til at vi bekrefter det de fant i fjor at varierende  $\chi$  er for vanskelig før vi bestemmer  $\chi$  skal være 50.

Nevn i further work at det er interessant å finne ut hvorfor  $\chi$  og  $r_E$  er så avhengig av hverandre, og om det kan fikses.

Viktig å få fram i rapporten av problemer er interessant men vi ikke har tid, interessant og vi tar tid, eller ikke interessant og ikke tar tid.

Nevn om vi kaster tvil over det fra i fjor, eller om vi bekrefter det fra i gjør, eller om vi finner noe de ikke nevnte/fant i fjor.

Er det mulig å få samme svar med forskjellig  $\chi/r_e$  i tavle-ligningen?

svar: du har 2 ligninger og 2 ukjent. ligningene er 2. grad, så du kan ha 2 løsninger. I prinsippet vil du ha 2 løsninger for  $r_e$  for hver ligning for hver verdi av  $\chi$ . Siden vi har 2 ligninger burde det være 2 løsninger. 2 ligninger 2 ukjente = 2 muligheter. Enten unik løsning eller uendelig løsninger i en dimensjon. Siden ligning ikke er linær er det vanskelig å si. HGS trodde i fjor det var uendelig løsningsrom. Det ser ut som det er slik i ett smalt område av  $\chi$  (30-70 f eks), men ikke nær 0 eller 1.

Hvis du beveger deg langt (nær 0 eller 1) spiller det en stor rolle.

Det hadde vært interessant å gjenta eksperimentene i 3.1 til å ha  $\chi$  fra 0.3 til 97 om vi har tid, for å sjekke.

Vi kan kjøre en test med kun ekstremverdier i  $\chi$  for å sjekke om hypoteser holder vann.

Kanskje 3 tester, 1 for 70-95, 1 for 30-70 og en for 5-30. Over 95 eller under 5 da vil hele systemet kollapse.

Når du treffer linseplanet får du en knekk (tavle figur). Hvis linsen treffer observator-planet skjer samme problemet. Linsen får for liten effekt ved ekstreme verdier. 5-95 er grei område å jobbe i. Skriv dette i rapporten som begrunnelse for hvorfor vi begrenser området.

Kan også bruke `\pageref` i latex

Formel på tavel er for pijnt mass, snakk med bdn når vi har skrevet.

## **F. Time lists**

**Kristian**

Total hours: 538

Date	Hours worked	Description of work
03.01.2023	3	Made Teams group, discussed expentations/work
04.01.2023		
05.01.2023		
06.01.2023	5	Read through wiki/2022 report
07/01/2023		
08/01/2023		
09/01/2023	9	Read rest of 2022 report, discussed possible work
10/01/2023	7.5	Installed software/dependencies/libraries
11/01/2023	7	Meeting, conan installation
12/01/2023	8	Pre-project report (almost done)
13/01/2023		
14/01/2023	4	Install CMake, finished pre-report
15/01/2023		
16/01/2023		
17/01/2023		
18/01/2023	6.5	Finished C++ setup, researched ML and possibilities
19/01/2023	7.5	Started looking at ML code
20/01/2023	6	Understanding ML code, started final report
21/01/2023		
22/01/2023		
23/01/2023	2	Generated train.csv, researched .slurm files
24/01/2023	4	Researched slurm files, installed WSL
25/01/2023	4	Meeting, python code dive
26/01/2023		
27/01/2023		
28/01/2023		
29/01/2023		
30/01/2023	3.5	Installed linux
31/01/2023	4	Installed linux
01/02/2023	3	Installing conan/cmake + dependencies
02/02/2023	7	Generated dataset
03/02/2023	6.5	Tested ml code
04/02/2023	4	Tested ml code
05/02/2023		
06/02/2023	2	Started backlog
07/02/2023	7	Tested ML code on GPU, prepared meeting
08/02/2023	7	Cmake/conan installation (develop branch)
09/02/2023	2	Fixed yesterdays problems
10/02/2023	8	Tested machine learning on 100k images
11/02/2023		
12/02/2023		
13/02/2023		
14/02/2023	5	More code testing / benchmark of previous code



15/02/2023	7.5	Tried different hyperparameters
16/02/2023	6	Tried different hyperparameters
17/02/2023	5	Tried different hyperparameters
18/02/2023		
19/02/2023		
20/02/2023	2	Generating images for testing same images
21/02/2023	6	Proving images are the same
22/02/2023	3	Meeting + coding
23/02/2023		
24/02/2023	4	Proving images are the same
25/02/2023		
26/02/2023		
27/02/2023	5	Researching neural networks
28/02/2023	5	Researching neural networks
01/03/2023	6	Researching neural networks
02/03/2023		
03/03/2023		
04/03/2023		
05/03/2023		
06/03/2023	7	Pytorch tutorial
07/03/2023	6	Pytorch tutorial
08/03/2023	5.5	Møte + AlexNet
09/03/2023	5	AlexNet coding
10/03/2023		
11/03/2023		
12/03/2023		
13/03/2023	N/A	Focusing on other subject
14/03/2023	N/A	Focusing on other subject
15/03/2023	N/A	Focusing on other subject
16/03/2023	N/A	Focusing on other subject
17/03/2023	N/A	Focusing on other subject
18/03/2023	N/A	Focusing on other subject
19/03/2023	N/A	Focusing on other subject
20/03/2023	N/A	Focusing on other subject
21/03/2023	4	Report writing
22/03/2023	5	Report planning and meeting
23/03/2023	5	Researching GAN
24/03/2023	6	Experimenting with GAN
25/03/2023		
26/03/2023		
27/03/2023	7	Coding GAN implementation
28/03/2023	4	Coding GAN implementation
29/03/2023	5	Coding GAN implementation
30/03/2023	3.5	Coding GAN implementation

31/03/2023	6	Coding GAN implementation
01/04/2023		
02/04/2023		
03/04/2023	5	Coding GAN implementation
04/04/2023	5.5	Report writing + building C++
05/04/2023	6	Trying different networks
06/04/2023	2.5	Trying different networks
07/04/2023	6	Trying different networks
08/04/2023		
09/04/2023		
10/04/2023	5	
11/04/2023	2	Report writing
12/04/2023	4	Report writing
13/04/2023	4.5	Report writing
14/04/2023	4	Fixing python
15/04/2023	1	Coding ML networks
16/04/2023	3	Building new version
17/04/2023	5.5	Write report + generate reference data set
18/04/2023	7	Modifying networks
19/04/2023	5.5	Modifying networks
20/04/2023	9.5	Report writing based on feedback
21/04/2023	7	Report writing
22/04/2023		
23/04/2023		
24/04/2023	8	Old code testing + report
25/04/2023	8.5	Old code testing + report
26/04/2023	8.5	Write report
27/04/2023	6	Researching networks
28/04/2023	7	Researching networks
29/04/2023		
30/04/2023		
01/05/2023	6	Report writing + testing networks
02/05/2023	8	Report writing + testing networks
03/05/2023	6	Report writing + testing networks
04/05/2023	7	Report writing + testing networks
05/05/2023	8	Report writing + testing networks
06/05/2023	4	Report writing + testing networks
07/05/2023	4	Report writing
08/05/2023	8	Report writing
09/05/2023	9	Report writing
10/05/2023	8	Report writing
11/05/2023	14	Report writing
12/05/2023	3	Report writing
13/05/2023	4	Report writing

14/05/2023	5.5	Report writing
15/05/2023	7.5	Report writing
16/05/2023	10	Report writing
17/05/2023	13	Report writing
18/05/2023	10	Report writing + making presentation
19/05/2023	10	Report writing + writing script
20/05/2023	9	Report writing + recording presentation video
21/05/2023	12	Report writing

**Modestas**

Total hours: 536

Date	Time	Description of what have been done
03/01/2023	2	Talked about expectations and weekly plans, decide the next meeting activity.
04/01/2023		
05/01/2023		
06/01/2023	5	Start readin up on wiki
07/01/2023		
08/01/2023		
09/01/2023	9	Read rapport and disscuse workspace
10/01/2023		
11/01/2023	8.5	Try running CosmoSim program
12/01/2023	8	Write pre-project report, try running Cosmosim, try VirtualBox(Ubuntu) for Cosmosim
13/01/2023		
14/01/2023	5	Working further with VirtualBox(Ubuntu)
15/01/2023		
16/01/2023	2.5	Working further with VirtualBox(Ubuntu)
17/01/2023		
18/01/2023	9	CosmoSim is finnaly working
19/01/2023	7	try different fits file viewer
20/01/2023	6	Read Gravitational lensing report by David and work further with fits
21/01/2023	4	Finnaly opend a FITS file, reaserch into different NN architectures
22/01/2023		
23/01/2023	2	Reading further on NN arhitecures
24/01/2023	4	Reading on papers about deep regression
25/01/2023	4	Meeting with supervisors, discust further with group
26/01/2023		
27/01/2023		
28/01/2023		
29/01/2023		
30/01/2023	2	Reading on CNN in regression task
31/01/2023	2	Reading on CNN in regression task
01/02/2023	2	Reading on CNN in regression task
02/02/2023	7	Setting up my home pc
03/02/2023	7	Setting up my home pc
04/02/2023	4	Setting up my home pc
05/02/2023		
06/02/2023		
07/02/2023	8	Setting up remote access to my home pc
08/02/2023	6	Continue on setting up home pc
09/02/2023	6	CosmoGUI work on home PC!!!

10/02/2023	8	Start running old mashine learning code on home pc
11/02/2023		
12/02/2023		
13/02/2023	6	Test old nn using different data sets
14/02/2023	6	Train network on 50k images
15/02/2023	7	Made a script to load in and test neural network trained weights and biases
16/02/2023	6	Try loading images from other directory than code directory
17/02/2023	5	Search for differnet networks to use in this project
18/02/2023		
19/02/2023		
20/02/2023		
21/02/2023	5	Continue researtch on differnet nn
22/02/2023	5	Had a meeting with supervisors, work with old ML code, continue researtch into nn
23/02/2023	2	Continue reading up on nn architectures
24/02/2023		
25/02/2023		
26/02/2023		
27/02/2023		
28/02/2023	7	Look at code changes that where made on inception source code
01/03/2023	7	Beggint to implement Efficiantnet into existing code
02/03/2023	7	Continue on implement Efficiantnet into existing code
03/03/2023	4	Finde a fix for code so that more or less than 4 parameters could be estimated
04/03/2023		
05/03/2023		
06/03/2023		
07/03/2023		
08/03/2023	6	Meeting with supervisor, implementing AlexNet
09/03/2023	6	Investigating a problem where outputs are the same no mather the inputs
10/03/2023	6	Continue investigating the problem
11/03/2023		
12/03/2023		
13/03/2023		
14/03/2023		
15/03/2023		
16/03/2023		
17/03/2023		
18/03/2023		

19/03/2023

20/03/2023

21/03/2023

22/03/2023

23/03/2023

24/03/2023

25/03/2023

26/03/2023

27/03/2023

28/03/2023

29/03/2023

30/03/2023

31/03/2023

01/04/2023

02/04/2023

03/04/2023

04/04/2023

05/04/2023

06/04/2023

07/04/2023

08/04/2023

09/04/2023

10/04/2023	8	Migrating from windows server 2019 to windows 11
11/04/2023	8	Fine tunig the parameter for data set with SIS rulette
12/04/2023	8	Meeting with supervisor, work on SIS Phi function data set
13/04/2023	8	Working further on Phi function data set
14/04/2023	8	
15/04/2023		
16/04/2023	5	Write rapport
17/04/2023	9	Write rapport
18/04/2023	8	Write rapport, make script for graphths.
19/04/2023	7.5	Meeting with supervisors, writing rapport, adding a script for cropping generated images
20/04/2023	9	Write rapport, make jupyter notebook code for graph
21/04/2023	8	Small meeting with supervisor discusing our rapport, write rapport
22/04/2023	3	Setup and run expierments
23/04/2023	7	Setup and run expierments
24/04/2023	8	Write rapport, make graphs out of colected data
25/04/2023	12	Write rapport, redo EfficiantNet and add extra layer option to Inception3 and AlexNet

26/04/2023	12	Write rapport, add ConvNeXt, ResNet, VGG, DensNet,
27/04/2023	12	Write rapport, add NASNet
28/04/2023	8	Write rapport, add MnasNet, meet supervisors
29/04/2023	4	Write rapport
30/04/2023		
01/05/2023	8	Start working on IDUN
02/05/2023	6	Write rapport, troubleshoot IDUN
03/05/2023	8	Write rapport, meet supervisors, start testing IDUN
04/05/2023	4	Write rapport, setup test for IDUN
05/05/2023	10	Write rapport, meet supervisors, setup test for IDUN
06/05/2023	6	Write rapport, setup test for IDUN
07/05/2023	3.5	Write rapport, setup test for IDUN
08/05/2023	8	Write rapport, setup test for IDUN
09/05/2023	5	Write rapport, collect the data from IDUN tests make graphs
10/05/2023	8	Write rapport, collect the data from IDUN tests make graphs
11/05/2023	6	Write rapport, collect the data from IDUN tests make graphs
12/05/2023	7	Write rapport, meet supervisors, collect the data from IDUN tests make graphs
13/05/2023	8	Write rapport, make graphs
14/05/2023	6	Write rapport, make graphs
15/05/2023	8	Write rapport, make graphs
16/05/2023	6	Write rapport, meet supervisors
17/05/2023	10	Write rapport, make graphs
18/05/2023	10	Write rapport
19/05/2023	12	Write rapport
20/05/2023	10	Write rapport
21/05/2023	9	Write rapport
22/05/2023	12	Write rapport



## G. Code tutorial

Here is a small tutorial on how CosmoSim and CosmoML code was setup and used. For CosmoSim the IDE software used was CLion, and for Cosmo-ML it was PyCharm. Everything was run on Windows.

Below is a step by step example for building and working with CosmoSim:

Step 1 - Copy the repository with command:  
`git clone <CosmoSim url>`

Step 2 - Reopen project in new folder.

Step 3 - Change toolchains in Clion settings to visual studio (x86\_amd64) also visual studio has to be on top of the list in toolchains tab.(Install visual studio version 16 with "Desktop development with C++" option if needed).

Step 4 - Install CMake if you don't have it already.

Step 5 - Go to Clion settings->CMake->Buildtype->Release.

Step 6 - Pip install conan 1.58.0(version 2 created problems) if you don't have it already.

Step 7 - Set conan profile to visual studio version 16 in directory `C:\Users\<user>\.conan\profiles\default`

Step 8 - Run this command in terminal:  
`conan install . -if .\cmake-build-release\ -b missing`

Step 9 - Go to Clion CMake tab -> Reset cache and reload project.

Step 10 - Change to CosmoSimPy -> Build.

After all these step it should be possible to run CosmoGUI.py with command:  
`python .\Python\CosmoGUI.py`

Look at the appendix B code for adding cropping feature to image generation.

## G. Code tutorial

To generate a data set for training, first data set file needs to be created. This uses datasetgen.py all preferred setting and options need to be adjusted here. When finished this command can be run to generate your file:

```
python .\Python\datasetgen.py
```

Lastly to generate all the images use datagen.py. Here is an example with few used arguments:

```
python .\Python\datagen.py -D "<directory for images>" -i "<data set csv file>" -Z  
"1000" -q -C -M
```

-Z is for setting image resolution, -q is for cropping, -C is for centering the images, -M is for mask when working on roulette lens.

This was the method with which all data sets in this project was created.

## G. Code tutorial

Below is a step by step guide for setting up and working with Cosmo-ML using Nvidia GPU:

Step 1 - Copy the repository with command:

```
git clone <Cosmo-ML url>
```

Step 2 - Install cuda toolkit from Nvidia if you don't have it already. It can be found here: <https://developer.nvidia.com/cuda-toolkit>

Step 3 - We recommend to use virtual python environment for this project. Then install all the needed libraries:

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

```
pip install -r requirements.txt
```

Step 4 - Selecting network and few hyperparameters: open `CudaModel.py` -> write a network to test on line 18 `model='<wanted neural network>'`(all neural network options are in `MLSystem.py`). At the same time number of epochs and learning rate can be chosen.

Step 5 - Choosing the rest of hyperparameters: open `MLSystem.py` here batch size, optimizer, criterion(loss function) is chosen.

Step 6 - To start testing command like this with arguments should be used:

```
python CudaModel.py -t "<csv file for training data set>" -i "<csv file for testing data set>" -T "<directory for training data set images>" -I "<directory for testing data set images>" -p "<csv file for predicted values on test set at the end of training>" -o "<csv file for total loss data of every epoch>"
```

Setup for IDUN is the same as for local machine.

Since IDUN can run multiple test at the same time, for every test a `CudaModel.py` file replacement is created like `VGG_example.py` that goes with slurm file `VGG.slurm`.

To start testing on IDUN command like this should be used:

```
sbatch VGG.slurm
```

To check the status of the tests:

```
squeue -user=<username>
```

To cancel the test:

```
scancel -name=<job name>
```

## H. Code for comparing images

```
import PIL.Image
from PIL import ImageChops
import math, operator

identical = 0
ld = 0.0
min = 1
max = 100

def rmsdiff(im1, im2):
    "Calculate the root-mean-square difference between two images"
    global identical
    global ld
    h = ImageChops.difference(im1, im2).histogram()
    e = ImageChops.difference(im1, im2)
    if e.getbbox() is None:
        identical += 1
    if e.getbbox():
        if float(math.sqrt(reduce(operator.add,
            map(lambda h, i: h*(i**2), h, range(256))
        ) / (float(im1.size[0]) * im1.size[1])) > ld):
            ld = float(math.sqrt(reduce(operator.add,
                map(lambda h, i: h*(i**2), h, range(256))
            ) / (float(im1.size[0]) * im1.size[1])))
            print(str(im1))

    # calculate rms
    return math.sqrt(reduce(operator.add,
        map(lambda h, i: h*(i**2), h, range(256))
    ) / (float(im1.size[0]) * im1.size[1]))

def equal(im1, im2):
    global identical

    return ImageChops.difference(im1, im2).getbbox() is None
```

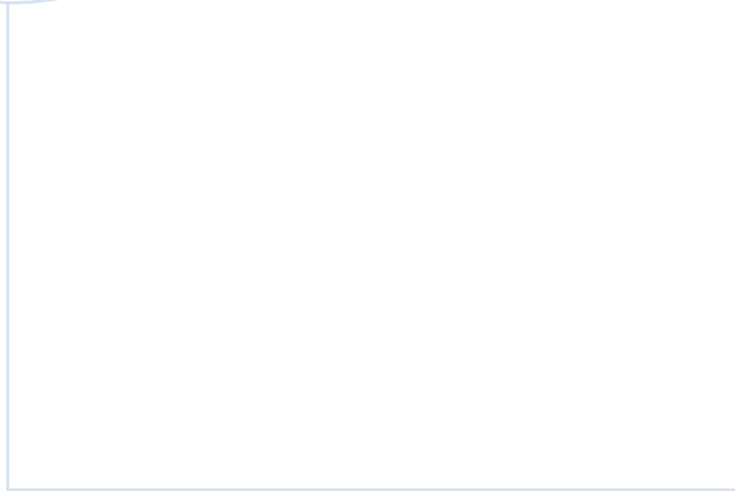
## H. Code for comparing images

```
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value

compare = True
if compare:
    counter = 1
    print("Starting tests: ")
    print("-----")
    for i in range(min, max):
        for j in range (1, max-i+1):
            x = str(i)
            y = str(i+j)
            im1 = PIL.Image.open(f'./92to98/image-{x}.png')
            im2 = PIL.Image.open(f'./92to98/image-{y}.png')

            print(f'Test {counter} (comparing image {x} and {y}):')
            print("Equal: " + str(bool(equal(im1, im2))))
            print("Difference: " + str(float(rmsdiff(im1, im2))))
            print("-----")
            counter += 1

diff = max-min
sum = int(diff * ( diff + 1 ) / 2)
print(f'Number of identical pairs: {identical} out of {sum} pairings')
print(f'Largest difference: {ld}')
```



 **NTNU**

Norwegian University of  
Science and Technology