NTNU
Norwegian University of
Science and Technology

Master's thesis

2020

Camilla Marie Dalan

Master's thesis

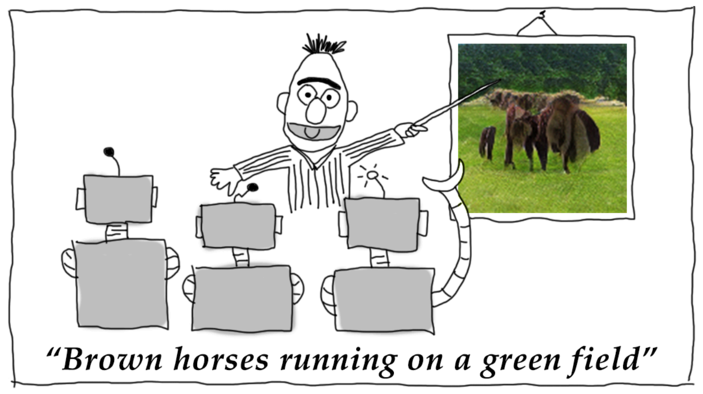
NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Camilla Marie Dalan

Text-to-Image Synthesis with a Pre-Trained Deep Learning Language Model

Learning T2I Synthesis with BERT

June 2020



NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Text-to-Image Synthesis with a Pre-Trained Deep Learning Language Model

Learning T2I Synthesis with BERT

Camilla Marie Dalan

Computer Science
Submission date: June 2020
Supervisor: Björn Gambäck

Norwegian University of Science and Technology
Department of Computer Science

Camilla Marie Dalan

Text-to-Image Synthesis with a Pre-Trained Deep Learning Language Model

Learning T2I Synthesis with BERT

Master's Thesis in Computer Science, Spring 2020

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Current state-of-the-art text-to-image (T2I) synthesis models are based on the architecture of Generative Adversarial Networks (GAN). These models perform well on limited datasets that contain images and corresponding image captions within one class or category, like birds, in that the images generated are realistic and of reasonably high resolution (256 x 256 pixels). However, they fail to generate high quality images when trained on more complex datasets containing a higher number of classes and complex scene compositions. Natural language processing models, such as the Bidirectional Encoder Representations from Transformers (BERT), hold great promise in improving T2I synthesis models.

This Master's Thesis integrates a recent GAN-based T2I synthesis model, MirrorGAN, with BERT, through MirrorGAN's sub-module for re-generation of image captions. The performance of this novel model is assessed through quantitative measures such as the Bilingual Evaluation Understudy (BLEU) for comparing the two versions of the image captioning module and the Fréchet Inception Distance (FID) for comparing the performance of the resulting T2I synthesis models. Additionally, a qualitative survey involving 33 participants was conducted to further compare the two models.

The comparison of the two versions of the image captioning module indicates that the novel version clearly outperforms the original. However, comparing the two versions of the final model through the Fréchet Inception Distance metric yields greater results for the original version for all epoch steps but the last. This is supported by the results from the conducted survey. Moreover, the novel version outperforms the original in the last epoch step not due to the novel version improving, but rather the deterioration of the original model. Further inspection indicates that both models suffer from commonly faced problems when training GANs, such as vanishing gradients and mode collapse. Moreover, the images can in some cases be fairly visually pleasing, which might propose that a T2I system could have other value apart from generating realistic images.

With further model optimisation and mitigation strategies for the problems encountered, the results indicate that this new combined model could produce superior images and should be the subject of further research.

Sammendrag

Dagens beste modeller innen tekst-til-bilde-generering (T2I) er basert på Generative Adverserielle Nettverk (GAN). Disse modellene gjør en god jobb nå de er trent på begrensede datasett som kun inneholder bilder og korresponderende bildetekst innen én klasse eller kategori, slik som fugler eller blomster. Når de er trent på slike datasett, klarer disse modellene å generere realistiske bilder av rimelig god kvalitet (256 x 256 piksler). På den andre siden mislykkes de i å generere bilder av høy kvalitet om de er trent på et mer komplekst datasett med et høyere antall klasser og mer komplekse scener. Modeller for naturlig språkbehandling, slik som BERT — Bidirectional Encoder Representations from Transformers — har gode forutsetninger til å forbedre slike T2I-modeller.

Denne masteroppgaven integrerer MirrorGAN, en nylig fremtredende GAN-basert T2I-modell, med BERT. Dette gjøres gjennom MirrorGANs submodul for regenerering av bildetekst. Ytelsen til denne nye kombinasjonen ble evaluert gjennom både kvantitative og kvalitative metoder. De kvantitative metodene inkluderer Bilingual Evaluation Understudy (BLEU) for å sammenligne de to versjonene av submoduler for regenerering av bildetekst, og Fréchet Inception Distance (FID) for å sammenligne den originale og den nye T2I-modellen. I tillegg ble det utført en kvalitativ spørreundersøkelse med 33 deltakere for å videre sammenligne de to modellene.

Sammenligningen av de to versjonene av modulene for regenerering av bildetekst viser at den nye versjonen helt avgjort utkonkurrerer den originale versjonen. På en annen side, ved sammenligning av de to endelige T2I-modellene gjennom FID-målingen, viser den originale versjonen å gjøre det bedre enn den nye. Dette er også støttet av resultatene fra den utførte spørreundersøkelsen. I tillegg kan man se at den nye versjonen utkonkurrerer den originale i det siste epokesteget; ikke fordi den nye versjonen ble bedre, men heller fordi den originale ble verre. Videre inspeksjon av resultatene indikerer at begge modellene lider av vanlige problemer som oppstår under trening av GANs. Disse problemene inkluderer forsvinnende gradienter og moduskollaps. I tillegg kan de genererte bildene i noen tilfeller være visuelt tilfredsstillende, noe som kan indikere at et T2I-system har verdi utenom å generere realistiske bilder.

Resultatene indikerer at med videre modelloptimalisering og benyttelse av strategier for å begrense treningsproblemene som ble møtt, kan den nye kombinerte modellen produsere overlegne bilder. Dette er derfor et tema som bør utforskes videre.

Preface

This Master's Thesis serves as the final part of obtaining a degree of Master of Science in Computer Science at the Norwegian University of Science and Technology (NTNU). It is written at the Department of Computer Science under the supervision of Björn Gambäck.

I wish to thank Taesung Park, Jacob Devlin, Maria-Elena Nilsback, Catherine Wah, Scott Reed, Han Zhang, Tao Xu and Tingting Qiao for permitting the use of the figures and results from their respective papers. An additional thanks goes out to Tingting Qiao for quick responses to my emails and valuable discussion.

I would also like to thank everyone who participated in the conducted survey, and my family for valuable support, especially during the last months of writing.

Furthermore, a thanks goes out to the HPC group at NTNU for the use of the IDUN cluster, and to my supervisor, Björn Gambäck, for providing valuable remarks and supervision.

A special thanks goes out to Tyler Stewart for love, support and fantastic food throughout the process of writing.

Additionally, I would like to refer to the use of figures published under the licenses Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)[1], Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)[2], and Attribution 4.0 International (CC BY 4.0)[3].

Camilla Marie Dalan
Oslo, 24th June 2020

[1]https://creativecommons.org/licenses/by-sa/4.0/
[2]https://creativecommons.org/licenses/by-nc-sa/4.0/
[3]https://creativecommons.org/licenses/by/4.0/

Contents

List of Figures

List of Tables

1. Introduction

Text-to-Image (T2I) synthesis has fascinated the machine learning community for years. This is mainly due to the “multi-modal” problem of bridging the gap between text and image representations, where multi-modal represents the many ways of translating from one domain to another, e.g. text to image. Despite this fascination, significant advances have only been made in recent years. These advances are mostly based on the use of Generative Adversarial Networks (GANs; Goodfellow et al., 2014), where one network is pitted against another network. The first network, the generator, generates an image from noise and sends it to the second network, the discriminator. The discriminator, then, evaluates if this generated image belongs to a set of supplied images, before giving feedback to the generator which adjusts itself thereafter. The goal of this architecture is to train the generator to generate images that approximate a set of supplied images. Recent advances in T2I synthesis have predominantly focused on extensions to the original GAN architecture. These advances include the use of stacking generators to produce images of higher resolution (Zhang et al., 2017), attention to generate more detailed images (Xu et al., 2018), and text re-generation to provide additional feedback to the generator (Qiao et al., 2019). In the field of T2I, this text re-generation more specifically re-generates text from the generated image, before the original and the re-generated texts are compared.

An important part of T2I synthesis is the processing of natural language input to create text feature representations that include important visual features described in the text. This processing falls under the field of natural language processing (NLP). In the case of cycle-consistency, a processing also goes the other way: from image to text. In the same way as the challenge of generating images from text, generating text from image (I2T) is also multi-modal. However, I2T is a more well-defined challenge than T2I due to the limitations imposed by the structure of language.

The focus of T2I synthesis has mostly been on smaller datasets consisting of particular objects. The field of T2I synthesis has been reasonably successful at generating images of single objects, like birds or flowers, but struggles to generate images of more complex scenes (e.g. of interacting objects) and of higher resolutions (e.g. 256 x 256 pixels or higher). These observations can be seen in figure 1.1, and are important starting points for further research in the field of T2I synthesis.

With this in mind, there are two major parts of T2I synthesis that require more attention: NLP and generation of complex scenes of high resolution.

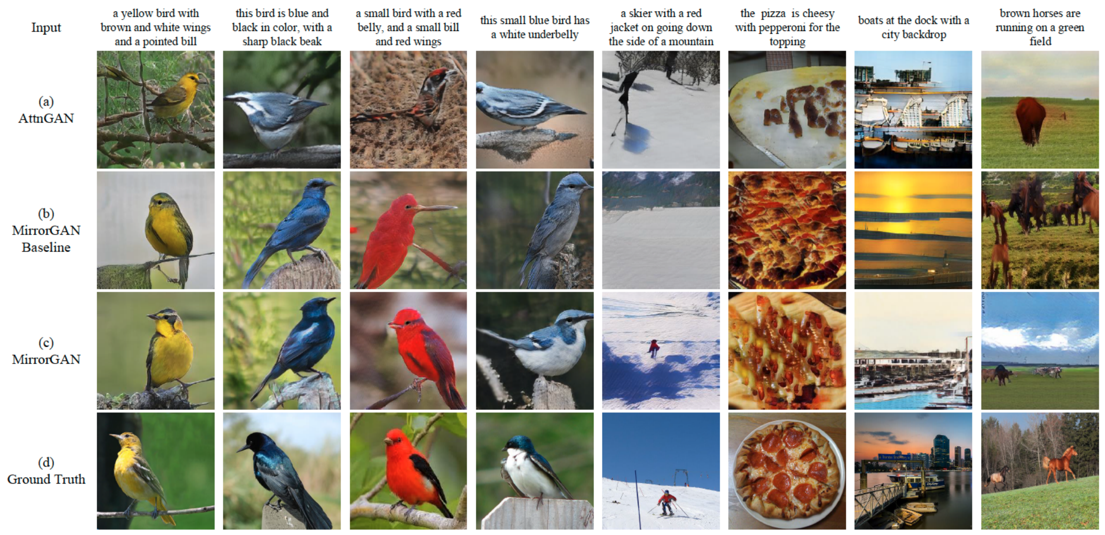

Input
a yellow bird with brown and white wings and a pointed bill
this bird is blue and black in color, with a sharp black beak
a small bird with a red belly, and a small bill and red wings
this small blue bird has a white underbelly
a skier with a red jacket on going down the side of a mountain
the pizza is cheesy with pepperoni for the topping
boats at the dock with a city backdrop
brown horses are running on a green field
(a) AttnGAN
(b) MirrorGAN Baseline
(c) MirrorGAN
(d) Ground Truth

Figure 1.1.: MirrorGAN example images. Figure from Qiao et al. (2019), with permission from Tingting Qiao.

1.1. Background and Motivation

One of the most prominent models in the field of T2I synthesis is MirrorGAN, introduced by Qiao et al. (2019). The model incorporates all major recent advances in the field (including the use of the GAN architecture, stacking and attention) in addition to employing text re-generation to provide a new state-of-the-art for GAN-based text-to-image synthesis.

An interesting aspect of T2I synthesis is the potential integration of newer NLP architectures and models. An important advance in the field of NLP in recent years is the Bidirectional Encoder Representations from Transformers (BERT) model, introduced by Devlin et al. (2018). The model was a huge breakthrough in the field of NLP, with versions being pre-trained on massive datasets (such as the whole English Wikipedia) and made easily available to the public. This facilitated the use of these powerful models by anyone. BERT has since been the basis of many state-of-the-art models in the field of NLP. Further, an interesting and potentially promising application of such pre-trained models is their incorporation with the MirrorGAN T2I synthesis model.

Like previous models, the work of Qiao et al. (2019) was mainly focused on generating images from descriptions of birds with training the model on the Caltech-UCSD Birds-200-2011 (CUB Birds-200) dataset. This dataset includes 8,855 training images of birds, with five text descriptions per image. In addition to the CUB Birds-200 dataset, the Microsoft Common Objects in Context (COCO) dataset is also commonly used for tasks like T2I synthesis and image captioning. This dataset is more complex and general than the CUB Birds-200 dataset as it contains an increased amount of images which depict a wide range of objects, including combinations of several objects in context. The first four columns of figure 1.1 show images generated by different models trained on the CUB Birds-200 dataset, including MirrorGAN in row three. The last four columns show images generated by the same models trained on the COCO dataset. It is evident that the images in the last four columns are far from perfect and, consequently, that such generalized T2I synthesis needs more attention.

Although the definition of quality of an image generated from text is debatable, it can be argued that it should include how well the image corresponds to the text, in addition to being of reasonable resolution. Additionally, the realism of an image can also be included in such a definition, where the ultimate goal is for the image to be indistinguishable from a real image. Although T2I performs well when dealing with a dataset focused on one object, like the CUB Birds-200 dataset, the field struggles when employing more generalized and complex datasets, like the COCO dataset. This can be seen in figure 1.1. The field has many applications, including art generation and computed aided design. It can also provide insight into areas like computational creativity. For these reasons it is an important field to develop further.

1.2. Goals and Research Questions

Goal *Explore the combination of a text-to-image synthesis system and a recent natural language processing model employing a complex dataset*

The focus of this Master's Thesis is the integration of the text-to-image model MirrorGAN and the natural language model BERT. The goal is to explore how such an integration can be performed and perform the integration before comparing the image generation capabilities of this new model relative to the original MirrorGAN model by Qiao et al. (2019). Both the new and the original versions of the model will be trained on the COCO dataset, which provides a more complex learning base than the CUB Birds-200 dataset.

The following research questions will assist in reaching this goal.

Research question 1 *How can the text-to-image system and the natural language model be combined?*

This research question addresses the task of finding what ways MirrorGAN and BERT can be combined, and details about how this integration can be performed. Additionally, the question implies the need for a study of how this integration performs by itself.

Research question 2 *How does the combination of the text-to-image system and the natural language model affect image generation?*

The aim for this research question is not to improve the image generation, but to analyze what differences there might be in the image generation by the two models, in addition to why such differences might arise. If the quality of the generated images of the original and the new models does not differ, the generated images by the new model may exhibit other features that are thought-provoking or otherwise interesting.

Research question 3 *How does the combination of the text-to-image and the natural language models perform when employing a dataset containing complex images?*

The field of T2I synthesis faces major problems when it comes to generating images of high resolution and of more complex character than depicting different types of birds. Consequently, the original and the new version of MirrorGAN will both be trained on the COCO dataset, which provides a more challenging and complex base compared to the CUB Birds-200 dataset.

1.3. Research Method

The goal of this Thesis is approached through experiments that build on an existing body of research, in addition to an evaluation of the findings. Both the original version of MirrorGAN and the version integrated with BERT (i.e. new version) will be trained on the COCO dataset. The results from the experiments are evaluated through the research questions introduced in the previous section.

1.4. Contributions

While the project implementation can be found on GitHub[1], this section provides a brief summary of the main contributions of this Master's Thesis. Thus, the main contributions are as follows:

1. *An overview of different GAN-based text-to-image generation systems.*
2. *A discussion on how BERT can be integrated with MirrorGAN.*
3. *Using BERT for the image caption re-generation in MirrorGAN.*
4. *A modular approach to the training of the final models and their sub-modules. This approach simplifies training of all networks separately before employing them to train the final model.*
5. *The use of parallel training of the models over multiple GPUs to accommodate for a higher batch size than what a single GPU can hold. This is needed as the amount of memory required for the COCO dataset quickly grows large.*
6. *A comparison of the two versions of the image caption re-generation sub-module, calculated through the Bilingual Evaluation Understudy (BLEU).*

[1] `https://github.com/cammiida/masters-project`

7. *A comparison of the two versions of MirrorGAN through conducting a survey, employing the Fréchet Inception Distance (FID) metric and performing subjective analysis.*
8. *A brief discussion of what other value GAN-based T2I systems might have, apart from generating realistic images.*

1.5. Thesis Structure

Chapter 2 provides the theory and background necessary to follow subsequent chapters. More specifically, it includes a general introduction to machine learning with artificial neural networks, an explanation of different GAN-based architectures, recent advances in NLP that are related to BERT, and a discussion of a recent image caption generation technique employed for the new version of the image caption re-generation sub-module.

Chapter 3 provides an overview of the current state of research with regards to GAN-based T2I synthesis, including a detailed explanation of MirrorGAN and its inner workings.

Chapter 4 explains the architecture employed for the contributions of this work. Moreover, the applied methodology for the implementation of the project is also discussed.

Chapter 5 introduces the conducted experiments and presents the results.

Chapter 6 provides an evaluation and discussion of the findings introduced in chapter 5. Additionally, it aims to answer the research questions and discuss how the overarching goal has been met. Moreover, it also includes a discussion of the limitations of the work conducted.

Chapter 7 concludes the work of this Thesis. Additionally, it provides a description of contributions, and summarizes potential future work that might be promising.

2. Theory and Background

This chapter presents background information and theory that is necessary to follow the rest of this Master's Thesis. Section 2.1 provides an introduction to machine learning including common terminology, inner workings of Artificial Neural Networks (ANNs) and training. Section 2.2 explains significant statistical functions and metrics for the work of this Thesis, while the subsequent section, section 2.3, provides an overview of different ANN architectures. The work of this Master's Thesis is focused on Text-to-Image (T2I) processing, which consists of two major fields in machine learning: image generation through the use of Generative Adversarial Networks (GANs) and natural language processing (NLP). With this in mind, sections 2.4 and 2.5 cater to these subjects, respectively. Furthermore, due to the MirrorGAN model employing image caption re-generation from the generated image, section 2.6 includes an introduction to automatic generation of image captions, called image captioning. Additionally, it provides an explanation of an image captioning technique employed in this work.

A handful of sections and subsections in this chapter are based on the preliminary studies to this Master's Thesis, conducted in the Fall of 2019. These sections include the main parts of section 2.1, section 2.1.3, most of section 2.3, section 2.4.1, the second halves of sections 2.5.3 and 2.5.6, and section 2.5.7.

Moreover; scalars, vectors and matrices follow the notation of being lowercase, bold lowercase and bold uppercase, respectively:

$$x = 1$$

$$\boldsymbol{y} = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

$$\boldsymbol{Z} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

2.1. Machine Learning

This section provides an introduction to machine learning and specifically to Artificial Neural Networks (ANNs). It includes the most important parts about the inner workings of ANNs, how they are used for prediction, and how they are trained.

Machine learning is a sub-field of artificial intelligence that employs algorithms that can automatically improve through a learning process, and focuses on tasks that are too complex to achieve otherwise. Machine learning is often implemented through the use of Artificial Neural Networks (ANNs). An ANN is a model that aims to learn how to predict an output based on a set of inputs through training. The model is comprised of

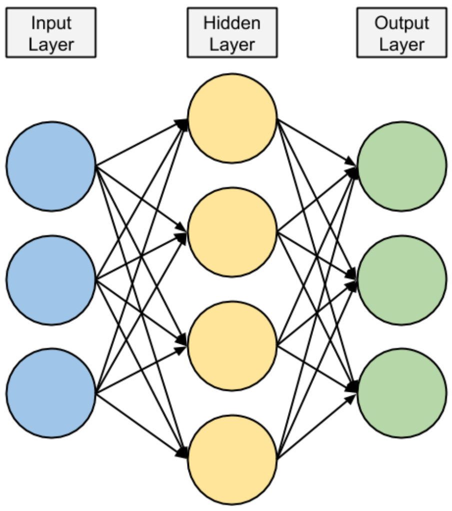


Figure 2.1.: A simple example of an artificial neural network.

nodes that are organized in layers. Each node in a layer computes a function based on its weights and bias values, and the outputs of the nodes from the previous layer. The weights and biases of the network are also called *parameters*. A simple an ANN can be seen in figure 2.1. The network learns through changing the values of its parameters through the process of trial and error. All other parameters for a network that are configurable by the developer are referred to as hyper-parameters.

The ANN was first presented in McCulloch and Pitts (1943) where the idea was computational mimicking of the animal brain, i.e. its neural network. This is where the A in the ANN comes from.

Machine learning through the use of ANNs have seen a wide set of applications, especially in recent years. Commonly employed applications of machine learning include spam filtering, speech recognition, machine translation, product recommendations and facial recognition.

2.1.1. Artificial Neural Networks

The main parts of an Artificial Neural Network are the nodes, layers, and its parameters. Figure 2.1 shows a simple figure of an ANN with three layers: one input layer, one hidden layer and one output layer. All layers that are not an input layer or an output layer are called hidden layers. The circles within each layer represent nodes (also called *artificial neurons*, *neurons*, or *perceptrons*), and each layer is often composed of several such neurons.

If each node in a network has a connection to all nodes in the next layer, the network is said to be “fully connected”.

2.1.2. Training and Gradient Descent

This subsection provides an overview of the mechanisms for training a neural network, in addition to introducing different types of training.

Forward Propagation

A neural network predicts its output by propagating a sample from the input layer through to the output layer. This propagation is represented as arrows in figure 2.1, and commonly referred to as *forward propagation* or *forward pass*. Each node calculates its value (also called activation) based on a set of values from previous nodes and parameters. This activation is then passed to the next layer of nodes.

While there are different types of neural networks, the function of a node is often just the weighted sum of the weights and biases of the nodes from the previous layer times the activations from the same nodes. This can be written as

$$f(x_1, x_2, ..., x_n) = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n, \tag{2.1}$$

or simplified to

$$\text{output} = \sum(\text{weights} \cdot \text{input}). \tag{2.2}$$

Here, $x_1, x_2, ..., x_n$ are the activations of the nodes from the previous layer (i.e. input values for the current node), and $w_1, w_2, ..., w_n$ are the weights for each corresponding activation. Each weight corresponds to a node from the previous layer and indicates how important that node is to the current node.

The activation of a node also often depends on an activation function employed in the same layer. The activation function determines the final activation of a node and introduces non-linearity to the network and the possibility for a network to represent any complex function. In its simplest binary form, an activation function decides if a node should fire (output 1), or not fire (output 0). The bias value of each node acts like a threshold for the activation function, and can be thought of as where the function intercepts the y-axis. By denoting the activation function by $g()$, the activation of each node can be expressed as:

$$y = g\left(\sum_{i=1}^{n}(w_i x_i) + b\right), \quad 1, 2, \ldots, n. \tag{2.3}$$

Common activation functions include softmax, sigmoid and ReLU. These are further explained in section 2.1.3.

When training a network from scratch, the values of the weights and the biases often start out as random values, which also results in the network predicting random values. It is through training that the network learns, e.g. how to predict correctly.

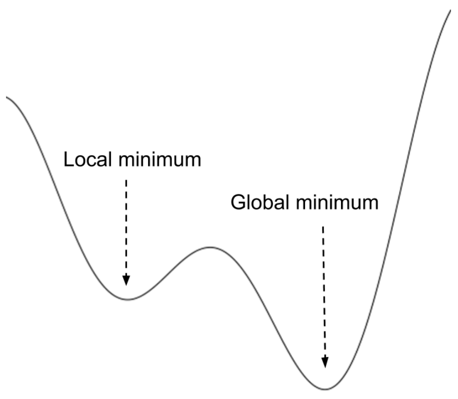


Figure 2.2.: Simple representation of a loss function with global and local minimums.

Gradient Descent and Backpropagation

One way of training a neural network is through supervised learning, where each input during training is supplemented with a value that contains the correct answer for what the output should be. After the output has been predicted, a loss function calculates how far from the ground truth, or how wrong, the predicted output was. This is also called the *loss* of the prediction. For a classification problem, a commonly used loss function is the Mean Square Error (MSE). The MSE takes the resulting value for each class in the output layer and subtracts it with its correct value, written as:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2, \tag{2.4}$$

where y and $\hat{y}$ are the true and predicted values, respectively.

This gives a loss vector that indicates how wrong the network was in classifying the input. The network then learns by propagating this loss vector back through the network, calculating and adjusting the parameters for each layer, called *backpropagation*. Backpropagation is calculated in a backwards fashion because of the dependence of later layers on previous layers. At each layer during backpropagation, the weights and biases are adjusted through an optimization algorithm called *gradient descent*. Gradient descent is an algorithm that calculates how the parameters of each layer should be adjusted to minimize the loss function of the network. It does so by finding the gradient of the loss function, and stepping in the direction of minimizing the loss with regards to the parameters in each layer. This might be a step in the right direction of the global minimum, although this is not certain.

To visualize gradient descent, we can look at figure 2.2. Imagine starting on the top of the hill on the left side. When moving down one small step at the time, and always stepping in the direction going, we find only the local minimum[1]. However, if we start on the right side, we will find the global minimum. This happens more or less by chance

[1]Given that the step size is small enough.

as the weights are initialized randomly. We will only find the direction in which to step to decrease the loss function value from the current point of view. Consequently, we have no means of knowing if that will actually find a global or a local minimum. All we know is that it is a step in a direction that is a little bit better than where we were, and so the parameters are updated accordingly. An important hyper-parameter of gradient descent is the step size, also called the learning rate. That is, how far we step each time the gradient is calculated. Different step sizes can have big impacts on the parameters of the network as too large steps can cause the algorithm to overshoot the optimum and "jump" back and forth. On the other hand, too small steps are almost certain to find only local minima. A common practice is to start with a large value and then decrease the step size as training progresses. Other optimization algorithms are also developed for aiding the network to converge (i.e. find a global or a good local minimum) and not become "stuck" at a sub-optimal local minimum.

Gradient descent is an iterative algorithm, and is applied over numerous iterations. The network learns through this process of repeatedly stepping in the downwards direction.

Types of Training

A neural network can be trained in a supervised, semi-supervised or unsupervised manner. For all types of training, the mechanism of gradient descent is the same. As previously seen, supervised learning is when the network is trained through labeled data, i.e. the network input is supplied with the right answer with which to compare its output or prediction. Unsupervised learning, however, deals with data that is not labeled and where the network has to discover information itself. Moreover, for semi-supervised learning, the training data typically contains a small set of labeled data, where the rest is unlabeled.

2.1.3. Activation Functions

This subsection introduces common activation functions that are significant to subsequent chapters. In these subsequent chapters, the activation functions are often referred to or depicted as separate layers. However, rather than layers containing their own nodes, they are functions that are applied to the output of a layer in the ANN, as shown in equation 2.3.

The Sigmoid Activation Function

The Sigmoid activation function is a collective name for a set of sigmoid functions that have a characteristic S-shape, such as the hyperbolic tangent. The standard Sigmoid function is the logistic function outputs values that lie between 0 and 1 and is denoted as

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

This Sigmoid function is a commonly applied activation function in classification tasks where more than one answer can be correct.

The Softmax Activation Function

The Softmax function can be thought of as an extension of the Sigmoid function in that it also output values between 0 and 1. However, the Softmax also constricts the output to follow a probability distribution, where all output values sum up to 1. The Softmax function is also called softargmax or a normalized exponential function, and is often employed for classification tasks where output values are mutually exclusive, i.e. only one answer is correct. The Softmax function is mathematically defined as

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{2.6}$$

for $i = 1, ..., K$.

Rectified Linear Unit Activation Function

The Rectified Linear Unit (ReLU) is a simple, yet effective, activation function that takes in the value of a node and outputs the max value of this value and 0. Consequently, no output from ReLU will be below 0. Mathematically, ReLU can be written as $y = \max(0, x)$. ReLU is commonly used especially in Convolutional Neural Networks (introduced in section 2.3), and is often a good first choice of an activation function.

2.1.4. Epochs, Batches and Iterations

When training an ANN, the data is often too big to handle all at once. Consequently, it is necessary to split up the dataset and load only small parts of it at a time. For this, we use terms like epochs, batches and iterations.

Epochs

One epoch is when all samples of a dataset are passed through the dataset once. It is common to use multiple epochs, i.e. passing over the entire training dataset multiple times. This is necessary since the gradient descent is iterative and passing over the training data only once is not enough to optimize the loss function. However, it is not always best to train for as many epochs as possible, as this can lead to overfitting to the dataset. This is where the network has learnt the dataset “too well”, i.e. performs very well on the dataset it has trained on, but performs poorly when presented with new training samples. This means that the network does not generalize well to other data it has not been trained on.

Batches

The size of a batch can vary between 1 and the size of the entire dataset, i.e. the size of an epoch. When the batch size is of less than the size of the entire dataset, i.e. mini-batches, the gradient descent algorithm is referred to stochastic gradient descent (SGD). However, when the batch size is of the size of the full dataset, the algorithm is referred to as just gradient descent. When employing SGD, the gradients are accumulated and averaged over all training samples in the mini-batch, before these averaged values are backpropagated through the network. Consequently, when employing gradient descent (i.e. batch size of the size of the dataset), the gradients are accumulated and averaged over the entire dataset before being backpropagated.

Iterations

One iteration is a pass over one batch. The number of iterations for one epoch is the amount of batches that fit in one epoch, and the number of iterations for the entire training is then how many batches there are in one epoch times the amount of epochs. As an example, say we have a dataset containing 256 training instances, and we have chosen a batch size of 32. Then we would have $256/32 = 8$ iterations for one epoch. If we then wanted to train for 100 epochs, we would have $8 \cdot 100 = 800$ iterations.

2.2. Statistics and Metrics

The following section introduces and explains statistical functions and metrics that are relevant for subsequent chapters.

2.2.1. Kullback–Leibler Divergence

The Kullback-Leibler (KL) divergence is a measure of how much two probability distributions differ. Specifically how much the second distribution differs from the first. Probability distributions in statistics are often approximated and simplified. However, through the KL divergence, it is possible to measure how much such a simplified distribution differs from the original, and thus how much information is lost.

Information is linked to the probability of a certain event within the probability distribution occurring, where an unlikely event provides more information than a likely event. Thus, the average of the information of all events of a probability distribution is the expected information from one single event in a probability distribution. This expected information is defined as the *entropy* of a probability distribution, denoted as H. Entropy can also be thought of as measuring the uncertainty of a probability distribution. The entropy can be used to measure how many bits are needed to encode the information in the probability, which is written as:

$$H = -\sum_{i=1}^{N} p(x_i) \cdot \log_2 p(x_i), \tag{2.7}$$

where p is the probability of a certain element x_i over the distribution of x. The binary logarithm is used here in the case of measuring *bits*. The formula for entropy is used as a basis for calculating the KL divergence between two distributions. More specifically, the entropy calculation serves as a measure of how many bits of information they differ by, i.e. how much information is lost. If we have the distribution p and the distribution q, we can write the KL divergence as

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log_2 p(x_i) - \log_2 q(x_i)), \tag{2.8}$$

or more commonly as

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot \left(\log_2 \frac{p(x_i)}{q(x_i)}\right). \tag{2.9}$$

While the KL divergence measures the information lost from one distribution to another, it does not measure the distance between the two distributions as the measure is not symmetric.

2.2.2. Cross-Entropy

While the KL divergence measures the information lost from one probability distribution to another (or their relative entropy), cross-entropy measures the total entropy between the two over the same set of events. Given two probability distributions p and q over a set of N events, we can write their cross entropy as

$$H(p,q) = -\sum_{i=1}^{N} p(x_i) \cdot \log_2 q(x_i). \tag{2.10}$$

Cross-entropy is commonly employed in machine learning as a loss function.

2.2.3. Inception Score

The Inception Score (IS) is a metric developed by Salimans et al. (2016) to approach an objective way of measuring the quality of generated images. Salimans et al. (2016) found that the IS correlated well with human evaluation of image quality, and it has therefore been widely adopted for evaluating images generated by Generative Adversarial Networks. In short, the IS measures both if each image depicts distinct objects and the variety of the generated images. The higher the score, the better the images generated (according to this score).

The IS is calculated through the use of a pre-trained version of the image classification Inception v3 model by Szegedy et al. (2016), from which the score takes its name. The model is employed over a large number of generated images to predict the likelihood of each image belonging to any of the 1,000 classes of the Inception v3 model. Consequently, this returns a probability distribution over all 1,000 classes for each image. The results from all generated images are summarized to create a marginal probability distribution from the whole set of generated images. Salimans et al. (2016) suggested using a set of 50,000 images to create this marginal probability distribution. The score subsequently combines the probability distribution of each generated image with the marginal probability distribution through the KL divergence. This means that if the two probability distributions, the class probability distribution of one image and the marginal probability distribution, differ significantly, we get a high score. Similarly, we get a low score if they are similar. An optimal score comes from when the marginal probability distribution is even (i.e. the probability of each class is fairly equal), while the probability distribution of one generated image is not (i.e. distinctly depicts a class).

Although the score is widely employed, Barratt and Sharma (2018) proposed five shortcomings of the score for use with generative models. These shortcomings include high scores on certain distributions like the uniform and the normal distribution and that slight variations in the classification distribution might yield dramatically different scores. Additionally, employing the IS on generative models trained on other datasets than the Inception v3 model (trained on the ImageNet dataset[2]), results in a discrepancy in the number and type of classes they contain. Barratt and Sharma (2018) thus argued that this can lead to misleading scores.

2.2.4. Fréchet Inception Distance

The Fréchet Inception Disctance (FID) is another metric for evaluating generated images, and was proposed by Heusel et al. (2017) as an improvement of the Inception Score. They argue that the IS lacks quality from not comparing the generated images to real images. The FID measures how much two groups of images differ where one set is the set of real images, and the other the set of generated images. Similarly to the IS, the FID employs the Inception v3 model. However, it discards the last classification layer, as it rather employs the different features of the images detected by the Inception v3 model. These features are calculated over the set of real and generated images, separately. For each set, the features calculated are summarized as a multivariate Gaussian distribution, with mean μ and covariance Σ. The FID is subsequently calculated from these two multivariate Gaussian distributions using the Fréchet distance (also called the Wasserstein-2 distance). This distance measures the similarity between two curves, which then can be said to measure the similarity of two image sets. Thus, a low score is better than a high score.

2.2.5. The Bilingual Evaluation Understudy

The Bilingual Evaluation Understudy (BLEU) is a machine translation metric proposed by Papineni et al. (2002). The metric compares a translated sentence to a reference sentence, where a perfect match yields a score of 1, and a complete mismatch yields a score of 0. The evaluation metric was proposed as an alternative to human evaluation and as a solution to the evaluation bottleneck that was prominent in the machine translation field.

The BLEU score is calculated through a modified n-gram precision score that matches n-grams of the two sentences to be compared. This modified n-gram precision score is found by finding the number of times an n-gram occurs in the test sentence and divides it by the number of n-grams in the reference sentence. This can be written as,

[2]http://www.image-net.org/

$$\text{modified n-gram precision} = \frac{\text{maximum number of times n-gram occurs in reference}}{\text{total number of n-grams in test sentence}}.$$

It is important to note that the maximum number of times the n-gram occurs in the reference is clipped to not be higher than the number of the same n-gram in the test sentence. Therefore, the modified n-gram precision never exceeds a value of 1. This modified n-gram precision is then calculated for all n-grams in the test sentence before it is averaged to produce the score for the test sentence.

1-gram, 2-gram, 3-gram, and 4-gram versions of the BLEU score are referred to as BLEU-1, BLEU-2, BLEU-3, and BLEU 4, respectively.

2.3. Types of Artificial Neural Networks

This section introduces a few types of artificial neural networks that are relevant for the rest of this Master's Thesis, including deep neural networks (DNNs), recurrent neural networks (RNNs), convolutional neural networks (CNNs) and residual neural networks (ResNets).

2.3.1. Fully Connected Neural Networks

In Fully Connected Neural Networks (FCNNs), all nodes in each layers are connected to all nodes in the next layer. The same term can be applied to the connection between two layers.

2.3.2. Feed-Forward Neural Networks

A Feed-Forward Neural Network is an ANN where there are no cycles, i.e. all input is only propagated forwards.

2.3.3. Deep Neural Networks

Deep Neural Networks differ from the ANN shown in figure 2.1 in that they contain more layers. That is, a DNN is deeper than the simple ANN with two or more hidden layers.

DNNs can be powerful if implemented correctly. However, as the network grows deeper, the problems of vanishing or exploding gradients become increasingly common. An exploding gradient problem is when accumulated gradients turn very large and produce major updates of the network parameters. This can result in unstable training and sometimes overflowing of the activations due to their size. At the other end of the spectrum, the vanishing gradient problem is when gradients become so small that they do not produce any visible network change, i.e. that the network does not learn. Neither of these phenomena is desired.

Because a DNN contains several layers, the gradients are backpropagated a long way to reach the start of the network. These gradients are calculated with respect to the

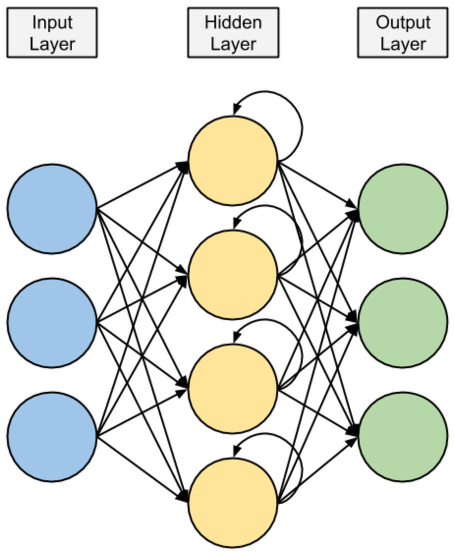


Figure 2.3.: A simple recurrent neural network: The arrows pointing back into the same node represent the recursion; output from a hidden layer coming in as input to the same layer.

weights of each layer, and consequently, the chain rule ensures that the gradients can easily vanish or explode. That is, when a network increases in size (i.e. grows deeper), if the weights are small (less than 1), they grow increasingly smaller (vanish) as they backpropagate through the network. Similarly, they grow increasingly bigger (explode) if their values are high (bigger than 1).

Exploding gradients can in general be avoided by careful configuration of the network. However, a common countermeasure for exploding gradients is the use of gradient clipping. This consists of either normalizing the gradient vector so that the magnitude of the vector equals a specified value, or setting a maximum value for each gradient.

There are several approaches or mechanisms to avoid the vanishing gradient problem. One such approach is the use of the Long Short-Term Memory architecture, which is discussed in further detail in section 2.5.3.

2.3.4. Recurrent Neural Networks

A recurrent neural network differs from other ANNs in the way that, for some nodes, the output comes back in as input (as shown in figure 2.3). Because the RNN can process its previous state along with the current state, it can process sequences of input, e.g. sentences. It is ideal for when a sequence in the input is more important than an individual item. Because of this, RNNs have been frequently used in the context of natural language processing (Mikolov et al., 2010).

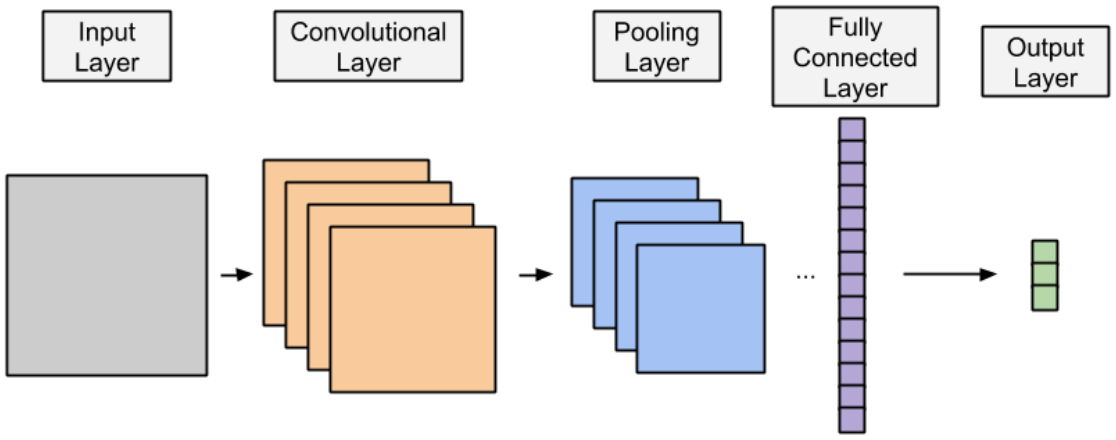


Figure 2.4.: A simple representation of a convolutional neural network with corresponding input matrix, convolution, pooling and classification.

2.3.5. Convolutional Neural Networks

A convolutional neural network (CNN) is a type of neural network that is especially good at learning features or patterns of its grid-like input such as time-series (1-D) and images (2-D). Consequently, this type of ANN is suitable for image processing. Image processing with CNNs is conducted through the use of matrix kernels, or filters. These filters each look at restricted and overlapping parts of the input matrix, starting in the top left corner and moving to the right. When a filter hits the end on the right, it starts on the next line on the left again. How much this filter should move each time is set through a hyper-parameter called the stride length. Moreover, CNNs often employ a pooling layer to reduce the spatial size of the previous layer. Such a pooling layer is often employed before a fully connected layer is added for classification. Figure 2.4 shows a high-level representation of an example CNN.

Deeper CNNs are made up of several filter layers, also called convolutional layers. Through the use of pooling layers between the convolutional layers, the separate convolutional layers learn different levels of features of the input, also called the feature hierarchy of a network. The early levels learn low-level features, while the subsequent layers learn higher-level features. Deeper CNNs are also called Deep Convolutional Neural Networks, or DCNNs. Generally, early convolutional layers learn low-level features, while later layers learn high-level features.

2.3.6. Residual Neural Networks

Although deeper neural networks can perform better than their shallower counterparts, they are more difficult to train and suffer from problems like vanishing and exploding gradients. He et al. (2016) showed that training a 20-layer CNN versus a 56-layer CNN yielded increased training error for the 56-layered version. To remedy this problem, He et al. (2016) proposed a solution by employing *shortcut connections* (also called *skip* *connections*) to feed-forward neural networks. Such a connection performs an identity mapping the output from an early layer to a subsequent layer. I.e. the skip connection skips one or several layers before adding this output to the output of the subsequent layer before its activation function is applied. As these skip connections perform an identity mapping, they do not introduce any new parameters to the network and can still be trained by normal gradient descent and backpropagation.

He et al. (2016) showed that their proposed technique is successful even when implemented on networks of more than 100 layers, and yield substantially better results than previous works. They showed state-of-the-art results on several dataset competitions, like the COCO segmentation and COCO detection tasks, employing their 152-layered residual net. However, the training degradation problem re-surges when the networks reach 1000 layers.

He et al. (2016) constructed several deep convolutional residual networks of several sizes, several of which can be downloaded as fully pre-trained networks. These networks include ResNet-50, ResNet-101 and ResNet-152, consisting of 50, 101 and 152 layers, respectively.

2.4. Generative Adversarial Networks

This section introduces the original architecture introduced by Goodfellow et al. (2014), in addition to a selection of extensions of the architecture, such as the conditional GAN and the CycleGAN architecture.

2.4.1. Generative Adversarial Network

Goodfellow et al. (2014) introduced the new framework Generative Adversarial Network (GAN) which employs an adversarial process where two models are pitted against each other. These two models are called the generator (G) and the discriminator (D), also shown in figure 2.5. The essence of the process between these two models is the generator creating samples from random noise and the discriminator evaluating if they are real or not. A common and much fitting analogy to this system is the work of a forger versus an expert. The forger tries to forge paintings by a particular artist, while the expert evaluates if the painting is real or not.

The discriminator is commonly a pre-trained classification network trained on the same data being employed in the process of training the GAN. That is, it could be trained on the paintings of a particular painter to assess whether other paintings are of the same artist. The generator, however, is not pre-trained and generates “forged paintings” from random noise. These paintings are then supplied to the discriminator, which assesses if the painting is real or not, before providing feedback to the generator. In the beginning, the generator naturally performs poorly. However, as it receives feedback it becomes increasingly better at creating forged paintings. Consequently, the discriminator becomes increasingly “confused” and might start to classify the forged paintings as real. The goal of the generator is for the discriminator to output the labels “real” or “fake” with the

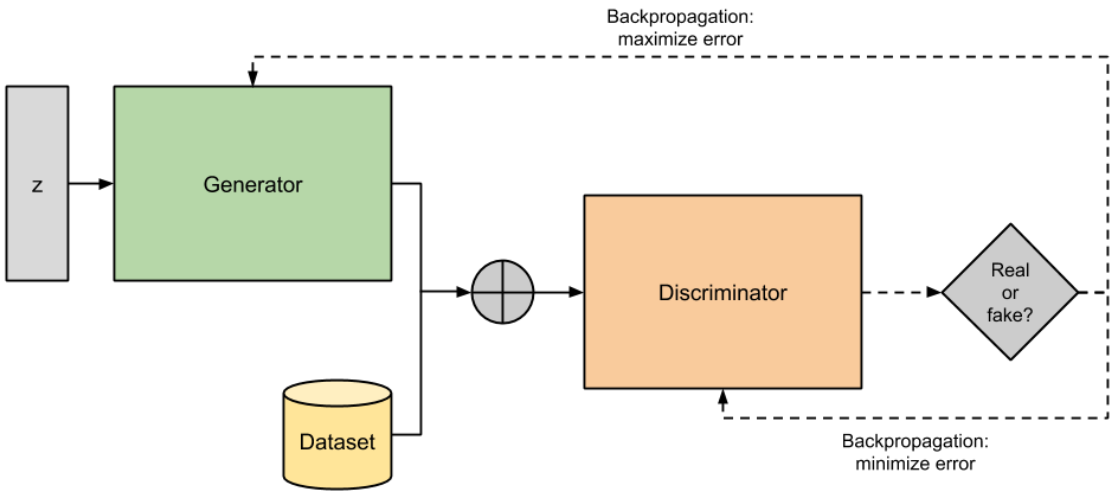


Figure 2.5.: A simple overview of a Generative Adversarial Network. The generator receives noise input $\boldsymbol{z}$, from which it generates the example for the discriminator to assess. The discriminator receives both the training dataset and the new generated example as input and tries to determine if the new example could belong to the dataset or not. The result is backpropagated through both networks where they try to minimize their respective loss functions, i.e. minimize and maximize the discriminator's error.

same probability. That is, the expert does not know whether the provided painting is real or not, and therefore only "guesses" its answer. The network is said to converge should it reach such a state. An overview of the network structure and inputs and outputs is shown in figure 2.5.

The discriminator's objective is to maximize the probability of appointing the correct label to the example and can be written as $\log(D(\boldsymbol{x}))$, where $\boldsymbol{x}$ is the example to be assessed and $D(x)$ is the probability that $\boldsymbol{x}$ is real (i.e. comes from the provided data). On the other hand, the generator's objective is to minimize the probability of the discriminator appointing the correct label, and can thus be written as $\log(1 - D(G(\boldsymbol{z})))$. $\boldsymbol{z}$ is here random noise input and $G(z)$ is the example generated by the generator.

The system described is essentially a two-player minmax adversarial game with its value function

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{2.11}$$

as expressed in Goodfellow et al. (2014). p_{data} and p_z are distributions over the supplied data and the noise, respectively.

The GAN architecture is an example of unsupervised learning, and was a breakthrough in computer vision for generating realistic-looking samples.

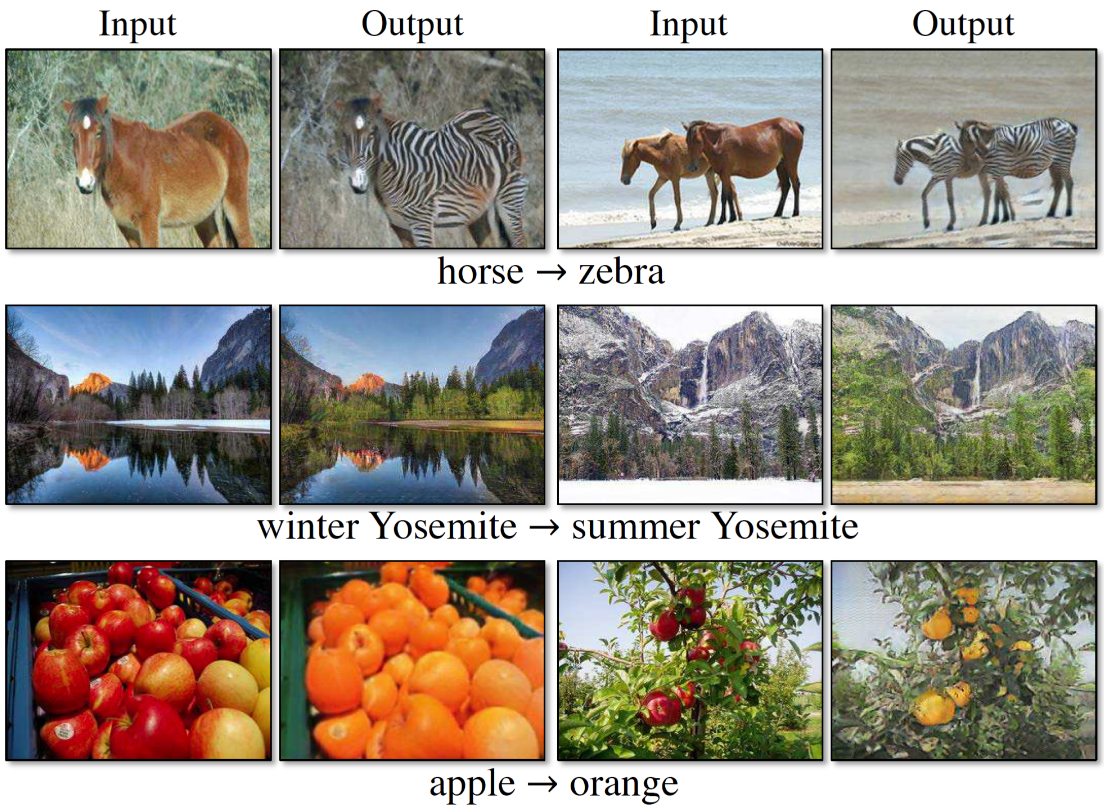


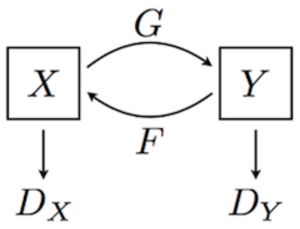
Figure 2.6.: CycleGAN example translations from horse to zebra, winter to summer, and apples to oranges. Figure from Zhu et al. (2017), with permission from Taesung Park.

2.4.2. Conditional Generative Adversarial Networks

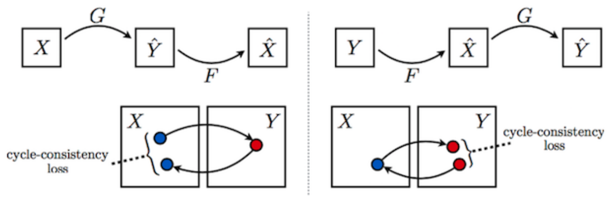
Mirza and Osindero (2014) proposed an extension to the original GAN architecture by introducing conditioning for directing the data generation process, a conditional GAN (CGAN). That is, introducing the possibility of supplying extra information to direct the generation to a specific item, e.g. an image of a dog. The conditioning is performed through supplying the extra information to both the generator and the discriminator. Consequently, the value function of the CGAN differs from the original value function by calculating the objectives of the two networks given the extra information $\boldsymbol{y}$. As stated in Mirza and Osindero (2014), the value function thus becomes,

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log(D(\boldsymbol{x}|\boldsymbol{y}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{z}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \quad (2.12)$$

The CGAN plays a major part in T2I synthesis using GANs, as the generated images are conditioned on the input text.



(a) Simple CycleGAN architecture, where G and F are separate generators, and X and Y are the two different domains.



(b) CycleGAN cycle-consistency loss where a sample is translated from the first domain to the second, and then back to the first, before the loss is calculated.

Figure 2.7.: Figures from Zhu et al. (2017), with permission from Taesung Park.

2.4.3. CycleGAN

In 2017, Zhu et al. introduced an image-to-image translation model called CycleGAN. This model was inspired by the need for image-to-image translation training without paired image examples (i.e. unpaired image-to-image translation). Image translation is, in essence, a controlled image modification. A classic example of this is to translate an image of a horse to an image of a zebra, like the examples shown in figure 2.6. The need for unpaired image-to-image translation was based on the difficulty of acquiring paired image examples from different domains for training. This could be the task of translating a photograph (domain X) into the style of the deceased painter (domain Y), where there would be very few paired examples.

CycleGAN translates from the first domain, X, to the second domain, Y, before translating back, from Y to X. To do this, it extends the traditional GAN architecture by training two GANs simultaneously. The two GANs evaluate their own part with each GAN's discriminator having access to a set of images from their respective domain. The generator for the same GAN is conditioned with an image that belongs to the other domain. As seen in figure 2.7a, the first GAN's generator takes an image from domain X and its discriminator evaluates it on the basis of domain Y. The second GAN's generator takes an image from domain Y and its discriminator evaluates it on the basis of domain X.

Additionally, the CycleGAN architecture also employs an extension called *cycle consistency*, illustrated in figure 2.7b. As the name implies, CycleGAN uses the output of the first generator (translating from X to Y) as input to the second generator (translating from Y back to X) before evaluating how well the cycled image matches the original image. Mathematically, we have the first generator $G : X \to Y$ and the second generator $F : Y \to X$. If the cycle consistency is fulfilled, we can say that G and F are inverses of each other and therefore should "cancel" each other out. This is the same as $F(G(x)) \approx x$ and $G(F(y)) \approx y$.

This cycle consistency makes sure that the translated images are actually translations

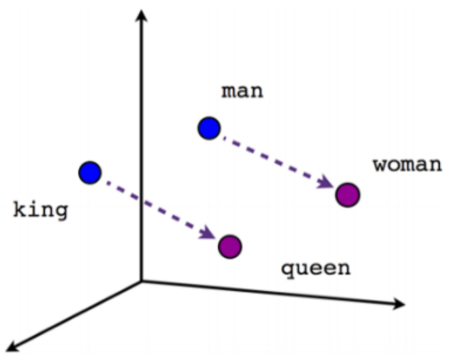


Figure 2.8.: Word embeddings example showing the relations between man, woman king and queen in vector space. Figure from "Glossary of Deep Learning: Word Embedding"[3], licensed under CC BY-SA 4.0.

of the input image, and not just new generations of images that fit the domain the discriminator wants.

2.5. Natural Language Processing

Natural Language Processing (NLP) is a sub-field of artificial intelligence concerned with the processing and analysis of natural language data, i.e. for the computer to understand and employ natural language. It is an important part of machine learning and has applications like voice assistance, translation, spam filtering and is used for chatbots.

Only a few of the concepts presented in this section are used in subsequent chapters. However, the remaining concepts are presented here to provide a background for understanding the more advanced concepts.

Language modeling is a central part of many NLP tasks. A language model aims to model the connections between words in natural language through predicting the next word, given a sequence of previous words. A neural language model uses a simple neural network which maps each word in its vocabulary to real-valued vectors. The network learns through a joint probability function of the word embeddings of the words in the input sequence, i.e. it maps the vectors of the words to a probability distribution.

2.5.1. Neural Language Modeling

Word embeddings is an example term of a language model that map words to vector embeddings in vector space. They have shown to exhibit properties of mapping words that have similar meaning to similar values. Additionally, the vectors allow for operations like subtraction and addition to arrive at new values that correspond well with natural language logic. A classic example of this is $king - man + woman = queen$, where the

[3] `https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca`

words correspond to their respective values in vector space. The relations between the words in this example are illustrated in figure 2.8.

2.5.2. Wordpiece Tokenization

Segmentation of the input sequence into meaningful tokens or words is a fundamental part of NLP. This process facilitates the creation of meaningful embeddings of these tokens or words. A common approach is to segment based on the words in the input sequence. Although this is intuitive, the approach has shown to not handle a rare or unseen word well. On the other hand, segmentation based on characters overcomes this limitation, but is too fine-grained and thus misses a lot of important information of the input sequence.

WordPiece tokenization is a combination of the two previously mentioned approaches where the input sequence is divided into smaller sub-word tokens or *wordpieces*. An example of this is the word "strawberries" being segmented into two tokens, namely "straw" and "berries". These wordpieces can subsequently be mapped to real-valued vectors, which together provide the meaning of the whole word. The technique was first introduced by Schuster and Nakajima (2012) where it was employed for Japanese and Korean voice search, and later applied to Google's Neural Machine Translation system by Wu et al. (2016). Amongst other qualities, the technique increases the robustness of NLP models with regards to handling of rare words.

2.5.3. Long Short-Term Memory

After segmenting the input sequence, NLP techniques commonly employ RNNs for processing of input sequences. A prominent problem and a bottleneck of the field has been the difficulty of dealing with longer input sequences, as the models tend to forget prior parts as the sequence grows longer. This is because RNNs have a tendency of suffering from vanishing gradients. This tendency is due to that RNNs are often trained through the use of backpropagation through the input sequence (also referred to as time, or steps). This means that when backpropagation is executed, the network is unrolled through time and essentially becomes a deep neural network. It thus shares the same property of the vanishing gradient problem as a DNN. Consequently, this problem of dealing with long-term relations may also lead to loss of important information.

To remedy the problem of vanishing gradients in RNNs, Hochreiter and Schmidhuber (1997) proposed a new RNN architecture called Long Short-Term Memory (LSTM). The LSTM employs an architecture with a fitting gradient-based learning algorithm. An LSTM picks out the most important information from the input sequence and discards unimportant information. This reduces computational complexity and computing time, and allows for long-term dependencies within the sequences.

In the LSTM network, the cells contain their own cell *states* and *gating mechanisms*. The cell states contain information from previous parts of the input sequence (also called hidden state), while the gates are there to decide what information to pass on and which information to discard. The gates are themselves neural networks that learn what is

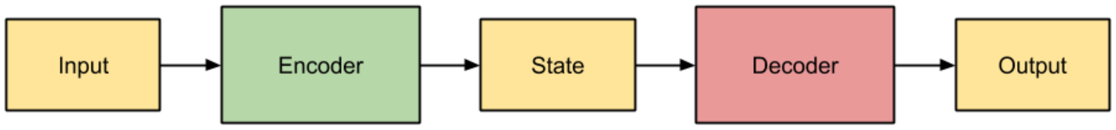


Figure 2.9.: A simple encoder-decoder architecture.

important for their specific task, such as forgetting. These gates include the *input*, *forget* and *output* gates. While the forget gate decides what information to discard (or forget), the input gate decides what new information to keep. Moreover, the output gate decides what information to send as the hidden state for the next part of the sequence.

2.5.4. Encoder-Decoder Architectures

The encoder-decoder is a commonly employed architecture in NLP models. As illustrated in figure 2.9 the architecture is divided into two parts, an encoder and a decoder. The encoder takes an input and encodes it to a state that is subsequently sent to the decoder. The decoder thus generates the output based on this encoded input. The architecture is commonly employed in neural machine translation (NMT) where the encoder encodes the input sequence of the first language to its corresponding sequence embeddings, while the decoder generates the translation based on these values. The encoder-decoder architecture was constructed to solve the problem where existing architecture (employing RNNs) in the field of NLP could only be applied where the inputs and outputs could be computed with vectors of a fixed size of dimensionality (Sutskever et al., 2014).

Sequence to Sequence (Seq2Seq) modeling, is an example of an encoder-decoder architecture, where the model maps input sequences to output sequences. It has a number of applications from speech recognition and machine translation to image captioning. The main idea behind the Seq2Seq architecture is to use one LSTM to encode the input sequence, and another to decode the output of the first to find the output sequence (Sutskever et al., 2014).

2.5.5. Attention

Although LSTMs provided an enhanced version of RNNs for longer sequence inputs, the challenge of dealing with longer sequences was still prominent. In the case of Seq2Seq, this was mainly due to the fixed-size context vector employed between the encoder and the decoder. Bahdanau et al. (2014) and Luong et al. (2015) proposed an attention technique that mitigated this problem. Before attention, an encoder-decoder architecture passed only the last hidden state (i.e context vector) of the encoder to the decoder. With attention, however, all of the hidden states of the encoder are passed to the decoder. This technique allows for the decoder to "pay attention to" the most important part of the input sequence at each time step. A simple overview of a translation of the sentence "I am a student" from French to English using attention is illustrated in figure 2.10.

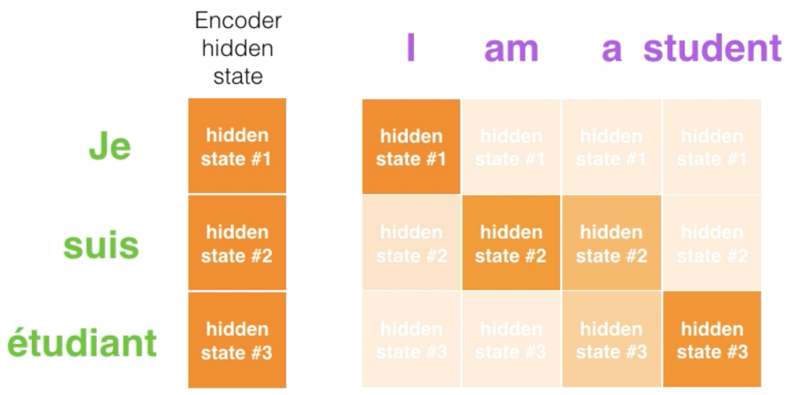


Figure 2.10.: Example of attention used for translation from French to English. Figure by Jay Alammar[4], licensed under CC BY-NC-SA 4.0.

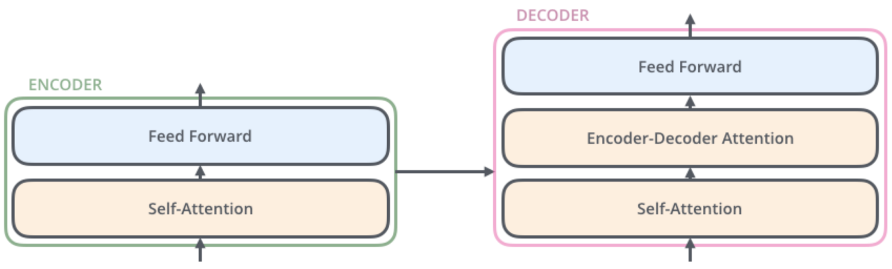


Figure 2.11.: Transformer encoder and decoder. Figure by Jay Alammar[5], licenced under CC BY-NC-SA 4.0.

2.5.6. Transformers

Like Seq2Seq, the Transformer is based on the encoder-decoder architecture. However, while the encoder of the Seq2Seq model employs an RNNs, the Transformer encoder is built up of a fully-connected feed-forward neural network and self-attention (explained later in this section). Similarly, the Transformer decoder also avoids RNNs by employing the same components as the Transformer encoder, in addition to a layer of encoder-decoder attention. This encoder-decoder attention performs attention over the output of the encoder. An overview of this can be seen in figure 2.11.

The Transformer contains one stack of encoders (of size 6 in Vaswani et al. (2017)), and one stack of decoders. The input sequence is fed into the first of the encoders in the encoder stack, which output serves as the input for the next encoder, and so on. The output of the last encoder in the stack serves as input to the stack of decoders. Here, the encoder output is sent to all decoders in the stack. The first decoder in the stack takes the input from the encoder and produces the input for the next decoder in the stack,

[4] https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
[5] http://jalammar.github.io/illustrated-transformer/

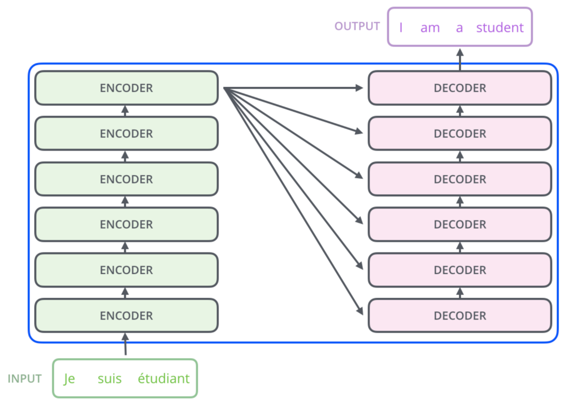


Figure 2.12.: Transformer architecture. Figure from Jay Alammar[6], licenced under CC BY-NC-SA 4.0.

while the next decoder is provided with the output from the previous in addition to the output of the encoder stack. This architecture is illustrated in figure 2.12.

With the encoder and the decoder parts of the Transformer being divided into subparts, the model lends itself to parallelization. This is a significant improvement from the previous state-of-the-art models employing RNNs, which were inherently serial or sequential.

Self-Attention

Attention can be used to find similarities and correlations between two sentences, but it can also be used to find correlations or other important words in the same sentence. When the model pays attention to the input sequence itself and computes a representation of that same sequence, it is called self-attention. Self-attention is also known as intra-attention.

Calculation of self-attention starts with creating three vectors from each of the word embeddings: a query ($\mathbf{q}$), key ($\mathbf{k}$) and value ($\mathbf{v}$) vector. Then, for each word embedding, the dot-product is calculated between the query vector and the key vector of every word in the input. This means that for the word in position 1, the dot product is calculated between $\boldsymbol{q_1}$ and $\boldsymbol{k_1}$, $\boldsymbol{k_2}$, $\boldsymbol{k_3}$, and so on. These dot products make up the dot-product scores used for later calculations. Figure 2.13 illustrates this self-attention. The dot-product scores are here the values corresponding to the label "score".

After the dot-product scores are computed, all scores are divided by $\sqrt{d_k}$, where d_k is the dimension of the queries and keys. It is called a scaled dot-product attention because of this scaling factor. The scaling factor is employed to scale down the values where the dot products grow so large that the softmax function outputs values that result in vanishing gradients of the network. The softmax function is then applied to produce the weights of the values, before multiplying this score with the value vector of the word embedding. Calculating the softmax of the dot-product score outputs a value between 0

[6]http://jalammar.github.io/illustrated-transformer/

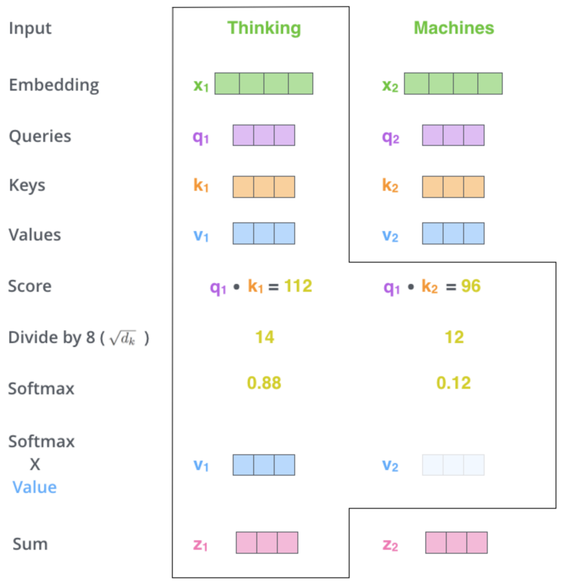


Figure 2.13.: Transformer self-attention example illustration. Figure from Jay Alammar[7], licenced under CC BY-NC-SA 4.0.

and ’, where a value closer to 0 signifies an unimportant word with regards to another, and a value close to 1 signifies an important word. Here, unimportant and important refer to the correspondence or match between two words, i.e. the word “it” corresponding considerably to the word “child” in the sentence “The child was hyperactive because it had just eaten too much sugar.”

All queries and corresponding key-value pairs are packed in matrices with the queries, keys and values denoted with $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ respectively. Scaled dot-product attention is thus performed on the full matrices of the input sequence.

The Scaled Dot-Product Attention function can then be written as

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}})\boldsymbol{V}, \tag{2.13}$$

where its result is a matrix, denoted with $\boldsymbol{Z}$.

Multi-Head Attention

In the architecture by Vaswani et al. (2017), the Scaled Dot-Product Attention is linearly projected h times with different weight matrices, called Multi-Head Attention. This results in h different, $\boldsymbol{Z}_i$, one for each head i. These resulting matrices are then concatenated before multiplied with an additional weight matrix $\boldsymbol{W}^O$, which results in a resulting matrix $\boldsymbol{Z}$ which thus contains a vector for each output word.

[7] http://jalammar.github.io/illustrated-transformer/

Positional Encoding

The Transformer model also incorporates positional awareness by including a positional encoding with the input embedding. This positional encoding includes information about the relative or the absolute position of the tokens in the sequence. In Vaswani et al. (2017) the model employed sine and cosine functions for the positional encoding because it, amongst other reasons, gives the advantage that it can scale to unknown lengths of sequences.

2.5.7. Bidirectional Encoder Representation from Transformers

A classic approach to language modelling is to predict the next word in a sequence based on previous words. This approach is only unidirectional (i.e. only sees backwards) and limits the information the model bases its prediction on. An enhancement to such an approach is to make the model bidirectional, i.e. predict a word based on both previous and subsequent words. The Bidirectional Encoder Representation from Transformers (BERT) is an attention-based language model built on the Transformers architecture, and is the first unsupervised language representation model that is deeply bidirectional. The model encodes sequences into meaningful representations where attention is paid between all words in the sequence, relatively. BERT tackles bidirectionality by employing an unsupervised masked language modeling (MLM) pre-training task, while also employing next sentence prediction task (NSP) for prediction of which of two sequences is subsequent to the other.

BERT obtained state-of-the-art performance on eleven NLP tasks like The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), The Multi-Genre Natural Language Inference (MultiNLI) corpus (Williams et al., 2017), and The Stanford Question Answering Dataset (SQuAD) v1.0 (Rajpurkar et al., 2016) and v2.0. The GLUE benchmark is a collection of natural language understanding tasks, the MultiNLI is a collection of sentence-pairs where the task is to label a hypothesis sentence given a premise, and the SQuAD v1.0 is a question answering task where the answer to the posed question might or might not exist in the presented text. SQuAD v2.0 extends v1.0 in the way that the answer might exist in the given text, but not explicitly stated.

Soon after Devlin et al. (2018) published their paper introducing BERT, the code to the model was open-sourced. Additionally a set of BERT models pre-trained on massive datasets were also released. Thus, BERT was marked as the beginning of a new era in NLP, where anyone without expertise in the field could effortlessly download and employ these powerful language models.

Architecture

Devlin et al. (2018) employed two versions of different sizes of the model: $BERT_{BASE}$ and $BERT_{LARGE}$.

[8] http://jalammar.github.io/illustrated-bert/

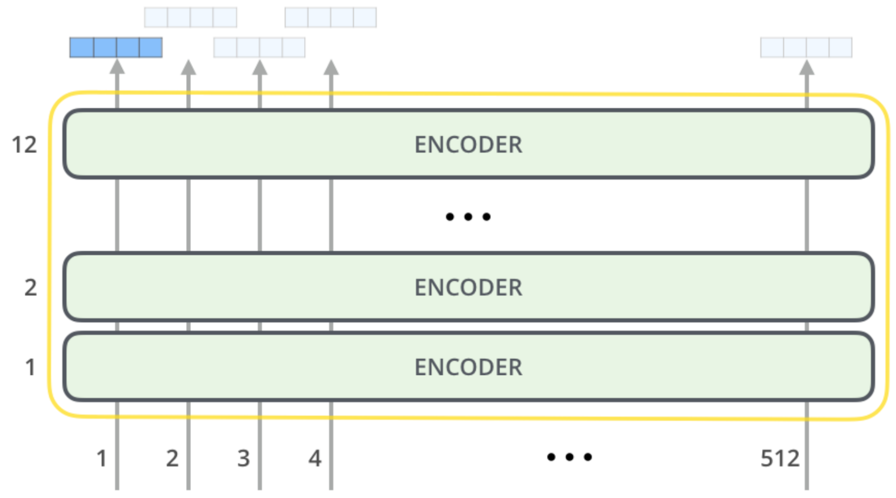


Figure 2.14.: BERT architecture for $BERT_{BASE}$. The encoder stack is made up of 12 encoders and each position of the input outputs a vector of the hidden size of the encoders. Figure from Jay Alammar[8], licenced under CC BY-NC-SA 4.0.

Although BERT is based on the Transformer architecture, the model only employs the Transformer encoder. The encoders are stacked in the same manner as in the Transformer, but the stacks are of size 12 and 16 for $BERT_{BASE}$ and $BERT_{LARGE}$, respectively. Each encoder layer (also called Transformer block) applies self-attention and passes the results to its feed-forward network before sending the output to the next Transformer block. In the last Transformer block, each position of the sequence outputs a vector of the same size as the feed-forward neural networks in each layer. A simple illustration of the BERT architecture is shown in figure 2.14.

$BERT_{BASE}$ has a total of 12 encoder layers, 12 attention heads per encoder, and 768 hidden units in its feed-forward network (i.e. hidden size). Consequently, the total number of parameters is 110 million. On the other hand, the larger model, $BERT_{LARGE}$, consists of 24 layers, 16 attention heads per encoder, and a hidden size of 1024. Thus, the number of parameters for $BERT_{LARGE}$ is 340 million.

BERT can receive both a single, or a pair of input sequences. Additionally, to improve the handling of rare or new words, BERT employs WordPiece embedding. Each embedding is extended to include a position embedding for where in the sequence the embedding is, and a segment embedding that signifies whether the token belongs to the first or the second input sequence. This input representation can be seen in figure 2.15.

Pre-training

The BERT model is pre-trained on two unsupervised tasks: a Masked Language Model (MLM) pre-training objective and Next Sentence Prediction (NSP). In MLM, random WordPiece tokens are masked out (i.e. switched out) with the `[MASK]` token. The task of

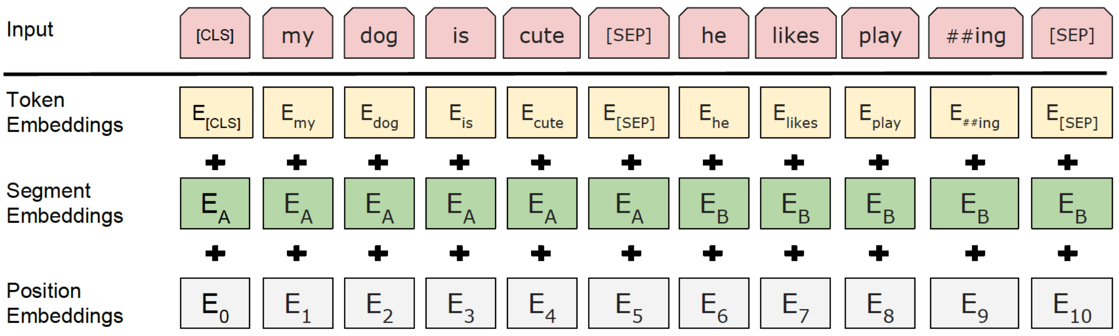

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2.15.: BERT input representation. The inputs are composed of position embeddings, segment embeddings and token embeddings. Figure from Devlin et al. (2018), with permission from Jacob Devlin.

the model is then to predict this masked word. This is different from other pre-training tasks like next word prediction in that the model looks at both left and right context of the word, and thus allows for a bidirectional pre-trained model.

In practice, 15% of all WordPiece tokens are masked out during pre-training. However, because the token `[MASK]` does not exist during fine-tuning, there is a mismatch between pre-training and fine-tuning. To mitigate this, the masked words were also replaced with other words in the vocabulary (instead of the `[MASK]` token), or left unchanged. Thus, for the 15% of WordPiece tokens to be masked out:

- 80% were replaced with the `[MASK]` token
- 10% were replaced with another word in the vocabulary
- 10% remained unchanged.

For the second pre-training task of BERT, Next-Sentence Prediction (NSP), the model receives two input sequences, A and B, and predicts if B is a subsequent sequence of A or not. 50% of the time, B is the successor, and 50% of the time it is not. The NSP pre-training task was added to the model for enhancing its capabilities of handling the relationship between multiple sentences.

Applications of BERT

BERT can be used for several tasks by employing a simple feed-forward network to the model and fine-tuning this for the specific task at hand. Applications for BERT include sentence classification for predicting which class a sentence belongs to (e.g. "spam" or "not spam"), question answering as in SQuAD v1.0 and v2.0, and named entity recognition (NER) for marking the entities within a sentence (e.g. person, date, action, etc.).

Moreover, BERT is not a natural choice for generating text due to its bidirectional nature. However, this may be overcome if employed in conjunction with e.g. an LSTM, as is explored in subsequent chapters.

2.6. Automatic Generation of Image Captions

Automatic generation of image captions, called image captioning, is closely tied to T2I synthesis in that they are, in essence, reverse processes. Moreover, it is especially relevant for this work due to MirrorGAN employing re-generation of the image caption as an extra objective function.

Throughout recent years, there has been multiple approaches to image captioning, ranging from hand-crafted encodings through filling in appropriate words in an existing template, to the generation of novel captions through employing deep neural networks. As approaches such as the first are specific to certain types of tasks and can only create a fixed-length caption, they are unsuitable for a more wide and diverse set of images. On the other hand, deep learning-based techniques can generate novel captions which also vary in length. Consequently, automatic generation of novel image captions through employing deep neural networks has gained popularity as the field has progressed and significantly surpassed the first type of approach. Moreover, image captioning through deep learning connects the field of computer vision and natural language processing through encoding visual features present in the input image, before translating these features into a semantic representation

A typical architecture for deep learning image captioning is based on the encoder-decoder structure discussed in section 2.5.4. As the process of image captioning can be thought of as "translating" from image to text, it can be considered closely related to neural machine translation, which task also often employs this architecture. This encoder-decoder architecture typically uses a CNN for extracting features from the image, and an RNN to generate the semantic caption. The major challenges of image captioning include finding the central parts of an image, representing how these objects relate to each other in context through natural language, and generating longer sequences (i.e. sentences).

Xu et al. (2015) proposed an approach to image captioning through the use of deep learning, encoder-decoder architecture, and employing an attention mechanism to choose which area of the image to focus on at each time step. Their approach directly extends the proposed attention mechanism by Bahdanau et al. (2014).

In the encoder, different features of the input image are extracted from an early convolutional layer of a pre-trained CNN in order to keep a high dimension of features. These extracted features are vectors denoted as

$$a = a_1, ..., a_L, \quad a_i \in \mathbb{R}^D, \tag{2.14}$$

where i is a particular area of the input image. Subsequently, the decoder is an LSTM with attention, which, for each time step, calculates the next output word of the output image caption. The attention, or "weight" of each annotation vector, is calculated through a multi-layer perceptron (i.e. a network with multiple layers) conditioned on the previous hidden state of the LSTM for each time step. Then softmax is applied over the weights to create a probability distribution where all weights sum to 1. The resulting values are denoted as

$$\alpha_{ti}, \quad i = 1, ..., L, \tag{2.15}$$

where t is the current time step, and i denotes the corresponding annotation vector.

The authors thus propose two ways to use these attention vectors; through a stochastic "hard" attention, and a deterministic "soft" attention. The soft attention is based on the work of Bahdanau et al. (2014) and is essentially the weighted sum of all annotation vectors a_i of the input image, i.e. $\sum_{i=1}^{L} a_i \alpha_i$ for each time step, i. However, the hard attention is different in that it provides a new feature at each time step, from which to generate a corresponding word. These features are "randomly" chosen through sampling their multinoulli (categorical) distribution. However, sampling within a neural network prevents backpropagation. Simply put, in order for the gradient to be calculated for a specific node, its function must be differentiable. This is not the case for randomly chosen features. Consequently, the re-parameterization trick is applied, where the sampling, or randomness, is separated out into it's own node. Thus, the network learns a differentiable function which receives the random variable as input, and backpropagation can then be applied. Xu et al. (2015) proposed this to be a function of the features, a_i, and their locations, s_i, which are one-hot encoded vectors with the value 1 only for the current location. This function then maximizes the probability of an image caption $\boldsymbol{y}$.

Xu et al. (2015) employed their trained models (with hard and soft attention, respectively) on several datasets, including the COCO dataset. The results were then evaluated through both the BLEU and the METEOR (Denkowski and Lavie, 2014) metrics. Xu et al. (2015) reported that both of their models outperformed previous image captioning models on all datasets and using both metrics. Generally, the model trained using the stochastic "hard" attention outperformed the "soft" version when evaluated through the BLEU metric, while the "soft" model performed better when evaluated through the METEOR metric.

3. Related Work

This chapter presents key related work and the datasets they employ for training and validation. As the main subject revolves around Text-to-Image synthesis based on the architecture of Generative Adversarial Networks introduced by Goodfellow et al. (2014), this chapter includes an overview of the main recent advances of T2I synthesis based on this architecture. These advances or models are presented in chronological order, where each model is based on the previous. Section 3.4 and 3.5 are based on the preliminary studies conducted during the Fall of 2019, and are covered in greater detail than the previous sections due to their importance for subsequent chapters. Additionally, section 3.1 provides an introduction to the three main datasets that are employed in the training of the models described in this chapter.

3.1. Datasets

This section introduces the different object recognition datasets employed in the related work explained in the rest of this chapter. The three datasets focus on different aspects of object recognition; like image classification using labels, object localization using bounding boxes and semantic segmentation in pixel-level.

3.1.1. The Oxford Flowers 102 Dataset

The Oxford Flowers 102 (Oxford-102) dataset provides a large number of classes within the domain of flowers (Nilsback and Zisserman, 2008). The dataset contains 102 classes of flowers commonly found in the United Kingdom. With each class consisting of between 40 and 258 images, the dataset contains in total 8,189 images. These images are separated into 1,020 training images, 1,020 validation images and 6,149 test images.

Each image depicts either a single flower, or a small number of the same type of flower. For each image, four different features or aspects of the flowers are also computed. These features are: local shape/texture, shape of the boundary, overall spatial distribution of petals and color. Examples of such images can be seen in figure 3.1a.

3.1.2. The Caltech-UCSD Birds-200-2011 Dataset

The Caltech-UCSD Birds-200-2011 (CUB Birds-200) dataset contains images and image captions of 200 bird species. The dataset contains 8,855 training and 2,933 test images, and there are 10 text descriptions per image (Wah et al., 2011). The images are all annotated with bounding boxes (i.e. boxes showing where the bird is located in the

(a) Oxford-102 Flowers dataset examples. Examples from Nilsback and Zisserman (2008), with permission from Maria-Elena Nilsback.

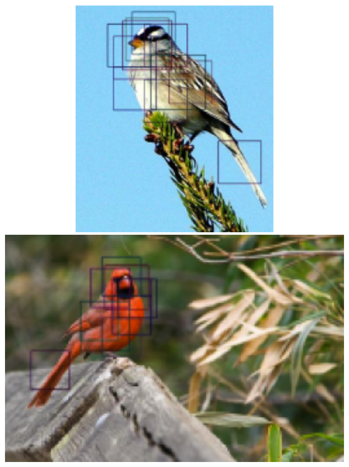

(b) Caltech-UCSD Birds-200-2011 dataset examples. Examples from Wah et al. (2011), with permission from Catherine Wah.

Figure 3.1.: Examples from the Oxford-102 and CUB Birds-200 datasets.

image), part locations and attribute labels. Examples from the dataset can be seen in figure 3.1b.

3.1.3. The Microsoft Common Objects in Context Dataset

The Microsoft Common Objects in COntext (COCO) dataset contains 82,783 training images and 40,504 validation images, and there are five text descriptions belonging to each image (Lin et al., 2014). In contrast to the two datasets previously introduced, the COCO dataset contains images depicting a wide range of categories, or objects. Instead of only depicting one or more of the same type of objects in one image, the COCO dataset contains images that show several objects at a time, i.e. showing objects in context as shown in figure 3.2.

Through these attributes, the COCO dataset is the most diverse of the three datasets presented. Moreover, the images in the dataset are sampled from the image and video hosting service, Flickr[1]. Consequently, the creators of the dataset do not own the rights to these images, although some are licensed under Attribution 4.0 International (CC BY 4.0), like the images in figure 3.2.

3.2. Generative Adversarial Network for Text-to-Image Synthesis

Reed et al. (2016) were the first to move away from using properties of attribute representations for generating images with their “Generative Adversarial Network for Text-to-Image Synthesis”. Instead, they employed the power of natural language processing

[1]https://www.flickr.com/
[2]https://www.tensorflow.org/datasets/catalog/coco

Figure 3.2.: Examples from the COCO dataset, taken from the TensorFlow website[2]. The images are licensed under CC BY 4.0.

(NLP) for T2I generation through the use of the GAN architecture introduced by Goodfellow et al. (2014). This was based on the understanding that natural language provides a flexible interface for describing visual categories. As an extension to the original GAN, they employed a version of a conditioned deep convolutional GAN (DCGAN). The DCGAN was proposed by Radford et al. (2015) and is an improvement to the original GAN in that it employs deep convolutional networks and batch normalization for normalization and scaling of feature detection. Both the generator and the discriminator of the T2I synthesis system proposed by Reed et al. (2016) are conditioned on learned feature representations, encoded by deep convolutional and recurrent text encoders.

Reed et al. (2016) argued that the most straightforward way to evaluate the generated images and corresponding text is to take in such pairs and judge if their combination is real or fake. However, this is a naïve approach in that the discriminator has no way of knowing if real images match the paired text or not. The approach implicitly contains two sources of error where one is unrealistic images and the other is a mismatch between the image and the text. With this in mind, Reed et al. (2016) wanted to separate these error sources to make them more explicit, and thus introduced a new set of training pairs to feed to the discriminator: real images with mismatched text. This, they argued, would optimize the matching of images and text, in addition to the image realism loss, and thus provide the generator with an additional learning signal.

This proposed architecture was trained on the CUB Birds-200 and the Oxford-102 datasets of flower images. Both datasets had five captions per image and the training images were of size 64 x 64 pixels. The trained model provided plausible examples when tested on descriptions of birds or flowers. When trained on a more diverse dataset, like the COCO dataset, the model also generated images of low resolution (64 x 64 pixels) and was not able to generate important details and the hierarchy of the objects in the image. Example results of generated images on the Oxford-102 and CUB Birds-200 datasets can be seen in figure 3.3.

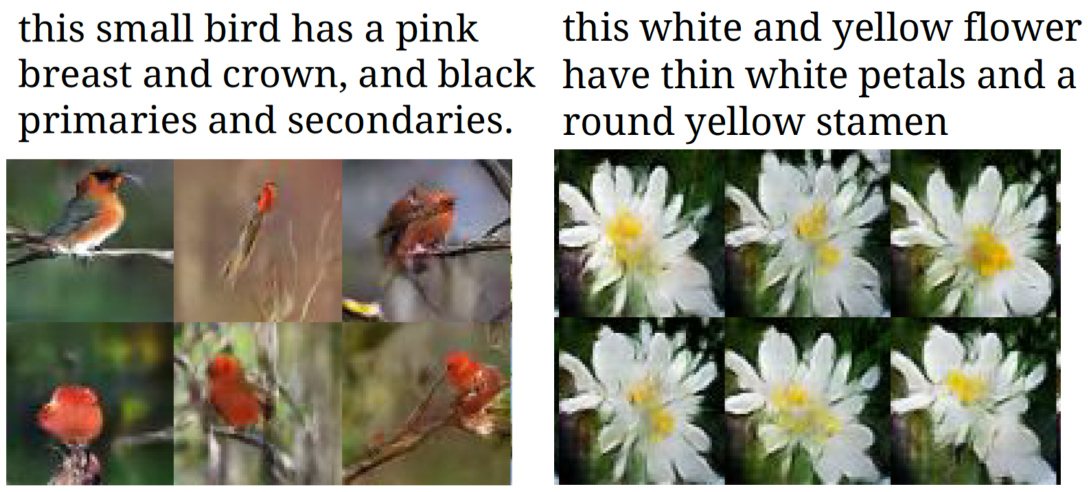

this small bird has a pink breast and crown, and black primaries and secondaries.
this white and yellow flower have thin white petals and a round yellow stamen

Figure 3.3.: GAN for T2I example results on the Oxford-102 and CUB Birds-200 datasets. Figure from Reed et al. (2016), with permission from Scott Reed.

3.3. Stacked Generative Adversarial Networks

Zhang et al. (2017) explored the usage of directly upsampling the generated image to gain a higher resolution image. However, this led to training instability and output that did not show real correspondence with the text input. Therefore, they proposed a new architecture, Stacked Generative Adversarial Networks (StackGAN). The main idea behind StackGAN was to employ two separate GANs (Stage-I and StageII GAN). This was to generate images of higher resolution and of higher level of detail than previous T2I architectures.

The Stage-I GAN is conditioned on the text input and generates a low resolution image of 64 x 64. The Stage-II GAN is thereafter conditioned on the image that Stage-I generated in addition to the text input. As seen in figure 3.4, Stage-II then increases the resolution of the input image (e.g. to 256x256) and fills in details. The results of the two different stages of StackGAN on the Oxford-102 and CUB Birds-200 datasets can be seen in figure 3.5.

Zhang et al. (2017) saw that the model proposed in Reed et al. (2016) quickly collapsed towards generating the same image when provided with the same sentence, called mode collapse. According to them, this happened because of discontinuities in the text embedding space, i.e. big distances between the encoded values of different words. Therefore, the model, when provided with the same sentence, easily falls back to the same values, or sentences. To remedy this problem, they introduced a Conditioning Augmentation (CA) technique to smooth out these discontinuities or gaps. This is done by randomly sampling latent text embedding variables from an independent Gaussian distribution, $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$. Here, φ_t is the text embedding and $\mu(\varphi_t)$ and $\Sigma(\varphi_t)$ are the mean and the diagonal covariance matrix over φ_t, respectively. Thus, employing this

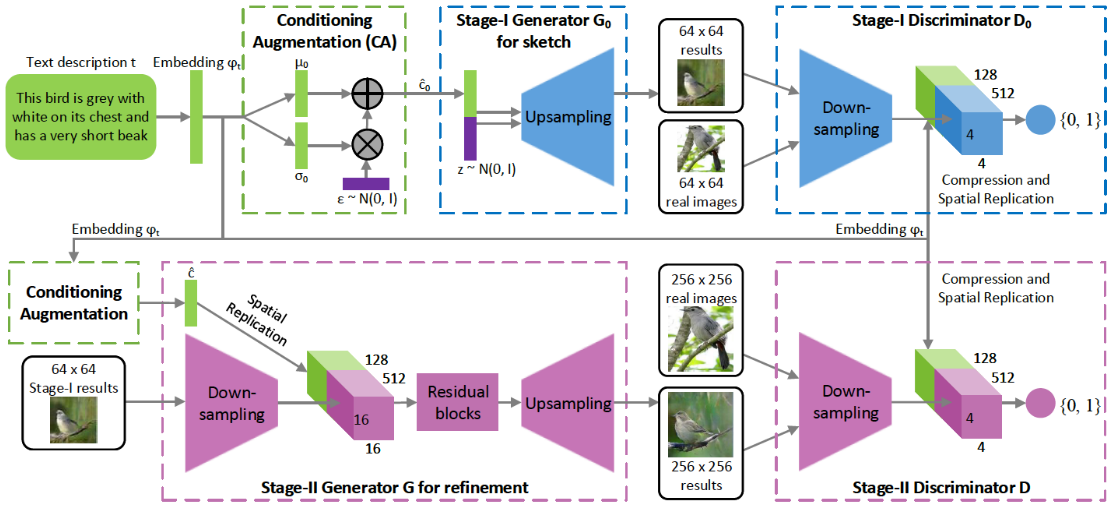


Figure 3.4.: StackGAN architecture. The image caption is encoded into its text embedding before being augmented by the CA. The Stage-I Generator then takes the augmented embedding and noise vector and generates a 64 x 64 image. The text embedding goes through the CA again before serving as input for the Stage-II generator, together with the 64 x 64 image from the Stage-I generator. Figure from Zhang et al. (2017), with permission from Han Zhang.

technique yields more training pairs. In addition to the CA, Zhang et al. (2017) added a regularization term to the objective of the generator through the Kullback-Leibler (KL) divergence to measure the difference between the standard Gaussian distribution and the conditioning Gaussian distribution. This further enhanced smoothness and aided the model in generalizing.

3.4. Attentional Generative Adversarial Network

Xu et al. (2018) saw that most T2I systems only employed a global sentence vector of the input text when generating its corresponding image, without attending to the details of the individual words. They therefore proposed a new model, the Attentional Generative Adversarial Network (AttnGAN), which integrated attention and individual word vectors to iteratively generate more details in the resulting image. As a part of this architecture, they also proposed a Deep Attentional Multimodal Similarity Model (DAMSM), which is a previously trained network for using attention to evaluate how well an image depicts the semantic content of the input text. Thus, the model employs two loss functions: a realism adversarial loss, and the DAMSM-loss.

The authors report that AttnGAN appreciably outperformed previous state-of-the-art systems on both the CUB Birds-200 and the COCO datasets. Generated images from both datasets can be seen in figure 3.7. Additionally, they proposed to use the R-Precision score which is an evaluation metric commonly employed for ranking retrieval results

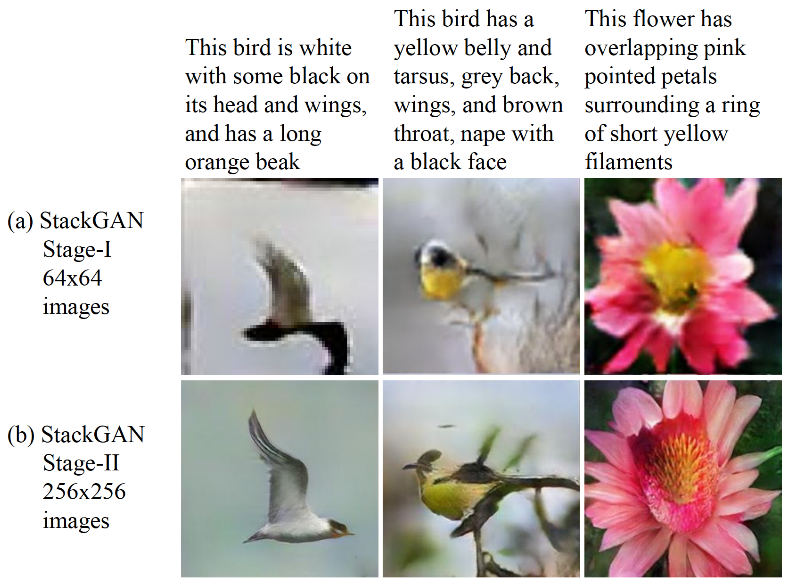

This bird is white with some black on its head and wings, and has a long orange beak
This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face
This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments
(a) StackGAN Stage-I 64x64 images
(b) StackGAN Stage-II 256x256 images

Figure 3.5.: StackGAN example results. Examples from Zhang et al. (2017), with permission from Han Zhang.

in information retrieval. This metric scores the retrieval based on r/R, where r is the number of relevant documents of the total number of retrieved documents R. Similarly, Xu et al. (2018) proposed to use this metric for T2I evaluation through querying relevant image captions for the generated image.

3.4.1. Attentional Generative Network

The first part of AttnGAN, the Attentional Generative Network, employs m generators which take m hidden states as input and generate m images, from small to large. For each iteration, the hidden state is computed using an attention mechanism. As shown in figure 3.6, this attention mechanism takes the word and image features from the previous hidden layer as input and computes a word-context matrix for the input image feature set. This word-context matrix essentially contains information about which words are important to focus on for the current iteration. The word-context matrix and corresponding image features are then combined to generate the images at the next stage.

3.4.2. The Deep Attentional Multimodal Similarity Model

The main task of the DAMSM is to evaluate how well the generated image depicts the input text through mapping the respective parts to a common semantic space and computing their text-image similarity. The model employs two neural networks: a text-encoder to encode the input text, and an image-encoder to encode the generated image.

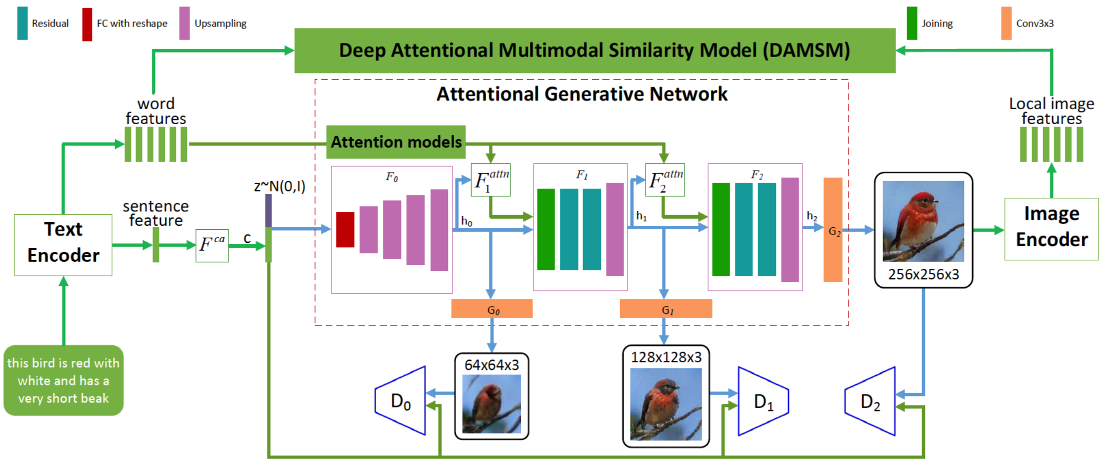


Figure 3.6.: AttnGAN architecture. The image caption is encoded into text feature representations before divided into sentence and word features. The sentence encoding is supplied to a conditioning augmentation, while the word features are supplied to each stacked attention model. The DAMSM produces the loss from matching word features and the generated image. Figure from Xu et al. (2018), with permission from Tao Xu.

The text encoder is a bi-directional LSTM that maps both words and the global sentence to features in semantic space. In this LSTM, each word corresponds to two hidden states, one for each direction. The two hidden states are concatenated to represent the semantic meaning of a word.

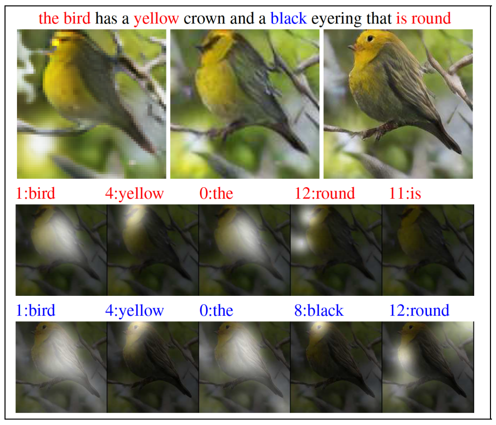
The image encoder maps images to semantic vectors. It is a CNN where the first layers learn local features of sub-regions of an image, while the later layers learn global features. It then maps these features to the common semantic space with the text encodings through a perceptron layer.

The matching is then carried out based on an attention model between the image and the text being evaluated. A similarity matrix is calculated through the dot-product similarity for all possible pairs. For each word, it uses attention to build a context vector for each region and computes a weighted average of these context vectors. This is essentially an abstraction of the image's abstraction of the token — the image's attention vector.

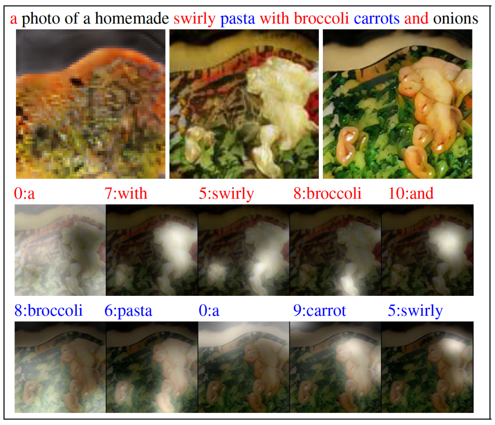
The similarity between this attention vector and the text encoding of the input text is then calculated. The loss for the entire model is calculated in a semi-supervised manner, where the only supervision is between the entire input texts and images.

3.5. MirrorGAN

In 2019, Qiao et al. (2019) introduced a new T2I model, MirrorGAN, built on the models previously mentioned in this chapter. While directly expanding the AttnGAN model,



(a) Images generated on the CUB Birds-200 dataset.



(b) Images generated on the COCO dataset.

Figure 3.7.: AttnGAN example results on the CUB Birds-200 and COCO datasets. Figures from Xu et al. (2018), with permission from Tao Xu.

its main idea was heavily influenced by the CycleGAN model, as it revolves around generating images from text with one GAN, before translating the image back to text with another GAN. It then applies cycle-consistency on the re-generated and the original text descriptions (see section 2.4.3).

The model incorporates the main breakthroughs from all previously mentioned T2I models in this chapter. More specifically, it employs the conditional GAN architecture presented in Reed et al. (2016), and stacks the image Generators and Discriminators in the same way as first presented in Zhang et al. (2017). The model also employs the Conditional Augmentation technique from Zhang et al. (2017), in addition to the attention mechanism proposed in Xu et al. (2018). MirrorGAN differs from the AttnGAN model mainly with regards to the generator loss as MirrorGAN employs the cycle-consistency loss through text re-generation, while AttnGAN employs the generator loss provided by the DAMSM.

The model was compared to other state-of-the-art models (such as AttnGAN) through the R-Precision evaluation metric proposed by Xu et al. (2018), and the Inception Score (IS). These metrics were employed to evaluate the MirrorGAN model trained on both the CUB Birds-200 and COCO datasets. The results show that MirrorGAN achieved new state-of-the-art performance for generating realistic images of high resolution. Additionally, the system was compared with AttnGAN through subjective visual comparisons where it was also deemed superior. Results compared to the AttnGAN model can be seen in figure 3.8.

All mathematical expressions in this section are shown as given by Qiao et al. (2019).

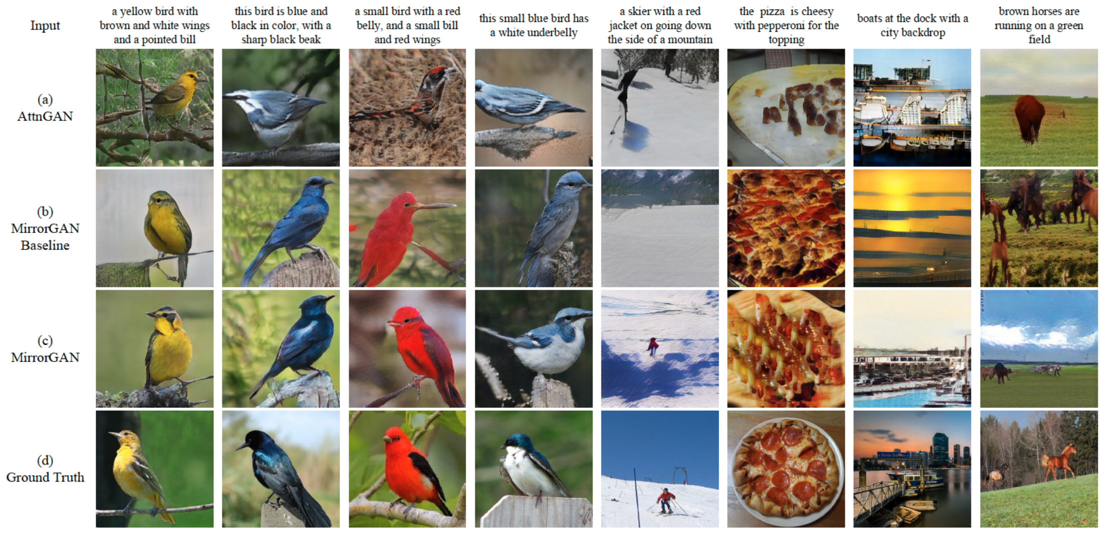

Input
a yellow bird with brown and white wings and a pointed bill
this bird is blue and black in color, with a sharp black beak
a small bird with a red belly, and a small bill and red wings
this small blue bird has a white underbelly
a skier with a red jacket on going down the side of a mountain
the pizza is cheesy with pepperoni for the topping
boats at the dock with a city backdrop
brown horses are running on a green field
(a) AttnGAN
(b) MirrorGAN Baseline
(c) MirrorGAN
(d) Ground Truth

Figure 3.8.: MirrorGAN example results in comparison to AttnGAN results. The left half of the images are generated by the models trained on the CUB Birds-200 dataset, while the right half are generated by the models trained on the COCO dataset. Figure from Qiao et al. (2019), with permission from Tingting Qiao. Repeated here for clarity.

3.5.1. Architecture

The architecture of MirrorGAN is comprised of three modules: A Semantic Text Embedding Module (STEM), a Global-Local collaborative Attentive Module in Cascaded Image Generators (GLAM), and a Semantic Text REgeneration and Alignment Module (STREAM). These modules create word embeddings from the text, generate a corresponding image and re-generate text from the generated image, respectively. All modules and the overall architecture is depicted in figure 3.9.

1. **STEM:** In the STEM module, the text provided serves as input to an RNN that extracts semantic embeddings from the provided text description. It creates both word and sentence level embeddings as stated below:

$$w, s = RNN(T). \tag{3.1}$$

   Where, w is the word embedding, s is the sentence embedding and T is the provided text description. The conditioning augmentation function is then applied to the sentence embedding, s, to create the augmented sentence embedding, s_{ca}.

2. **GLAM:** The second module in MirrorGAN is the GLAM module: a multi-stage cascaded generator which consists of a sequential stack of three image generation networks. The word- and sentence-level attention models generate attentive word- and sentence-context features, respectively. These context features are then used

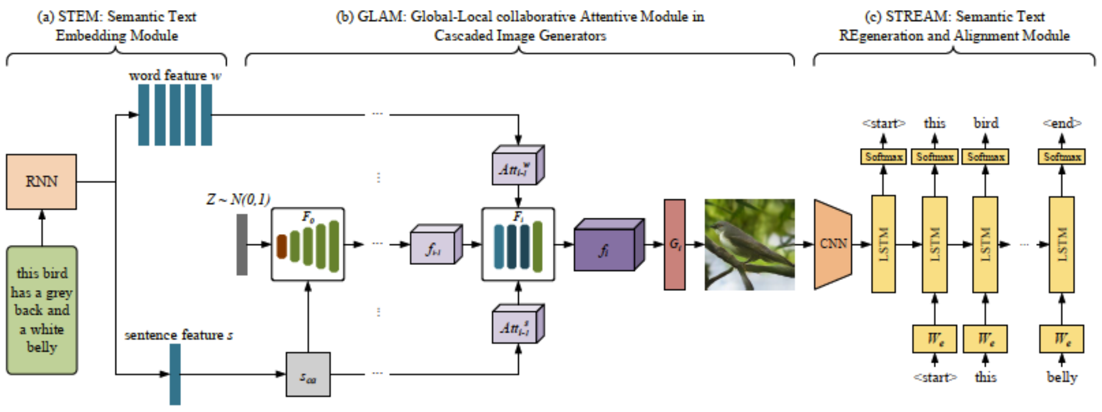

(a) STEM: Semantic Text Embedding Module
(b) GLAM: Global-Local collaborative Attentive Module in Cascaded Image Generators
(c) STREAM: Semantic Text REgeneration and Alignment Module
word feature w
RNN
this bird has a grey back and a white belly
Z ~ N(0,1)
sentence feature s
CNN
LSTM
Softmax
<start>
this
bird
<end>
belly

Figure 3.9.: MirrorGAN architecture showing the STEM, GLAM and STREAM modules. Figure from Qiao et al. (2019), with permission from Tingting Qiao.

to generate visual features that are employed by the image generators to generate a resulting image.

3. **STREAM:** The third module of MirrorGAN, STREAM, is concerned with text re-generation from the resulting image from the GLAM module and is explained in further detail in section 3.5.3.

3.5.2. Generator and Discriminator Objective Functions

The MirrorGAN model aims to generate photo-realistic images through generator stacking and minimizing two adversarial losses: a visual realism adversarial loss, and a text-image paired consistency adversarial loss. The generator, G, and the discriminator, D, of the MirrorGAN are trained alternately during each stage of training.

Generator

For the generator, the i^{th} stage is trained by minimizing a version of the general GAN generator loss, expressed as

$$\mathcal{L}_{G_i} = -\frac{1}{2}\mathbb{E}_{I_i \sim p_{I_i}}[\log(D_i(I_i))] - \frac{1}{2}\mathbb{E}_{I_i \sim p_{I_i}}[\log(D_i(I_i, s))]. \tag{3.2}$$

The first term of the expression is the visual realism adversarial loss while the second is the text-image paired semantic consistency adversarial loss. G_i and D_i are the image generator and discriminator at the i^{th} stage, respectively. I_i is a generated image sampled from the distribution p_{I_i} in the i^{th} stage. In addition to the two adversarial losses, the generator applies a cross-entropy based text-semantic reconstruction loss, called the STREAM loss. This is to compare the underlying semantics of the re-generated image caption of the generated image with original image caption, and is expressed as

$$\mathcal{L}_{stream} = -\sum_{t=0}^{L-1} \log p_t(T_t), \tag{3.3}$$

where T is the provided text description and p_t is the word probability distribution generated from the image in the GLAM module. The resulting generator's total objective function is thus a combination of each ith stage generator's adversarial loss and the STREAM loss, as follows

$$\mathcal{L}_G = \sum_{i=0}^{m-1} \mathcal{L}_{G_i} + \lambda \mathcal{L}_{stream}. \tag{3.4}$$

Discriminator

The objective function for the ith discriminator also consists of visual realism adversarial loss and text-image paired semantic consistency adversarial loss. The loss function can be expressed mathematically as

$$\begin{aligned}\mathcal{L}_{D_i} = &-\frac{1}{2}\mathbb{E}_{I_i^{GT}\sim p_{I_i^{GT}}}[\log(D_i(I_i^{GT}))]\\ &-\frac{1}{2}\mathbb{E}_{I_i\sim p_{I_i}}[\log(1-D_i(I_i))]\\ &-\frac{1}{2}\mathbb{E}_{I_i^{GT}\sim p_{I_i^{GT}}}[\log(D_i(I_i^{GT}, s))]\\ &-\frac{1}{2}\mathbb{E}_{I_i\sim p_{I_i}}[\log(1-D_i(I_i, s))],\end{aligned}$$

where I_i^{GT} comes from the real image distribution $p_{I_i^{GT}}$ in the i^{th} stage. Unlike the ith stage generator which tries to maximize the probability of a generated image being classified as real, the ith stage discriminator tries to minimize this likelihood. At the same time, it also tries to maximize the probability of a real image being classified as real.

The total objective function of the discriminator can then be expressed as

$$\mathcal{L}_D = \sum_{i=0}^{m-1} \mathcal{L}_{D_i}.$$

3.5.3. Details of the STREAM Module

The STREAM module employs an encoder-decoder architecture where the encoder is a CNN that translates from the generated image of the GLAM module to image features representations. Subsequently, the encoder is an RNN (i.e. LSTM) that translates from the image features to a predicted probability distribution of words. The probability

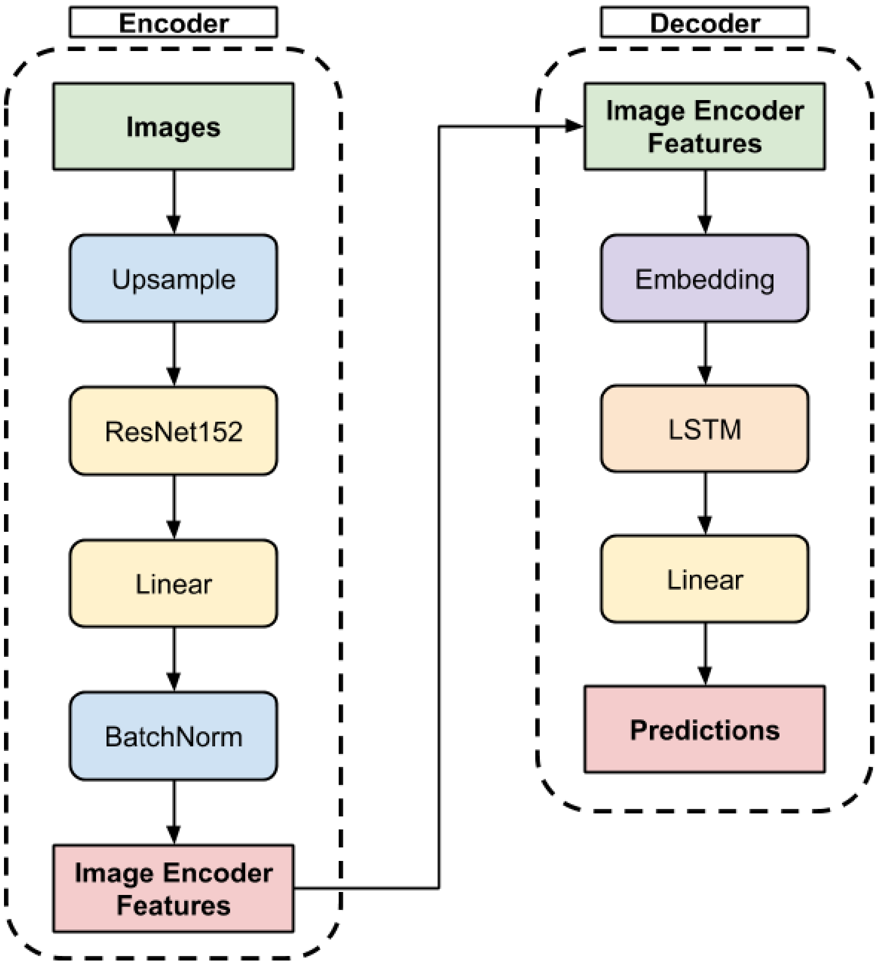


Figure 3.10.: A simple overview of the original STREAM decoder architecture.

distribution thus makes up the predicted re-generated image caption. Figure 3.10 provides a more detailed architectural overview of the STREAM module.

The encoder is made up of a pre-trained version of ResNet152 (see section 2.3.6), where the last fully connected layer is exchanged for a linear transformation layer and a batch normalization layer for re-scaling the parameters of the network. The decoder's first layer is an embedding layer, which acts as a lookup table for the vocabulary the model is trained on. This lookup table is thus concatenated with the image features supplied by the encoder, before serving as input to an LSTM. The last layer is a linear layer which provides the predicted word probability distribution (i.e. image caption).

4. Architecture and Methodology

This chapter introduces the overarching architecture and source code[1] for the project, in addition to a more detailed discussion on the integration of BERT. The project is directly based upon the original implementation of MirrorGAN[2]. Consequently, the overarching architecture of the project remains the same as discussed in section 3.5 and shown in figure 3.9 (see page 44). However, because this work proposes an integration of BERT with MirrorGAN, section 4.1 describes possible integration approaches, and provides insight to the approach chosen. Moreover, section 4.2 introduces the main software employed in the implementation of this integration, while section 4.3 presents the methodology with which the project was implemented.

The term *architecture* is henceforth referred to as the overarching setup of the system. While the term is mostly used for the setup of the final model, it can also be used to refer to the architecture of a sub-part of this final model. A *module* is a separate, stand-alone, part of a bigger model, e.g. the different modules of MirrorGAN: STEM, GLAM and STREAM. A *model*, on the other hand, is either a specific artificial neural network, or an entire system as whole, e.g. the MirrorGAN model. Throughout this and its subsequent chapters, the terms *original* and *new* are used for the two versions of the MirrorGAN model; the original and the one integrated with BERT, respectively.

4.1. Integration of Bidirectional Encoder Representation from Transformers with MirrorGAN

There are three main ways of integrating BERT with MirrorGAN. These approaches include employing BERT for generating sentence and word-level embeddings from the original image caption in the STEM module, evaluating the semantic consistency of the original and the re-generated image captions, and re-generating the image caption from the generated image in the STREAM module. This work will focus on the latter of these approaches as it was suggested as a potential improvement to the STREAM module by Qiao et al. (2019).

Furthermore, the proposed extension is built on the image captioning model proposed by Xu et al. (2015), which is then combined with BERT. This combination is based on work found on GitHub[3], which was modified to work the rest of the project architecture. The encoder of this implementation uses a pre-trained version of ResNet101 from which

[1] `https://github.com/cammiida/masters-project/`
[2] `https://github.com/qiaott/MirrorGAN/`
[3] `https://github.com/ajamjoom/Image-Captions/`

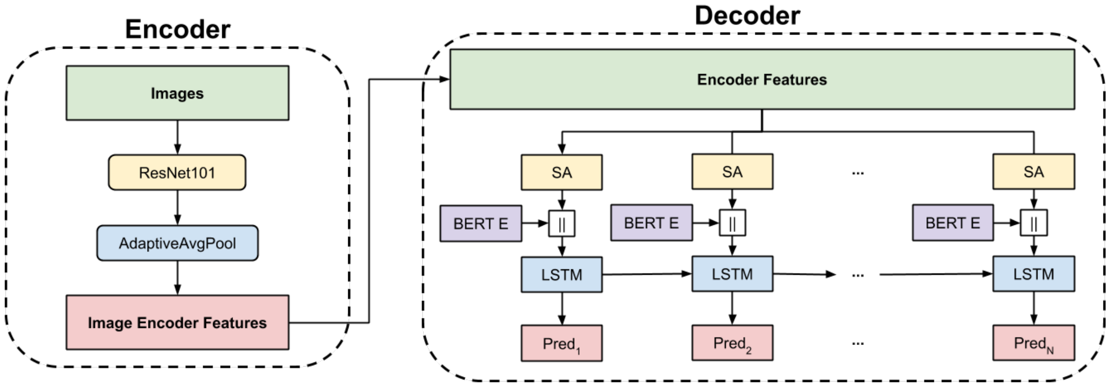


Figure 4.1.: The new STREAM architecture. The encoder uses a pre-trained version of ResNet101 for extracting features before performing an average pooling to mitigate sensitivity to feature locations. The decoder takes these features, calculates the BERT embedding values and performs soft attention independently, before feeding the results to an LSTM. This happens for each time step, where one time step produces the next word in the image caption. The nodes “BERT E” and “SA” denote BERT embeddings and soft attention, respectively.

it extracts the image features. Additionally, the encoder employs adaptive pooling to downsample (i.e. reduce the size of) the feature mapping to mitigate the network being sensitive to the location of the features. The decoder then takes these features, applies the deterministic “soft” attention as described in 2.6 for each time step. Additionally, the captions corresponding to each encoded image are valued through a pre-trained version of BERT base and supplied to an LSTM along with the attention-weighted embeddings for each time step. Figure 4.1 shows an unrolled (i.e. over time) and simplified overview of the implementation of this integration.

4.2. Software

The project was implemented using Python 3.7.4 and PyTorch 1.4.0[4]. As the original implementation of MirrorGAN employed PyTorch for its implementation, this proved to be the natural choice for which deep learning software library to use. The choice of versions for both PyTorch and Python landed on the newest stable releases at the time of implementation (version 1.4.0 and 3.7.4, respectively) for reasons like support, ease of use and functionality. Because the original implementation employed older versions of both library and language, some refactoring was required.

[4]https://pytorch.org/

4.3. Methodology

This section explains the methodology followed when implementing the project. The main focus of this methodology includes the flexibility and reproducibility of the implementation. Consequently this section explains how this was approached.

4.3.1. Reproducibility

A modular approach was adopted, where easy application was the main focus. As the project consists of several models that needed to be trained, the need for a simple overview was prominent. With this in mind, the general structure consists of a few startup or training scripts, one for each module in the architecture (e.g. the STEM module), and one for each experiment. The different configuration settings and hyper-parameters for the models are set through configuration files, which can be supplied as arguments through the command line when running the model. All configuration files employed can be found in Appendix C. Other important configuration parameters are also made available through the command line, such as setting a random seed, and choosing which additional configuration file to supply.

Important assets for the different models are separated into different folders. These folders contain the dataset, the output (or results) from the training of the different models, and the actual trained models. Other shared functionality between the different files in the architecture, like loss and utility functions, are also separated into their own folders and files.

The employed Python libraries are saved to a `requirements.txt` file with their respective versions included. This provides a simple way for other developers to install the correct version of the different Python libraries employed through a terminal command, which consequently facilitates running the project correctly. Additionally, the project is implemented in such a manner that it can resume training on a saved model, if necessary. Given the large number of epochs and the time-consuming task of training, this functionality simplifies the training of the generators and discriminators in that it can be divided into smaller "chunks" of training.

Further instructions to run the code are included in the `README.md` file of the project.

4.3.2. Flexibility

To provide flexibility in the training of the different models, the use of different configuration files was employed. Through these configuration files it is possible to set a significant amount of configuration parameters and hyper-parameters for the models. These parameters include the possibility of setting which version of the STREAM module to use; the new or the original. Additionally, all paths to resources like the dataset are set through these configuration files. This simplified an individual setup for storing the resources.

Moreover, the project also incorporates a resizing module to create a smaller version of the dataset having the same setup as the full version. Consequently, this resizing facilitates testing of the code due to the smaller amount of data employed.

5. Experiments and Results

This chapter introduces the conducted experiments, including their corresponding results and evaluation. Section 5.1 explains the experimental setup, while section 5.2 presents the overarching plan for the experiments. Moreover, section 5.3 presents the results from the conducted experiments.

As stated in the previous chapter, this and subsequent chapters use the terms *original* and *new* models for the two versions of the MirrorGAN model. These two models refer to the original implementation by Qiao et al. (2019), and the one integrated with BERT. This integration is employed in the STREAM module, which re-generates the image caption of the generated image. The implementation of the proposed model and experiments can also be found on GitHub[1].

5.1. Experimental Setup and Constraints

This section explains the setup for conducting the experiments. This includes a description of the hardware employed to train the models and conduct the experiments, complications regarding memory capacity in the GPUs employed and training time, and a description of the input sentences which are used to generate the images supplied in Appendix B.

Most experiments were run through startup scripts which use the files supplied in the experiments folder of the project. Moreover, the configuration parameters for all models (i.e. hyper-parameters) and experiments are provided in Appendix C.

5.1.1. Hardware

The training and evaluation was performed on the NTNU IDUN computing cluster (Själander et al., 2019). The cluster has more than 70 nodes and 90 GPGPUs. Each node contains two Intel Xeon cores, at least 128 GB of main memory, and is connected to an Infiniband network. Half of the nodes are equipped with two or more Nvidia Tesla P100 or V100 GPGPUs. IDUN's storage is provided by two storage arrays and a Lustre parallel distributed file system.

5.1.2. Training Complications

At the beginning of training the GAN models, it was evident that the batch size of 64 employed in the original implementation of MirrorGAN[2] was too big for single GPU on

[1] `https://github.com/cammiida/masters-project`
[2] `https://github.com/qiaott/MirrorGAN/`

the IDUN cluster. To mitigate this problem, a DataParallel container in PyTorch[3] was implemented for the generator and discriminator models. This container parallelizes the model through distributing the input batch over the available GPUs. The division of the batch is implemented in such a manner that the output is gathered from the different devices to one device once the computation ends.

Further, training of the new model proved to be more time-consuming than the original version, with one epoch taking 40 minutes to finish for the new, and 25 for the original model. Training over the full 600 epochs for the new model would require $600 \div (60 \div 40) = 400$ hours, which in turn is $400 \div 24 \approx 17$ days. Moreover, for the original model it would require $600 \div (60 \div 25) = 250$ hours, which is $250 \div 24 \approx 10$ days.

Through e-mail correspondence, the main author of the MirrorGAN paper (Qiao et al., 2019), Tingting Qiao, provided further helpful insight to their implementation of the MirrorGAN model. While running the original model themselves, it took eight days to run the entire model on the COCO dataset over 600 epochs. While the amount of resources they had available is not known, this is relatively close to the training time experienced in this work.

Moreover, due to time constraints at the end of this work, it was unattainable to train over all 600 epochs for both versions of the model. Consequently, they were trained as much as possible, and reached a total of 160 epochs. Both models were trained over the exact same amount of epochs to ensure logical comparison.

5.1.3. Experiment Inputs

Two distinct sets of sentences were chosen as input for conducting the experiments. The first set contains sentences taken from Qiao et al. (2019) where they serve as a comparison to AttnGAN, while also demonstrating the limitations of a generalized T2I model at the time of writing. Generalized means the ability to generate a wide range of realistic images based on varying natural language input. In addition to these sentences, the set is supplied with a selection of example captions from the COCO dataset that also describe physical scenes.

While the first set of sentences describe physical scenes, the second set does not. The terms “descriptive” and “non-descriptive sets” are used to refer to the separate sets of sentences. The two sets can be found in tables 5.1 and 5.2. The difference between these two sets of sentences is used to see how the models respond to the two types of text input: one slightly ambiguous and the other very ambiguous. The word ambiguous is used here for the multi-modality of the problem where a sentence “A car was standing in the middle of the street” definitely yields for a car in the middle of a street, and “I felt like wearing blue today” does not have any clear “guidelines” (except that it might be appropriate to use the color blue).

[3]https://pytorch.org/docs/stable/nn.html#dataparallel/

Table 5.1.: Experiment inputs: Descriptive sentences.

#	Sentence
1	A skier with a red jacket on going down the side of a mountain.
2	The pizza is cheesy with pepperoni for the topping.
3	Boats at the dock with a city backdrop.
4	Brown horses are running on a green field.
5	A bunch of vehicles that are in the street.
6	A lamp with a shade sitting on top of an older model television.
7	A stationary train with the door wide open.
8	A street that goes on to a high way with the light on red.
9	A large white teddy bear sitting on top of an SUV.

Table 5.2.: Experiment inputs: Non-descriptive sentences.

#	Sentence
1	I love going to art galleries and seeing all the beautiful and strange creations they have there.
2	It is utterly terrific that he got accepted to that school.
3	The weather has been really horrible these last few days.
4	Today we are going to do some shopping for Lucy's prom.
5	We are about five minutes late, I'm afraid.

5.2. Experimental Plan

Preliminary to the experiments, a set of four images for each sentence in the descriptive and non-descriptive sets was generated for each of the models, and repeated over every 32nd epoch up to epoch 160. This yields five different epoch steps, and consequently 4 images · 14 sentences · 5 steps = 280 images per model. The total amount of images is thus 280 · 2 models = 560 images. Due to this large amount of images, only a few examples are included in this chapter. The full set of images can be found in Appendix B.

The overarching plan for the experiments is divided in three. The first part is concerned with comparing the two STREAM modules: the new and the original, employing the 1-, 2-, 3- and 4-gram versions of the Bilingual Evaluation Understudy (BLEU) metric. As the change of the STREAM module is a core component of this work, it is natural to investigate how the new module performs compared to the original version. The second part of the experiments focuses on the computation of the FID score of the images generated by the final two T2I models. The scores computed are then used to compare the two models in a quantitative and objective manner. The third part of the experiments revolves around generating images for the two distinct sets of sentences introduced in section 5.1.3 by both models, before evaluating a subset of these images

Scores	New model	Original Model
BLEU-1	0.780	0.357
BLEU-2	0.603	0.110
BLEU-3	0.477	0.037
BLEU-4	0.379	0.015

Table 5.3.: BLEU scores for both new and original STREAM models calculated and averaged over the entire validation set of the COCO dataset.

through conducting a survey. This survey aims to serve as a qualitative support to the FID metric, in addition to providing interesting thoughts on the generated images and T2I synthesis in general.

5.3. Results

This section presents the results from the different experiments. These are presented in the same order as introduced in section 5.2 starting with the BLEU scores of the compared STREAM modules. Then, the FID scores of the two final versions of the MirrorGAN model are presented, before the results from the conducted survey are summarized.

5.3.1. Comparing the STREAM Modules

The two STREAM modules were compared using the BLEU metric, which is commonly employed to evaluate the quality of machine translated text, including I2T. The score was calculated for each batch of the validation set of the COCO dataset and averaged over all batches. The results for both models are presented in table 5.3 and show that the new STREAM model clearly outperforms the original version when measured through the BLEU metric.

As the change to MirrorGAN proposed in this work is confined to the STREAM module, the comparison of the new and original versions of the module is central. This is due to the generator loss being partly composed of the semantic similarity loss between the input and the re-generated image caption, which suggests that an improvement of the re-generation of the image caption means improved learning for the generator. Consequently, change in this module might be the direct cause of any performance differences of the two final MirrorGAN models.

5.3.2. Fréchet Inception Distance

The second part of the experiments is based on evaluating the performance of the two versions of MirrorGAN using the Fréchet Inception Distance (FID) metric. The FID metric was developed as an improvement to the commonly employed Inception Score through, amongst other traits, basing its score on comparing the generated images to

	Runs			
	1	2	3	Average
New model	94.16	93.62	91.87	93.22
Original model	96.57	96.74	96.51	96.61

Table 5.4.: FID scores for both the new and the original model run multiple times on a subset of the validation set.

real images. Consequently, this section employs the FID metric to approach an objective evaluation of the generated images of both models as an addition to the conducted survey.

Due to the significant overhead of computing the FID score, a subset of the validation set in the COCO dataset was created through the dataset resizing module of the project (see section 4.3.2). For this resizing, a random sample of 5% from the validation set was chosen, which resulted in a subset of 2,023 images with corresponding image captions. Moreover, although the dataset contains five image captions per image, only the first caption per image was chosen for this task to ensure that the amount of images generated was the same as the amount of images in the validation set. All generated images were saved to the same folder, before a PyTorch implementation of the FID score[4] was employed to calculate the FID between this folder and the folder containing the validation images from the resized version of the dataset. The FID was computed on this subset three times on both models for the last epoch and provides a clear indication of how the models perform relative to each other.

The results of the calculations are shown in table 5.4. A lower FID score indicates a "shorter distance" between the real and the generated images and is interpreted as a better result. It is clear that the new model slightly outperforms the original in each run. However, from subjective inspection, the images generated by the models trained over 160 epochs were judged to be of lower quality than the images generated by the models trained for fewer epochs. This was especially prominent for the original model. Due to this observation, the FID score was also calculated for all intermediate steps in the training, with step size 32. The results of this calculation is provided in table 5.5. This table indicates that the original version of MirrorGAN performed best overall when trained over 64 epochs, and degraded with each epoch step after that. Moreover, this is in accordance with the subjective evaluation of the quality of the images. Even though the new model outperforms the original when trained over all epochs, any other epoch step shows that the original version is superior. A further discussion is provided in section 6.1.

5.3.3. Survey

The generated image sets from the last epoch (epoch 160) were included in a survey that was conducted to supplement the quantitative measures of the two previous experiments

[4]https://github.com/mseitzer/pytorch-fid/

	Epochs				
	32	64	96	128	160
New model	95.42	93.08	106.50	93.28	93.94
Original model	75.69	72.71	80.93	85.56	97.55

Table 5.5.: FID scores for both the new and the original model run on a subset of the validation set for each epoch step.

with a qualitative comparison of the generated images by the two models.

The initial section of the survey included questions about the respondents' experience in photography, visual arts and photo composition. This section was included to provide context of what type of people were answering the survey. The second section of the survey compared two sets of images generated by the new and original models for each sentence in the descriptive set by asking the participants of the survey which of the image sets corresponds best with the provided sentence. Furthermore, the third part compared the image sets generated by the sentences in the non-descriptive set by asking which one is more interesting.

The survey was designed to primarily compare the two models, while also providing additional optional questions. This ensured that the survey could be completed in a short amount of time while also providing the option of including more information. These additional questions focused on the overall impression of the images, if any separate image sets stood out from the rest, and in what ways a T2I system could be useful. Moreover, the survey was posted on Facebook and the social news aggregation, web content rating, and discussion website, Reddit[5]. The survey received a total of 33 responses and is provided in Appendix A.

The first section included questions about the respondents' experience within photography, visual arts and image composition. The values range from 1 for not experienced/interested, to 5 for very experienced/interested. Table 5.6 shows the results from this section of the survey. Each value cell in the table is the percentage of respondents choosing a particular value for a corresponding question, represented as a decimal number. Consequently, all numbers for any one question sum up to 1. The table provides a summary of what type of people have answered the survey. It indicates that most of the respondents are fairly interested in photography and visual arts. Moreover, the respondents also generally report a low to medium amount of knowledge and experience in both photography and visual arts. However, it can also be seen that a few respondents state to be very experienced in both fields. Moreover, many also report low to medium knowledge of image composition.

By inspection of the answers to the second section, presented in table 5.7, it can be seen that the image set generated by the original model (image set 2) scored higher than the new (image set 1) 5/9 times. Furthermore, table 5.8 presents the results from the third

[5]Posted to the subreddits (i.e. channels): `https://www.reddit.com/r/SampleSize/` and `https://www.reddit.com/r/takemysurvey/`

Table 5.6.: Survey section 1: Showing the questions and responses for section 1 of the survey. An answer of 1 means low or poor, while 5 means great or high.

Experience	1	2	3	4	5
How would you rate your experience with photography?	0.152	0.485	0.182	0.182	0.000
To what degree have you done photography?	0.242	0.515	0.152	0.061	0.030
How would you rate your knowledge in visual art?	0.242	0.364	0.303	0.061	0.030
To what degree have you created visual art (not photography)? (e.g. painting, drawing, pottery)	0.212	0.364	0.273	0.061	0.91
To what degree would you rate your knowledge of image composition?	0.303	0.364	0.182	0.121	0.030
How interested are you in photography?	0.121	0.303	0.333	0.242	0.000
How interested are you in visual arts?	0.061	0.242	0.455	0.182	0.061

section, which addresses how interesting the images generated from the non-descriptive set are. The results from this section show that the images generated by the original model were again preferred over the new, having a higher score than the new for 3/5 questions.

The last part of the survey includes extra optional questions about the respondent's general perception of the images and if any image set stood out from the rest. More specifically, the questions were as follows:

1. What do you think of the quality of the images you have been presented with?
2. Does any (one or several) image set(s) stand out to you?
3. Do you think a system that generates images based on text would be useful? If so, what for?

The first question received answers ranging from "Quite poor" and "They seem very abstract and surreal" to "It is not perfect, but it was surprisingly good. I expected it to be lower." Moreover, several respondents saw the generated images as interesting and that some look like art, based on statements such as "I didn't think they matched very well the description, but some generated interesting art which kinda had elements of what you wanted to generate" and "They looks like partly figurative art, not very much like the text descriptions". When asked if there were any images that stood out from the rest in question 2, most respondents did not think so. However, one respondent mentions "The first image of each pair seemed more vibrant and meaningful compared to the second." While it is not certain whether the respondent meant the first image of each set or image set 1, another statement supports the latter interpretation through "Yeah,

Table 5.7.: Survey section 2: Asking which image set corresponds best with the given text description. Image set 1 refers to the images generated by the new model, while image set 2 refers to the images generated by the original model.

Sentence	Image set 1	Image set 2	Both image sets equally fit the description
A skier with a red jacket on going down the side of a mountain.	0.242	0.697	0.061
The pizza is cheesy with pepperoni for the topping.	0.636	0.182	0.182
Boats at the dock with a city backdrop.	0.576	0.242	0.182
Brown horses are running on a green field.	0.242	0.364	0.394
A bunch of vehicles that are in the street.	0.333	0.515	0.152
A lamp with a shade sitting on top of an older model television.	0.061	0.485	0.455
A stationary train with the door wide open.	0.939	0.030	0.030
A street that goes on to a high way with the light on red.	0.788	0.152	0.061
A large white teddy bear sitting on top of an SUV.	0.212	0.333	0.455

overall I found that when I was asked to consider images based on interest/aesthetics, I preferred image set 1 consistently over image set 2." Moreover, there were multiple interesting suggestions provided for question 3. The answers convey a general positiveness towards the usefulness of a well-functioning T2I system. A list of all responses to these questions can be found in Appendix A. It is important to note that the questions were optional, and thus the number of responses for these questions is not as high as the total number of respondents.

Table 5.8.: Survey section 3: Asking which image set is more interesting. The question is supplied with the corresponding text description. Image set 1 refers to the images generated by the new model, while image set 2 refers to the images generated by the original model.

Sentence	Image set 1	Image set 2	Both of them are equally interesting	Neither are any interesting
I love going to art galleries and seeing all the beautiful and strange creations they have there.	0.273	0.515	0.091	0.121
It is utterly terrific that he got accepted to that school.	0.182	0.394	0.091	0.333
The weather has been really horrible these last few days.	0.303	0.364	0.091	0.242
Today we are going to do some shopping for Lucy's prom.	0.424	0.212	0.182	0.182
We are about five minutes late, I'm afraid.	0.667	0.182	0.061	0.091

6. Evaluation and Discussion

This chapter provides a final discussion of the work presented in previous chapters. Evaluation of the results is an important part and a challenge of image generation. There are no clear guidelines for evaluating what is a high quality generated image as there are multiple ways of generating an image which can all be deemed equally good. Additionally, the evaluation of the results is often highly subjective, although it may be argued that generated images of high quality should exhibit some form of realism[1]. Specifically, in the case of T2I synthesis, they should depict the objects mentioned in the input sentence. Even so, this chapter aims to discuss the different results presented in section 5.3, through both quantitative and qualitative measures, in an attempt to compare the two versions of the MirrorGAN model.

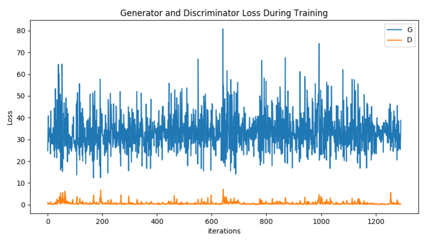
Hence, sections 6.1 and 6.2 discuss the quantitative and qualitative results of the experiments, respectively. Subsequently, section 6.3 aims to answer the research questions through re-stating each question, before listing major findings that help answer the question at hand. Moreover, after each research question has been discussed, this section also includes a review of how the Thesis goal has been met. Additionally, as the last part of this chapter, section 6.4 will discuss the limitations of this work.

6.1. Quantitative Evaluation

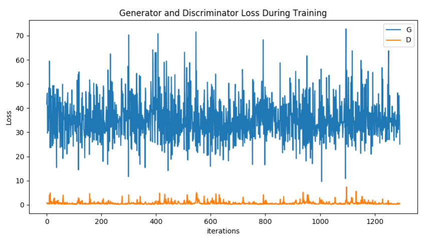
As observed in section 5.3.1, the new, proposed version of STREAM clearly outperforms the original version. Consequently, it is natural to think that this would improve the image generation in the final model. However, when comparing the final MirrorGAN models over all epoch steps, as listed in table 5.5, it is evident that the original version generally outperforms the new when measured through the FID score. This is true for all but the last epoch step, as shown in table 5.4, where the new version slightly outperforms the original in every computed iteration. However, this is not due to the new model improving, but rather the deterioration of the original version. From the FID scores in table 5.5, we can infer that neither of the two models learn after epoch 64, as their FID scores do not decrease. Consequently, the question arises about why this happened.

In the training of a GAN, the two players (the generator and discriminator) are competing in a zero sum game, also called a minmax game. This training is difficult because the goal is not for either player to win, but rather have them reach a state where neither player gains anything from changing its behaviour. Such a state is also called a Nash equilibrium, and reaching it means that we have a stable, or converged,

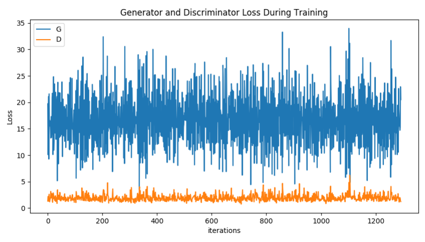
[1]This is dependent on the application, but true for this work.



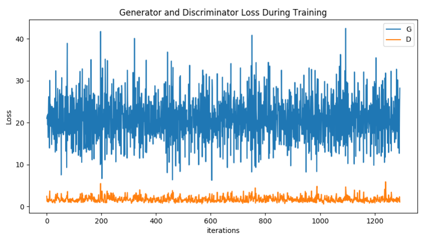
(a) New model. VE: 32



(b) Original model. VE: 32



(c) New model. VE: 128

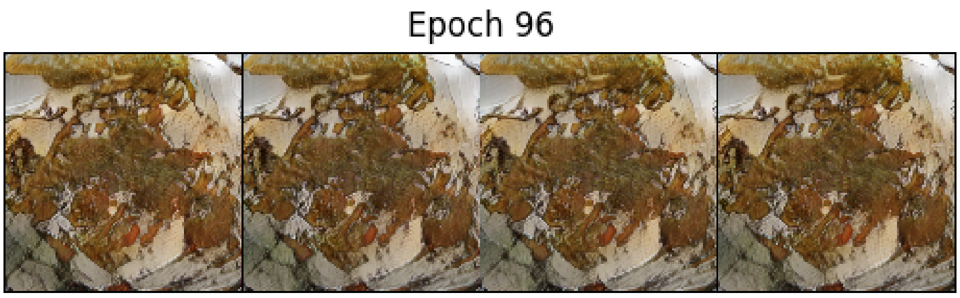


(d) Original model. VE: 128

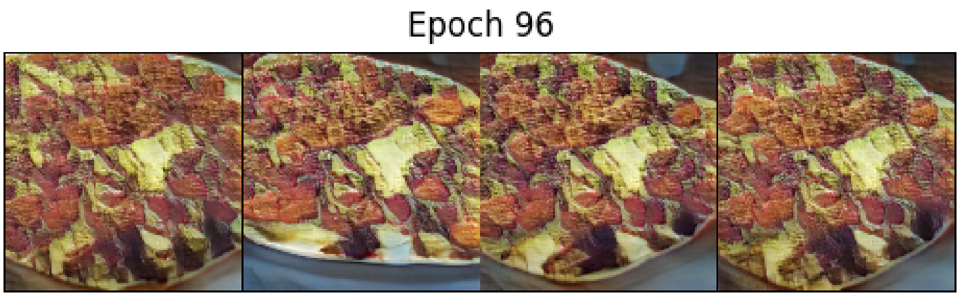
Figure 6.1.: Comparison of the new and original models trained over 20 epochs with a visual embedding size of 128. “VE” denotes the visual embedding size.

network. Reaching this state is theoretically feasible, although reasonably difficult to achieve in practice. On the other hand, in an unstable network, the two players attempt to “counteract” the move of the other player at each iteration, which contributes to sub-optimal learning and the network not converging. Another frequent problem when training GANs is called mode collapse means that the generator fails to generate varying images (i.e. multiple modes). That is, if a generator is supposed to generate an image of any number between 0 and 9, and it only generates the even numbers, it has fallen into a partial mode collapse. In a full mode collapse, on the other hand, the generator would only generate one out of the ten numbers. A third common problem is when one player becomes much more powerful than the other, e.g. if the discriminator becomes too good at detecting whether an image is real or fake. Research indicates that this hinders the learning of the generator, as it contributes to vanishing gradients (Arjovsky and Bottou, 2017). Because of all aforementioned problems, the hyper-parameters of a GAN need to be carefully constructed to create an optimal game.

When inspecting the loss plots for the new and original models, it is clear that the discriminator loss approaches 0 in early stages of the training. The loss plots of the new and the original model trained over 20 epochs are shown in figure 6.1a and 6.1b. They are typical examples of the imbalance in the minmax game previously introduced, where one player is exceedingly better than the other. As stated in Goodfellow et al. (2014), vanishing gradients for the generator is a common problem when the discriminator’s job is very easy. This is the case for T2I synthesis as it is a highly multi-modal problem, where the generator performs poorly in the early stages of training (i.e. generates unrealistic



(a) New model



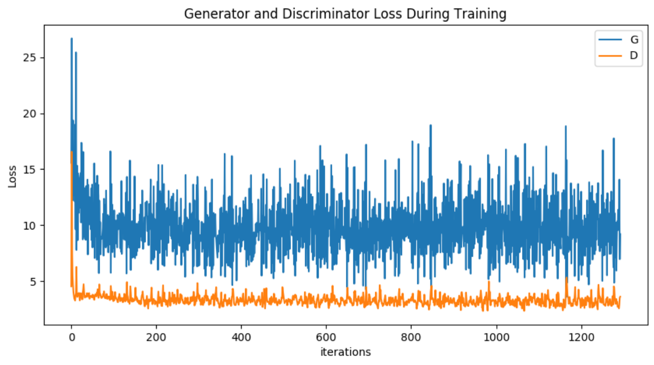
(b) Original model

Figure 6.2.: Similar generated images for "The pizza is cheesy with pepperoni for the topping"

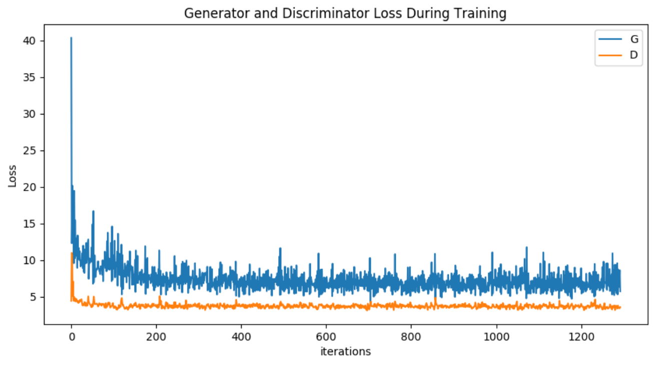
images far from the target distribution). This is especially true for a generalized T2I task as is explored in this work through employing the COCO dataset.

While there may be several causes for the discriminator exceedingly outperforming the generator, a likely scenario is the sub-optimal setting of hyper-parameters of the network. One such sub-optimal hyper-parameter may be the low visual embedding size of 32. To investigate if this hyper-parameter adjustment might improve training, the new and the original models were re-trained with an increased visual embedding size of 128 over 20 epochs. The loss plots of these models are shown in figure 6.1c and 6.1d. When comparing all loss plots in figure 6.1, it is evident that in the loss plots for the models with a higher visual embedding size, the discriminator loss has not yet approached 0. This is a clear indication that an increased visual embedding size may mitigate the problem of vanishing gradients, and thus improve the learning of the generator. However, this is only one example of a hyper-parameter change, and there may be various other adjustments that could further improve training. Furthermore, other approaches have been developed to mitigate the problem of vanishing gradients, including the Wasserstein loss and a modified minmax loss, where, instead of minimizing $\log(1 - D(G(z)))$, the generator maximizes $\log D(G(z))$, as proposed by Goodfellow et al. (2014).

Further comparison of the generator losses of the new and the original model indicates that the new model yields lower loss values. While inspecting the loss values of the generator is generally not a good indicator for how well a GAN network performs, these



(a) Visual embedding size: 32



(b) Visual embedding size: 128

Figure 6.3.: First epoch losses for the new model with visual embedding size 32 versus 128.

differences suggest that the new model might have some merit.

Through inspection of the generated images for all epoch steps, provided in Appendix B, we can see that several of the images generated by both models in the later steps are exceedingly similar. An example of this is shown in figure 6.2. These observations strongly suggest that the individual GANs have entered a partial mode collapse. The mode collapse indicates that the generator has one or a small sample of fixed images that it deems as optimal to minimize its loss. Various approaches have been made to remedy the problem of mode collapse, including the Wasserstein loss (Arjovsky et al., 2017) and unrolling GANs (Metz et al., 2016). While it is unclear why mode collapses happen, a partial cause in this case may be the low visual embedding size employed during training of both models. However, due to time constraints the mitigation strategies for both vanishing gradients and mode collapse are left for further research.

Moreover, the generator in MirrorGAN tries to minimize the visual realism adversarial loss and the text-image paired semantic consistency adversarial loss. In addition to these adversarial losses, it also applies a text-semantic reconstruction loss to evaluate the similarity of the original input and the re-constructed image caption, called the STREAM loss (see section 3.5). While the STREAM loss is supplied to the generator's loss, it is not supplied to the discriminator's loss. With a better image captioning model (i.e. image caption re-generation), this might lead to a more accurate STREAM loss, and thus help the generator learn. However, when the generator performs poorly in generating realistic images, the re-generation of the image caption from the unrealistic images will also perform poorly, regardless of how well the image captioning model proves to perform on real images. Because the models trained in this work ran into problems like vanishing gradients and mode collapse, in addition to not being trained enough, the generators in the respective models were not able to generate realistic images. Therefore, the real advantage of an improved STREAM model is not proved, and should be further investigated in future work.

6.2. Qualitative Evaluation

The survey indicates that for slightly more than half of the image set comparisons, the respondents prefer the images generated by the original model over the ones generated by the new. However, through further inspection of the results, it can be argued that the differences are not major. Also, while the results might favour the original model for these particular images, they might change for different images, generated from a different set of sentences. This is supported by the FID scores in table 5.4 in that these scores also indicate that the difference between the two models is not great.

Moreover, through the answers of the survey, a question arises if a T2I system might have other value apart from generating realistic images, such as generating creative art. This, however, calls for a brief discussion on what creativity is, and how it relates to computers. The rest of this paragraph is based on the pre-study of this work, conducted during the Fall of 2019. Boden (2004) described creativity in several ways, one of which defines an artefact as creative if it is surprising and valuable. Moreover, Colton and Wiggins (2012) discussed the impact of the acts of the software or hardware in addition to the output alone. In essence, they argue that a focus on the whole process is key when it comes to creative generation, and explaining the process of production is essentially framing the generated artefact with information that adds value.

Following these definitions, it can be argued that a T2I system based on neural networks is itself creative. This is due to its inner workings, or that the network parameters essentially make up a "black box". This means that it is difficult to connect certain parameter values to specific traits of the input, e.g. features in an image. Moreover, the parameters can be seen as the network's representation of what it has learned, which is not directly controlled by humans. Consequently, through the definition of creativity by Colton and Wiggins (2012), and observing that the images are described as "interesting" in the conducted survey, it can be argued that the networks employed in this work are creative. This is facilitated through the employment of the COCO dataset, in that it provides a more complex base than the CUB Birds-200 and thus leaves room for a higher degree of interpretation.

Moreover, a few answers to the last question of the survey convey that the generated images can look like art. These answers include statements that the images presented "look really interesting", "looks partly like figurative art " and "the designs are interesting". Through this, it can be argued that the images are surprising, while having value through being interesting and making people think, which follows the definition of creative artifacts stated by Boden (2004).

6.3. Thesis Goal and Research Questions

Research question 1 *How can the text-to-image system and the natural language model be combined?*

The natural language model, BERT, was integrated with the text-to-image model, MirrorGAN, through the image caption re-generation in the STREAM module. While there may exist additional ways to integrate BERT with MirrorGAN, three major approaches have been discussed. These approaches revolve around employing BERT for creating text embeddings in the STEM module, for semantic consistency evaluation between the input and the re-generated image caption, and for re-generation of the image caption in the STREAM module. The latter of these approaches was chosen based on a suggestion for future work in Qiao et al. (2019).

The new version of the STREAM module clearly outperforms the original version when compared through the BLEU metric, as presented in table 5.3. The new STREAM model was built up of the approach proposed by Xu et al. (2015) which is then integrated with BERT (Devlin et al., 2018). As the generator loss is partly composed of the semantic similarity loss between the input and the re-generated image caption, an improvement of the re-generation of the image caption suggests improved learning for the generator.

Research question 2 *How does the combination of the text-to-image system and the natural language model affect image generation?*

When measured through the FID metric, the new version of MirrorGAN only outperforms the original in the last epoch step. However, this is only due to the deterioration of the original model rather than an improvement of the new. These results can be seen in table 5.4 and 5.5. It is not clear why the new model does not deteriorate as much as the original, as both models seem to encounter the problem of the discriminator loss approaching 0 (i.e. vanishing gradients for the generator) in early stages of the training. This is shown in figure 6.1a and 6.1b. Additionally, both generators show signs of a partial mode collapse through generating similar images for the same sentence. Examples of such similar images are presented in figure 6.2.

These problems impair the determination of possible merits of the new STREAM module. This is because of the STREAM module not being able to re-generate high quality image captions due to the lack of realism in the generated images, and not due to the quality of the STREAM module itself. However, the presented work does not provide data to support this. Consequently, this should be investigated further through measuring how well each STREAM module performs during training.

Moreover, through the results of the conducted survey, it can be argued that the images generated by the new model can appear slightly more appealing and pleasing than the images generated by the original model. However, this might have happened by chance and the survey received too few responses to draw any conclusions.

Research question 3 *How does the combination of the text-to-image and the natural language models perform when employing a dataset containing complex images?*

Moreover, due to the complexity of the COCO dataset and the task of T2I synthesis, in addition to the complications faced during training, the generated images do not seem realistic. However, they do exhibit some resemblance of the input sentence, although of low resolution and quality. As stated in the answers of the survey, the image captions often needed to be supplied with the image to make it clear what it was trying to depict. Although the training of the model encountered problems, in addition to the model not being trained enough, the generated images do not appear to be notably worse than the images generated by Qiao et al. (2019) trained on the same dataset. These images can be seen in the last four columns of row three in figure 1.1.

On the other hand, it can be argued that a GAN-based T2I system is creative and has value apart from generating realistic images through the definitions of creativity by Boden (2004) and Colton and Wiggins (2012). This is supported by a handful of responses of the conducted survey indicating that several of the generated images were visually pleasing or interesting.

Goal *Explore the combination of a text-to-image synthesis system and a recent natural language processing model employing a complex dataset*

The goal of this Thesis has been focused on exploring how the MirrorGAN model can be integrated with BERT, and how this combination compares to the original version of MirrorGAN. This focus is based on related GAN-based T2I systems in that they show promising results for generating images from descriptions of single objects like birds or flowers, but struggle to generalize to more complex scenes and generate images of high resolution. Consequently, the models were trained on the large and complex COCO dataset. Additionally, the emergence of new and powerful models in field of NLP provides new opportunities of exploring the possible merits of employing such models in T2I synthesis.

Furthermore, the goal of this work has been reasonably open. This has been deliberate to cater for the possibility of the work to take different directions, as the direction was not clear from the beginning and the field was unfamiliar. Moreover, it was not clear how to compare the two models, specifically. Consequently, the different forms of comparison of the two models took form as the work progressed.

These forms of comparison, or experiments, include a quantitative comparison of the STREAM modules through the BLEU metric, a quantitative comparison of the final T2I model versions through the FID metric and a qualitative comparison of the generated images through a conducted survey.

It can be argued that the goal has been met through the aforementioned comparisons and exploration, although the field needs further research. Additionally, this work highlights the potential to explore other ways of including BERT or other natural language models, possibly based on BERT, while also bringing up the question of how the models would have performed were they trained on another dataset.

6.4. Limitations

The limitations of this work are manifold; this section serves to highlight these limitations, in addition to providing context and mitigation approaches.

Firstly, it is uncertain how much BERT itself contributed to the improvement of the STREAM module. This is due to the implementation of a combination of the image captioning approach introduced by Xu et al. (2015) and BERT in the same module, which makes it difficult to distinguish what merits the individual parts provide. Consequently, an improvement would compare how this module performs with and without BERT.

Furthermore, due to the problems of vanishing gradients, mode collapse and non-convergence during the training of the final T2I models, it is difficult to quantify the merits of the new version. Consequently, different hyper-parameters and other techniques to mitigate these problems and ensure proper training are important. Additionally, with these problems mitigated, training over all 600 epochs is essential to investigate the full potential of the new model.

Moreover, due to the difficulties faced during training of the T2I models, it is hard to say how much of an impact an improved STREAM module has. It is hypothesized that it does not considerably impact the models trained in this work because the images generated are of low quality. Consequently, this ensures that the image caption re-generation is also poor, which limits the learning of the generator. However, further quantitative inspection must be performed to properly support this hypothesis.

Additionally, the survey only received a small number of responses, which limits its leverage when it comes to comparing the two T2I models. Furthermore, the length of the survey was limited, in addition to providing images solely from the last epoch. It would have been interesting to see if the respondents may have preferred images generated by the models trained over fewer epochs.

7. Conclusion and Future Work

This Master's Thesis has compared the text-to-image synthesis model, MirrorGAN, with a version of itself combined with the natural language model, Bidirectional Encoder Representations from Transformers. BERT was integrated into the STREAM (i.e. image captioning) module of the MirrorGAN model and the implementation was based on the proposed image captioning model by Xu et al. (2015). Subsequently, the proposed new version of the module and the original module were compared through employing the Bilingual Evaluation Understudy (BLEU) metric. Through this metric, the new version was deemed considerably superior to the original.

The two versions of the MirrorGAN model were then trained over 160 out of 600 epochs, before being compared through employing the Fréchet Inception Distance (FID) metric, conducting a survey, and performing a subjective evaluation. The new model showed to slightly outperform the original through both its FID score when trained over 160 epochs. However, the original version scored better through the FID metric on all previous measured epoch stages to epoch 160, in addition to being slightly more preferred in the conducted survey. A further study of the FID scores of both models indicates that neither had improved after epoch 64. Moreover, this can be attributed to both models encountering problems such as vanishing gradients due to the discriminator performing too well, and mode collapse. These problems most certainly lead to sub-optimal learning.

Furthermore, this sub-optimal learning, in conjunction with the models not being trained over all epochs, resulted in low quality of the generated images with regards to realism for both models. Thus, when these images are of low quality it likely impairs the performance of the STREAM, regardless of how well it is trained and performs on real images. Consequently, it is challenging to determine the merits of the new STREAM module applied in the MirrorGAN model from this work, although the comparison of the STREAM modules through the BLEU metric is promising, and should be the subject of further research.

In addition to the aforementioned findings, it can be argued that the images generated by the T2I models are interesting, which is supported by claims from the conducted survey. This indicates that a T2I system may have value apart from generating realistic images. Moreover, through the definition of creativity by Colton and Wiggins (2012), it can be argued that a GAN-based T2I system is inherently creative. Furthermore, through the answers of the conducted survey, it can be inferred that several respondents found the images surprising and interesting, which follows the definition of creativity by Boden (2004) of that a creative artifact should be “surprising” and “valuable”.

7.1. Contributions

The main contributions of this Thesis lie in the work of combining T2I with BERT. This includes the initial discussion of how they can be combined, in addition to how such a combination may be achieved. The work provides a versatile implementation of the combination of MirrorGAN and BERT, which supports different configurations and separate training of the STEM and the STREAM modules.

A comparison of the two STREAM modules through the BLEU metric show that the new version integrated with BERT clearly outperforms the original version, and implies that the combination of BERT and MirrorGAN might have merit. An overview of related work and summary of the main GAN-based advances in the field of T2I is also provided. Moreover, the two versions of MirrorGAN have been thoroughly compared through calculating their FID scores, conducting a survey and performing subjective analysis.

This work also provides a brief discussion of GAN-based T2I systems having value apart from generating realistic images, such as being creative and interesting. Other contributions include a module for resizing the COCO dataset, and the accommodation for parallel training of the models over multiple GPUs.

7.2. Future Work

This section simplifies suggestions included in previous discussions and serves as a summary of the distinct proposed items for future work. Additional items not previously mentioned are also included. The potential future work is divided into two categories. The first category revolves around improvements and extensions that can be made to the work presented in this Thesis, as follows:

- Train both versions of the model with different hyper-parameter configurations and employ mitigation strategies to avoid vanishing gradients, mode collapse and non-convergence.

- Once the training problems are mitigated, train over an increased number of epochs to evaluate if the differences in quality of the generated images of the two models remain the same.

- Investigate how the STREAM module performs during training to evaluate how much of an impact it has in the early stages of training before the model is able to generate realistic images.

- Investigate how the STREAM module employing the technique proposed by Xu et al. (2015) performs with and without the integration of BERT.

The second category targets a different direction than what has been investigated in this work, as follows:

- Experiment with other ways of combining BERT, such as employing BERT in the Semantic Text Embedding Module (STEM) or for evaluating the semantic similarity of the original and the re-generated image captions.
- Employ ALBERT (Lan et al., 2020) instead of BERT to examine any changes from the original model or the model integrated with BERT. Additionally, it would be interesting to explore how this affects training time.

Bibliography

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv e-prints*, art. arXiv:1701.04862, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv e-prints*, art. arXiv:1701.07875, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, art. arXiv:1409.0473, September 2014.

Shane Barratt and Rishi Sharma. A Note on the Inception Score. *arXiv e-prints*, art. arXiv:1801.01973, January 2018.

Margaret A. Boden. *The creative mind: Myths and mechanisms*. Routledge, 2004.

Simon Colton and Geraint A. Wiggins. Computational creativity: The final frontier? In *European Conference on Artificial Intelligence (ECAI)*, volume 12, pages 21–26. Montpelier, France, 2012.

Michael Denkowski and Alon Lavie. Meteor Universal: Language Specific Translation Evaluation for any Target Language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA, 2014. Association for Computational Linguistics (ACL).

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, art. arXiv:1810.04805, October 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Conference of Neural Information Processing Systems*, volume 27, pages 2672–2680, Montréal, Canada, 2014. Advances in Neural Information Processing Systems (NIPS).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, Nevada, USA, June 2016. IEEE.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Conference of Neural Information Processing Systems*, volume 30, pages 6626–6637, Long Beach, California, USA, 2017. Advances in Neural Information Processing Systems (NIPS).

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. Published by MIT Press.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *The International Conference on Learning Representations (ICLR)*, Virtual (Online) Conference ~~Addis Ababa, Ethiopia~~, 2020.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in COntext. In T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *European Conference on Computer Vision (ECCV). Lecture Notes in Computer Science*, volume 8693, pages 740–755. Springer, Cham, 2014.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints*, art. arXiv:1508.04025, August 2015.

Warren S McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1611.02163, 2016.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent Neural Network Based Language Model. In *11th annual conference of the international speech communication association — INTERSPEECH*, pages 1045–1048, Makuhari, Chiba, Japan, September 2010.

Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv e-prints*, art. arXiv:1411.1784, November 2014.

Maria-Elena Nilsback and Andrew Zisserman. Automated Flower Classification over a Large Number of Classes. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, Bhubaneswar, India, 2008. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135.

Tingting Qiao, Jing Zhang, Duanqing Xu, and Dacheng Tao. MirrorGAN: Learning Text-To-Image Generation by Redescription. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1505–1514, Long Beach, California, USA, June 2019. IEEE.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1511.06434, November 2015.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv e-prints*, art. arXiv:1606.05250, June 2016.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. *arXiv e-prints*, art. arXiv:1605.05396, May 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *Conference of Neural Information Processing Systems*, volume 29, pages 2234–2242, Barcelona, Spain, 2016. Advances in Neural Information Processing Systems (NIPS).

Mike Schuster and Kaisuke Nakajima. Japanese and Korean Voice Search. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, Kyoto, Japan, 2012. IEEE.

Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure. *arXiv e-prints*, art. arXiv:1912.05848, 2019.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Conference of Neural Information Processing Systems*, volume 27, pages 3104–3112, Montréal, Canada, 2014. Advances in Neural Information Processing Systems (NIPS).

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR))*, pages 2818–2826, Las Vegas, Nevada, USA, 2016. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Conference of Neural Information Processing Systems*, volume 30, pages 5998–6008, Long Beach, California, USA, 2017. Advances in Neural Information Processing Systems (NIPS).

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv e-prints*, art. arXiv:1804.07461, 2018.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *arXiv e-prints*, art. arXiv:1704.05426, 2017.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation system: Bridging the Gap Between Human and Machine Translation. *arXiv e-prints*, art. arXiv:1609.08144, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *arXiv e-prints*, art. arXiv:1502.03044, 2015.

Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-Grained Text to Image Generation With Attentional Generative Adversarial Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1316–1324, Salt Lake City, Utah, USA, June 2018. IEEE.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N. Metaxas. StackGAN: Text to Photo-Realistic Image Synthesis With Stacked Generative Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5907–5915, Venice, Italy, Oct 2017. IEEE.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2223–2232, Venice, Italy, Oct 2017. IEEE.

Appendices

A. Survey

The survey information and questions are included in this Appendix. Each subsection is pursuant to the sections in the survey. All images are omitted in this overview, but can be found in Appendix B as the images generated for epoch 160. Descriptions in italics signifies that the text was supplied in the survey itself.

A.1. Introduction — Evaluating Images Automatically Generated from Text

This is a survey for evaluating images for my Master's Thesis in computer science at the Norwegian University of Science and Technology (NTNU). All images have been generated from text using machine-learning models. The models that have generated the images are far from fully trained. Therefore, please note that the images will not be realistic or of high quality. This survey serves as input to evaluate whether the proposed approach may have merit.

The survey is completely anonymous. However, the first section includes a few questions about experience within photography and visual arts.

It takes between 5 - 10 minutes to complete the survey.

Thank you so much for your contribution, it is highly appreciated!

A.2. Section 1 — Experience

Each question could be answered with a number in the range 1-5. A lower number signifies uninterested/inexperienced, while a higher value signifies very interested/experienced.

- How would you rate your experience with photography?
- To what degree have you done photography?
- How would you rate your knowledge in visual art?
- To what degree have you created visual art (not photography)? (e.g. painting, drawing, pottery)

- To what degree would you rate your knowledge of image composition?
- How interested are you in photography?
- How interested are you in visual arts?

A.3. Section 2 — Comparing Image Sets

This section provides image sets containing four images each. All images are generated from the provided text description using machine learning models. The images might not correspond well to the text description, but please choose the image set you believe corresponds best. Additionally, there is a text box provided for each question where you can fill in any thoughts you might have about why you chose one image set over the other.

Each question was supplied with three options: “Image set 1”, “Image set 2”, and “Both image sets equally fit the description.”

- Which of the image sets above fit the description “A skier with a red jacket on going down the side of a mountain.” best?
- Which of the image sets above fit the description “The pizza is cheesy with pepperoni for the topping.” best?
- Which of the image sets above fit the description “Boats at the dock with a city backdrop.” best?
- Which of the image sets above fit the description “Brown horses are running on a green field.” best?
- Which of the image sets above fit the description “A bunch of vehicles that are in the street.” best?
- Which of the image sets above fit the description “A lamp with a shade sitting on top of an older model television.” best?
- Which of the image sets above fit the description “A stationary train with the door wide open.” best?
- Which of the image sets above fit the description “A street that goes on to a high way with the light on red.” best?
- Which of the image sets above fit the description “A large white teddy bear sitting on top of an SUV.” best?

A.4. Section 3 — Which Image Sets Are the Most Interesting?

This section contains new image sets from the previous section. The images are generated from the text provided in the question. Please choose which image set is the most interesting out of the two. This could be because they include a lot of color, makes you think, have interesting patterns, match the text in a different way than expected, or anything else.

Each question is again supplied with an optional short description where you can write down why you chose either of the image sets (or neither of them).

Each question was supplied with four options: “Image set 1”, “Image set 2”, “Both of them are equally interesting.” and “Neither are any interesting.”

- “I love going to art galleries and seeing all the beautiful and strange creations they have there.” Considering the images above, which set is more interesting?
- “It is utterly terrific that he got accepted to that school.” Considering the images above, which set is more interesting?
- “The weather has been really horrible these last few days.” Considering the images above, which set is more interesting?
- “Today we are going to do some shopping for Lucy’s prom.” Considering the images above, which set is more interesting?
- “We are about five minutes late, I’m afraid.” Considering the images above, which set is more interesting?

A.5. Closing Questions

This section contains optional additional questions about the images you have seen. They do not need to be answered, but any answer is highly appreciated.

- What do you think of the quality of the images you have been presented with?
  - “They seem very abstract and surreal.”
  - “Warped”
  - “Generally, they seem to be very artificial and I think they would benefit from more time spent training the model. A lot of the pictures (particularly the second in each pair) were unrecognisable.”
  - “Creepy as hecc”
  - “The images do not accurately describe the descriptions presented, although the designs are interesting.”

  - “Most come nowhere near communicating the intended text description were they to stand alone. I only begin to see connections by reading the supporting text. It is interesting to see that some are starting to take forms from text.”
  - “Quite low”
  - “Poorly. Did not understand any of the pictures really”
  - “Bad quality. Couldnt really see what was happening....”
  - “I didn’t think they matched very well the description, but some generated interesting art which kinda had elements of what you wanted to generate”
  - “The content had no meaning to me”
  - “It is not perfect, but it was surprisingly good. I expected it to be lower.”
  - “They looks like partly figurative art, not very much like the text descriptions”
  - “Quite poor”
  - “Quite poor. But I don’t know what to expect.”
  - “Some are able to depict what the sentence entails, but most seem to generate scrambled images with some correlation to the words in the given input sentence.”
  - “Good quality on most of the photos”
  - “Low”
  - “Variable and impressive”
  - “Depends on the definition of quality. They are high quality abstract images where you can (sometimes) see hints of the text in the image without beeing obvious. But low quality in what the text ask specifically.”
  - “They look cool but I’m not sure what they are supposed to be.”
  - “they make me feel like i’m having a stroke”
  - “Kinda hard to see what they are supposed to be” without context, but very interesting and almost creepy to see”

- Does any (one or several) image set(s) stand out to you?
  - “No.”
  - “The horses”
  - “The first image of each pair seemed more vibrant and meaningful compared to the second.”
  - “The old teddy bear one when zoomed in looks like a waterbear and the setting has a black mirroresque vibe to it”
  - “Not really.”
  - “Yeah, overall I found that when I was asked to consider images based on interest/aesthetics, I preferred image set 1 consistently over image set 2.”

- "I think it was image set 1of the "it's terrific that he got accepted into that school". The description had the word "terrific" in it but the images looked absolutely horrifying and creepy, which are the total opposite of terrific."
- "Not really"
- "No"
- "Not really. I suspect that they are too random to be memorable."
- "No."
- "No"
- "Not that I can think of"
- "no"
- "I think I was presented with so much abstract odd images that none really stood out. They became a blur near the end"
- "Yes, some of them looks really interesting."

B. Generated Images

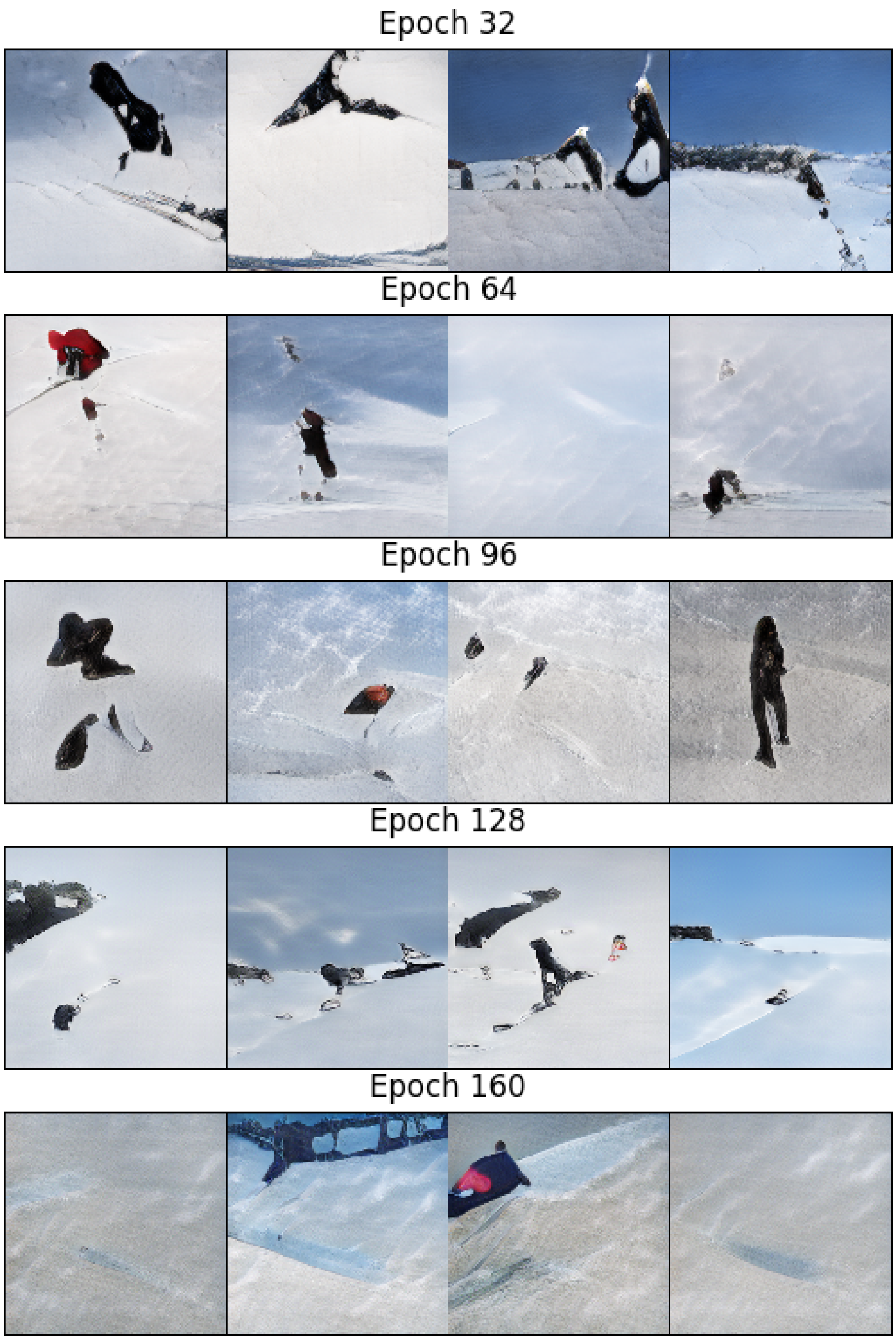


Figure B.1.: New model generated images: “A skier with a red jacked on going down the side of a mountain.”

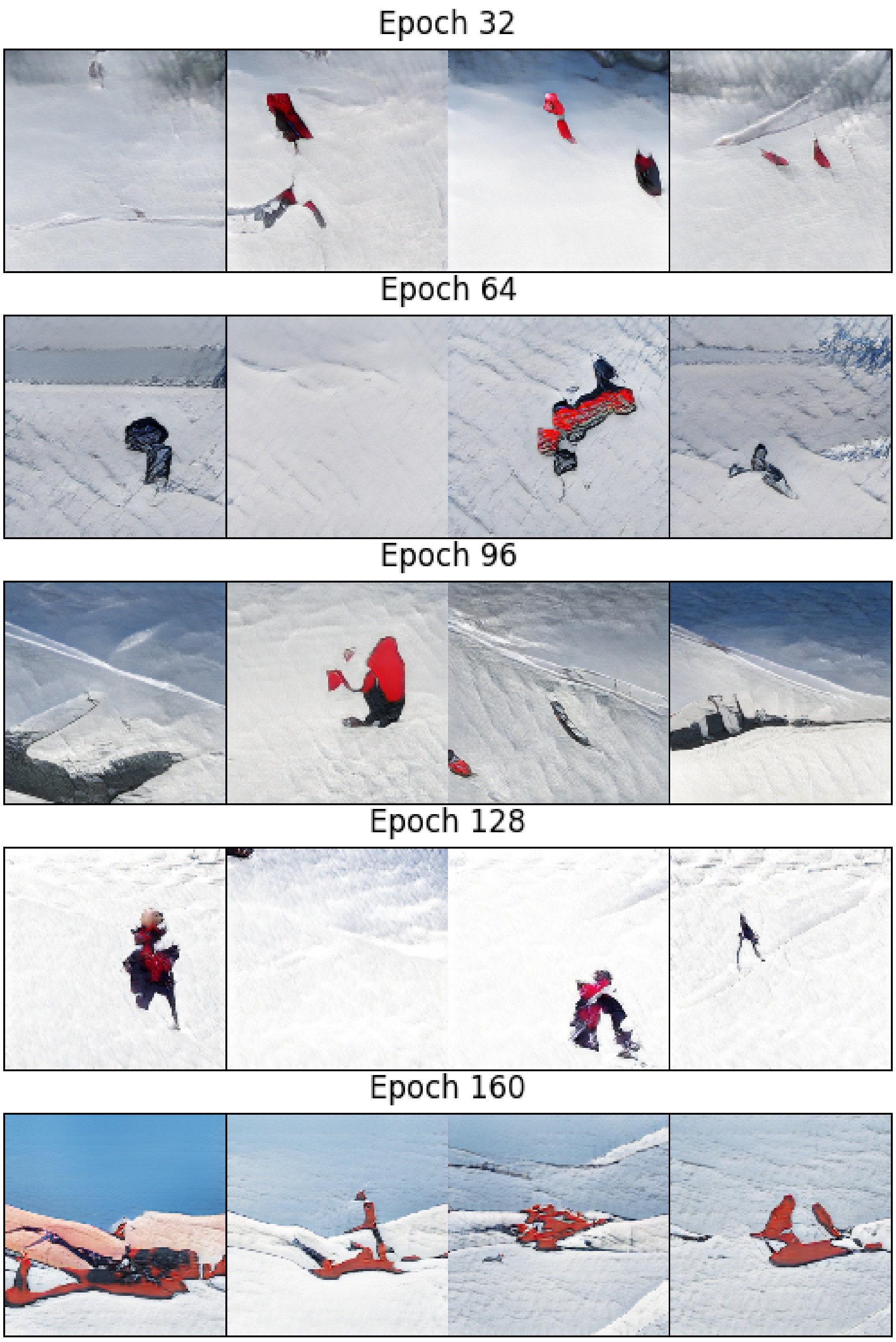


Figure B.2.: Original model generated images: “A skier with a red jacked on going down the side of a mountain.”

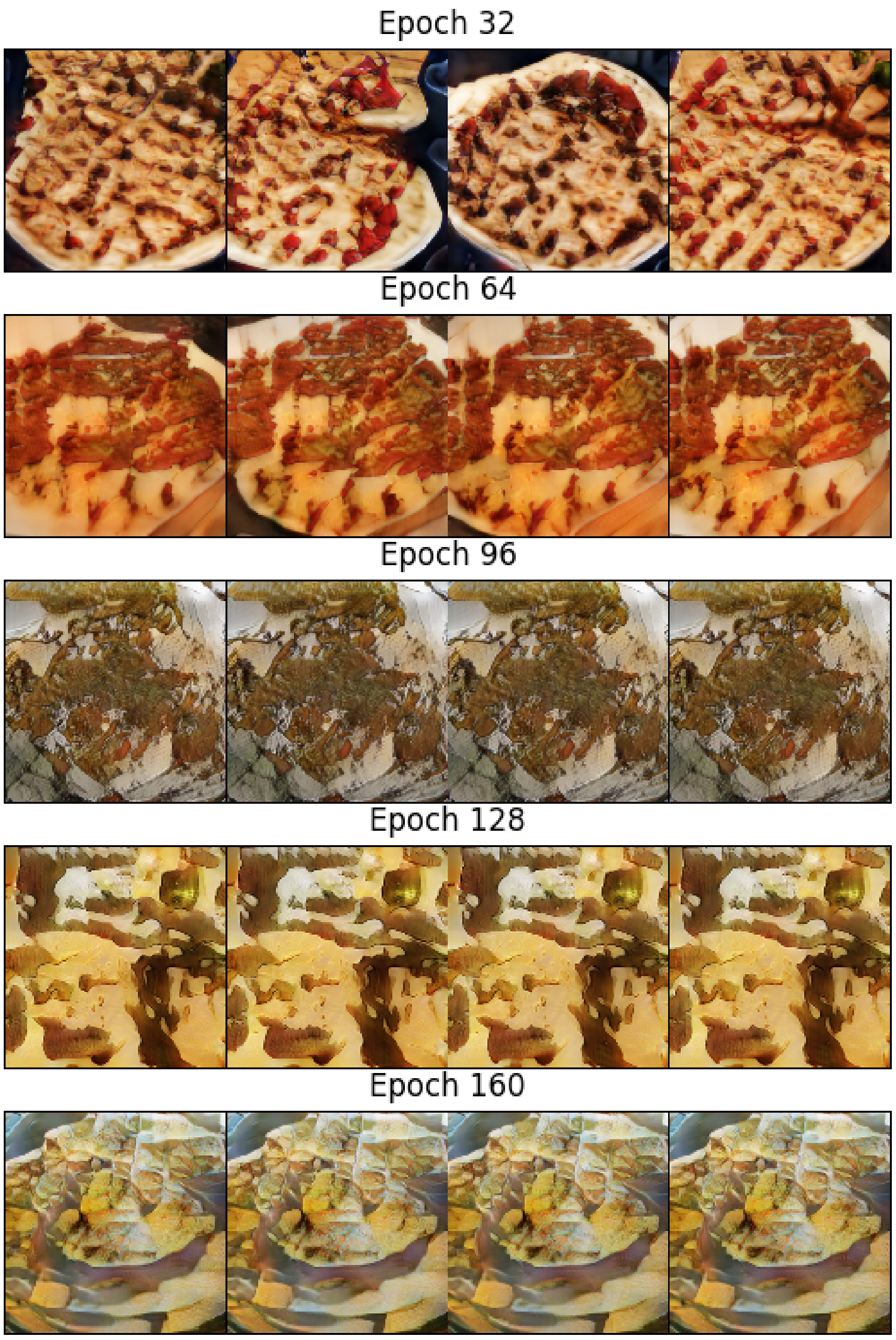


Figure B.3.: New model generated images: “The pizza is cheesy with pepperoni for the topping.”

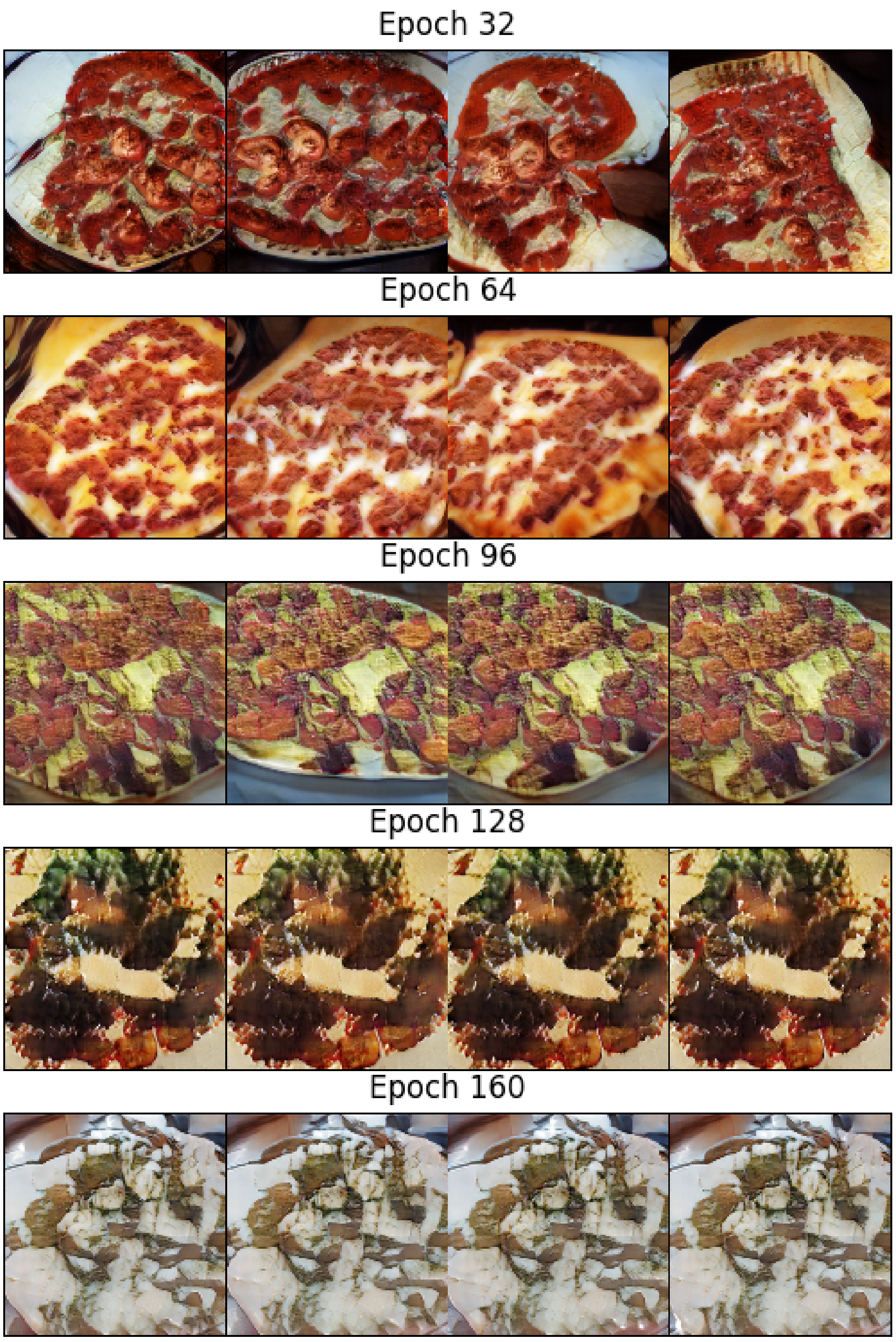


Figure B.4.: Original model generated images: “The pizza is cheesy with pepperoni for the topping.”

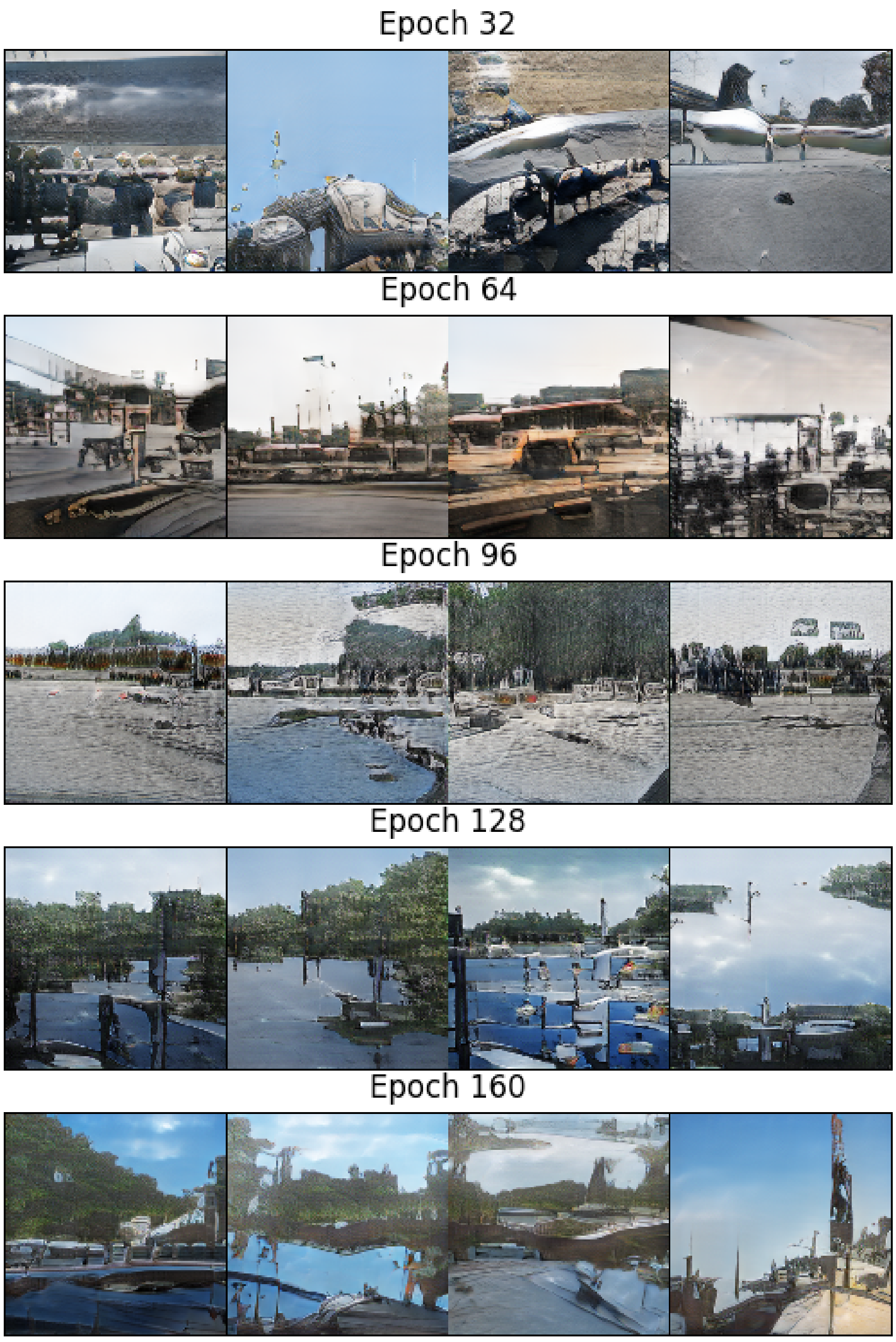


Figure B.5.: New model generated images: “Boats at the dock with a city backdrop.”

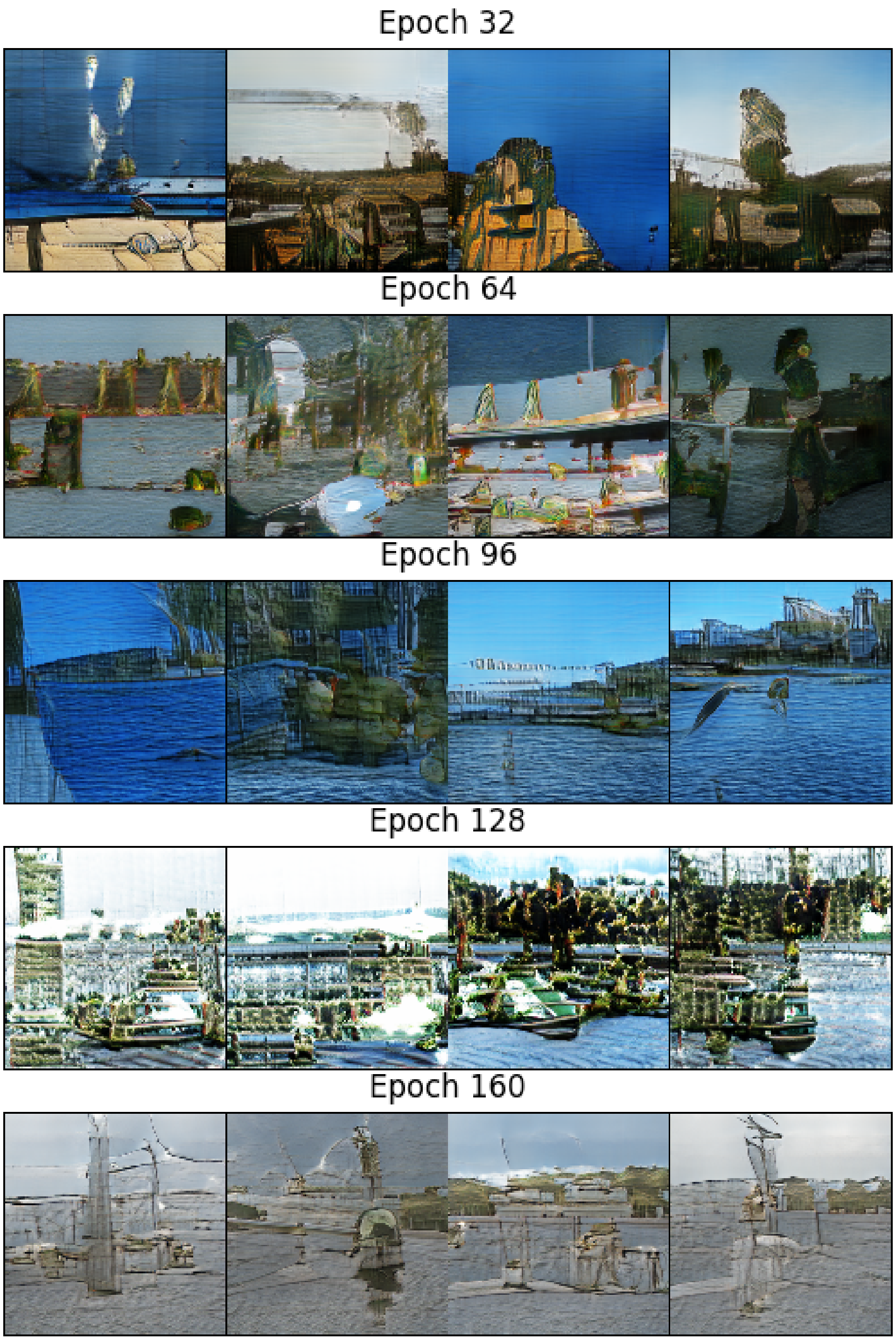


Figure B.6.: Original model generated images: “Boats at the dock with a city backdrop.”

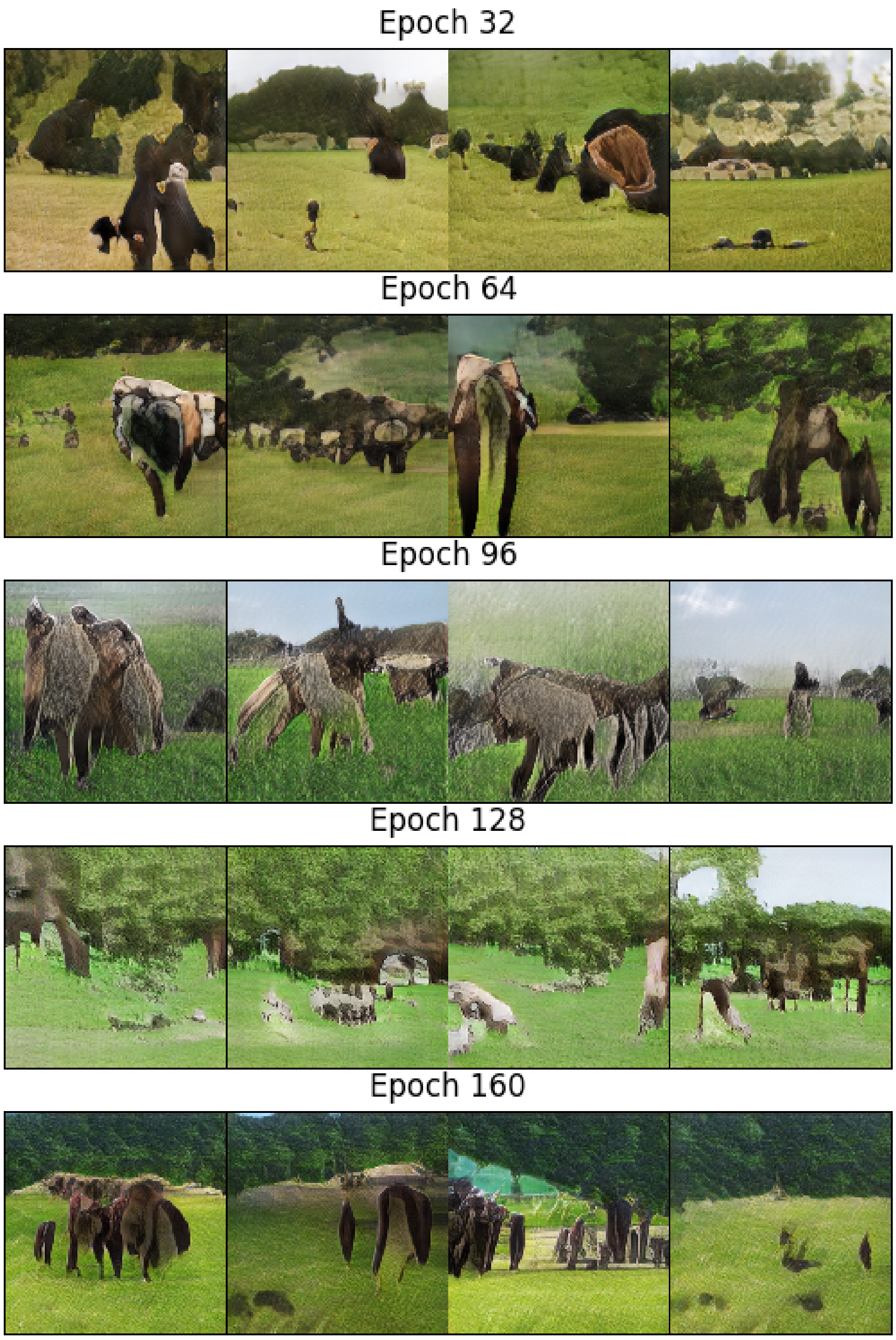


Figure B.7.: New model generated images: “Brown horses are running on a green field.”

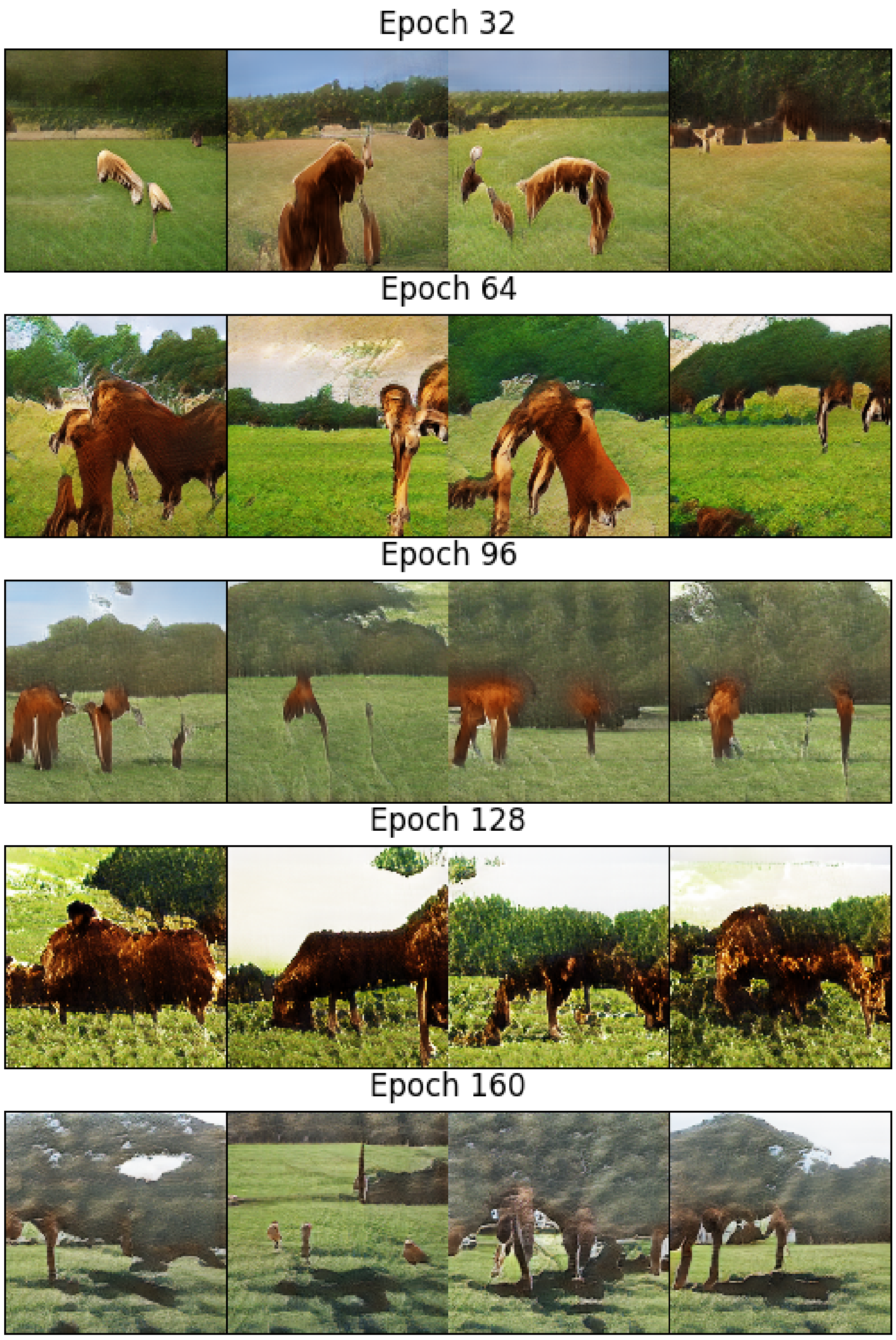


Figure B.8.: Original model generated images: “Brown horses are running on a green field.”

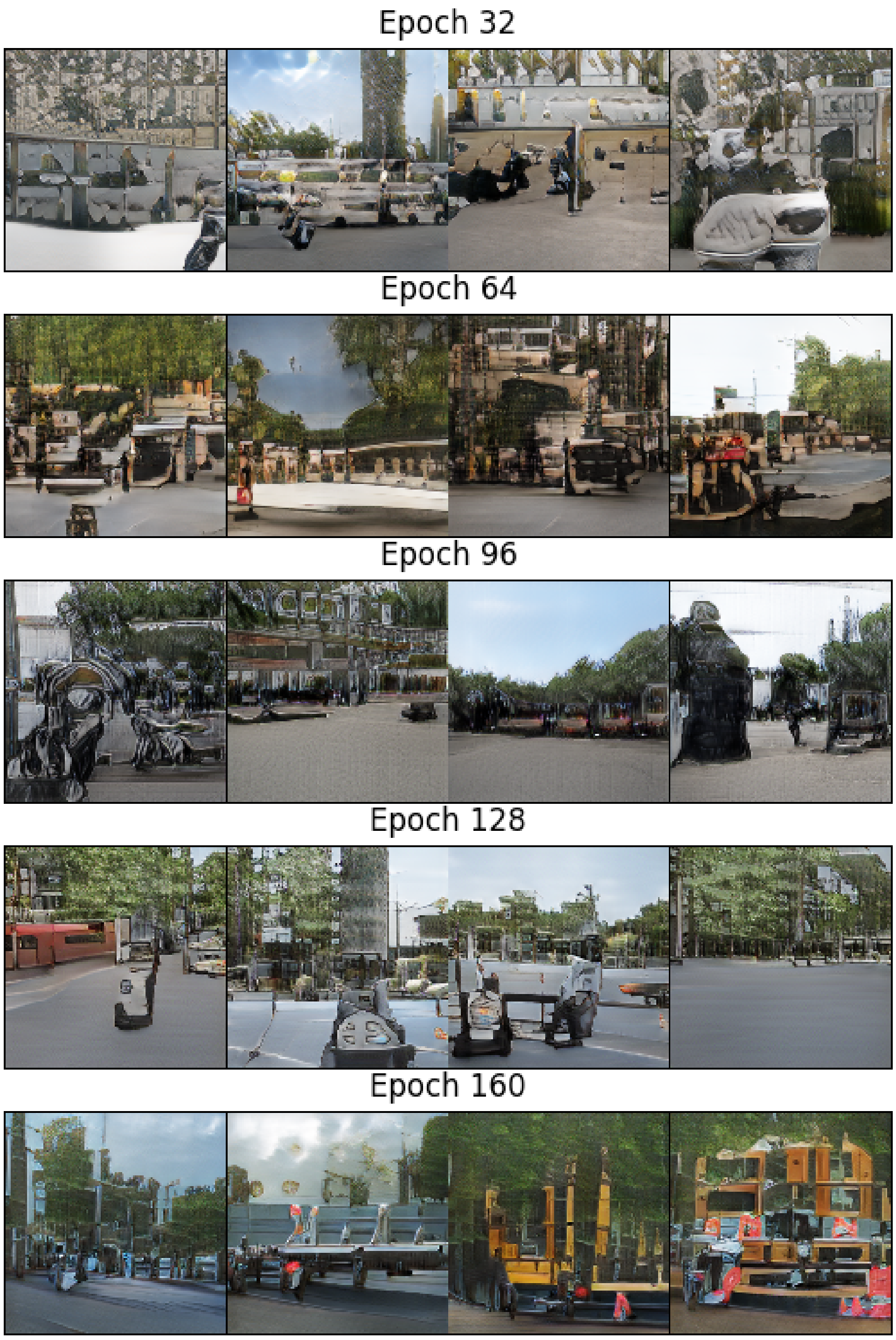


Figure B.9.: New model generated images: “A bunch of vehicles that are in the street11

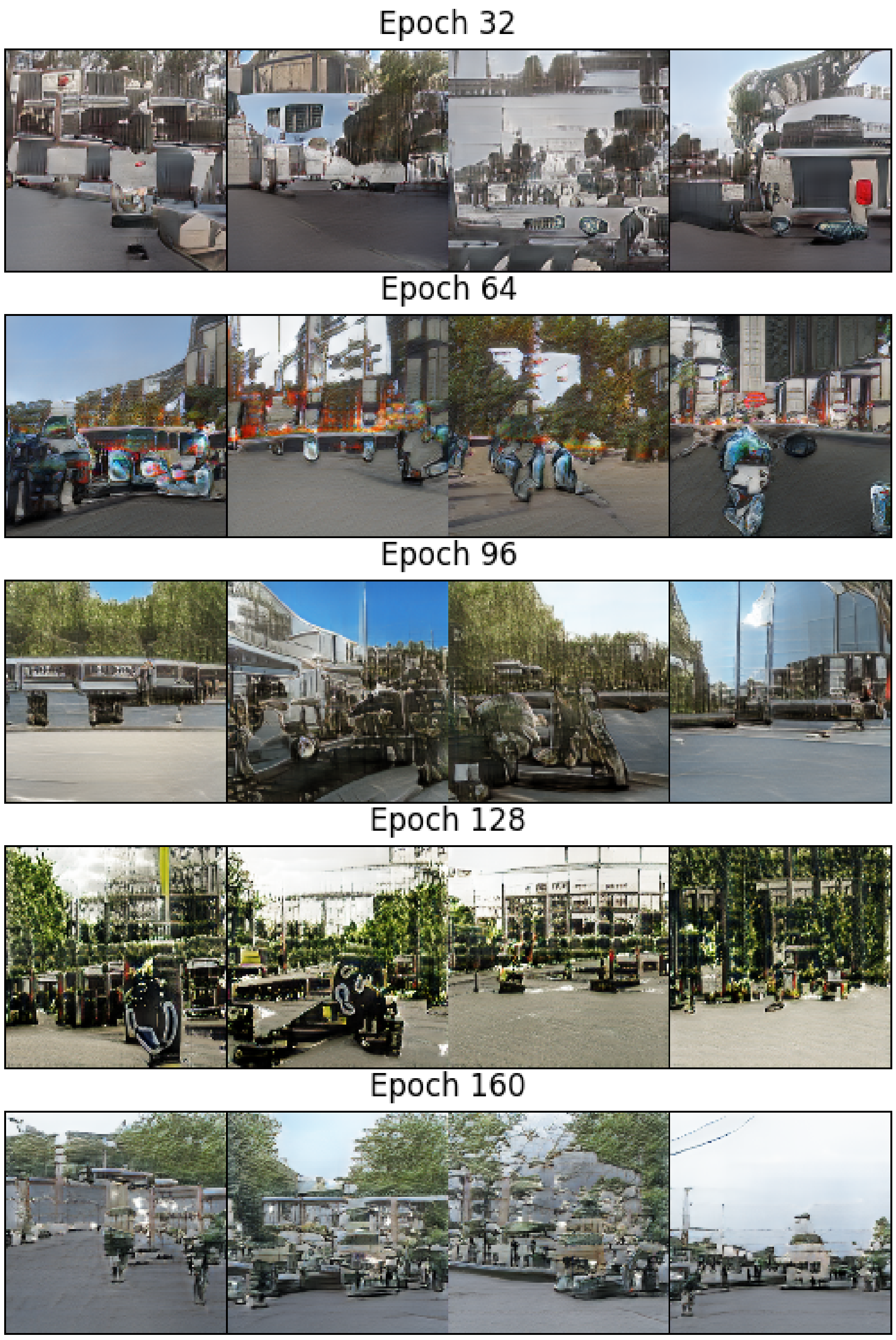


Figure B.10.: Original model generated images: “A bunch of vehicles that are in the street.”

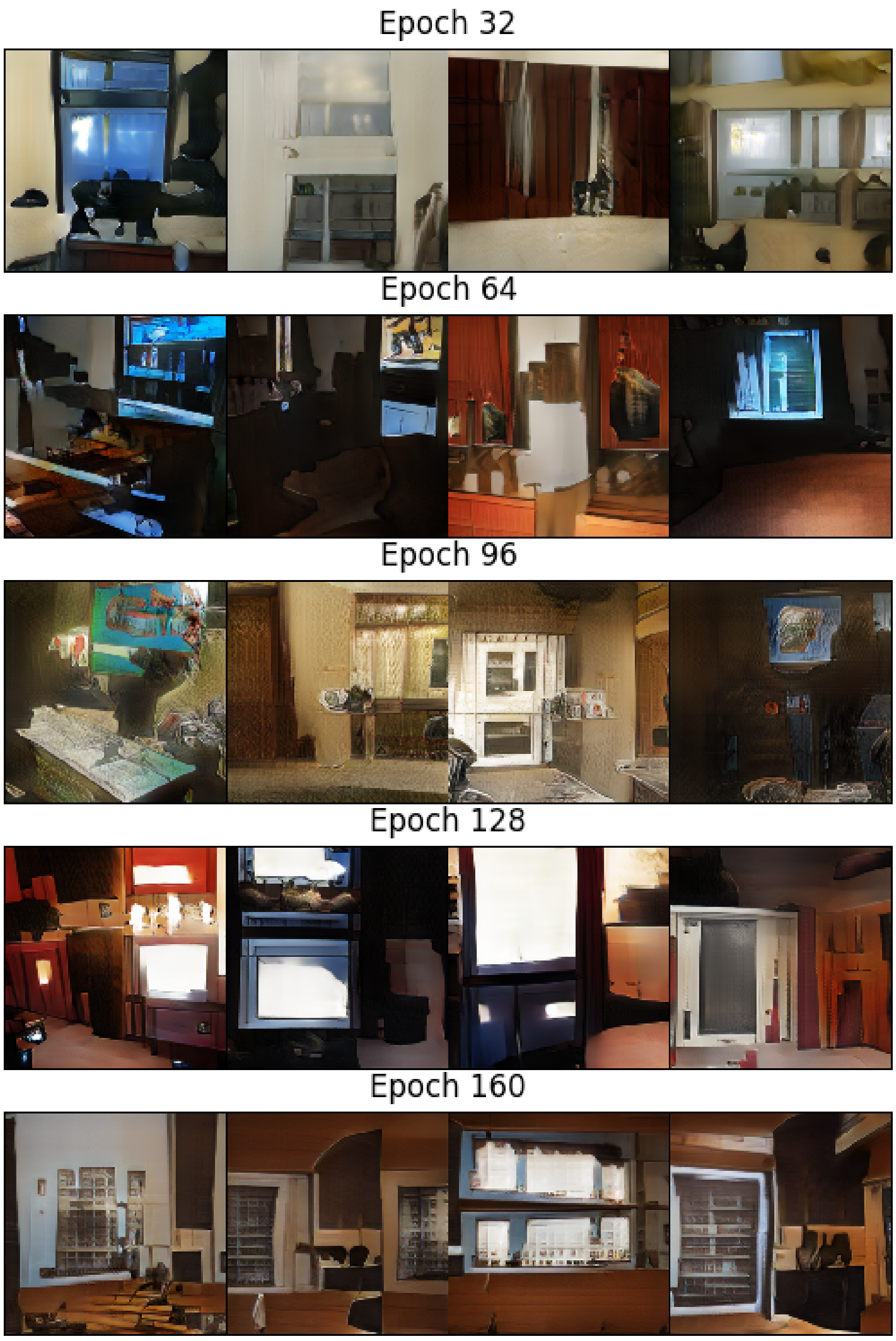


Figure B.11.: New model generated images: “A lamp with a shade sitting on top of an older model television.”

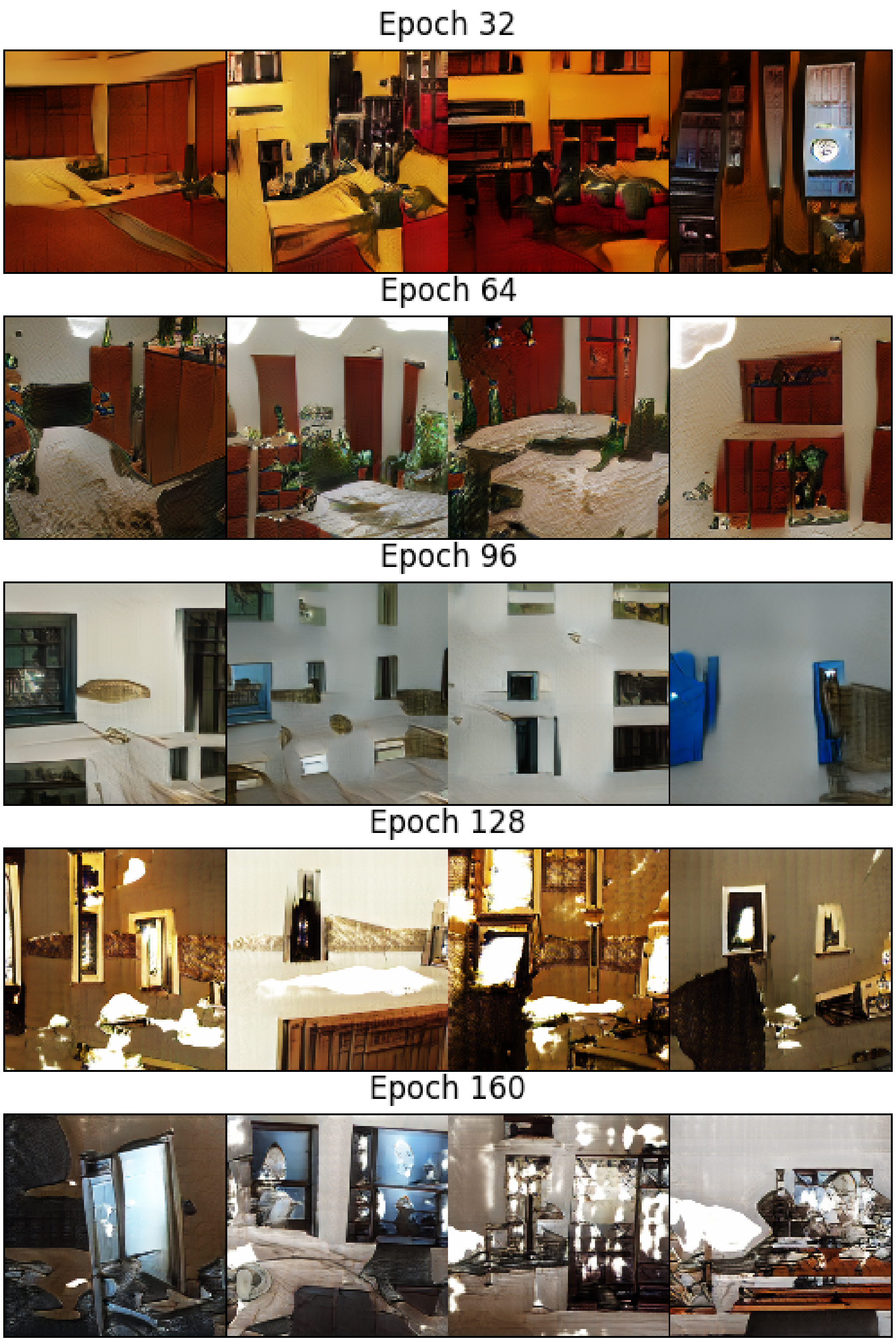


Figure B.12.: Original model generated images: “A lamp with a shade sitting on top of an older model television.”

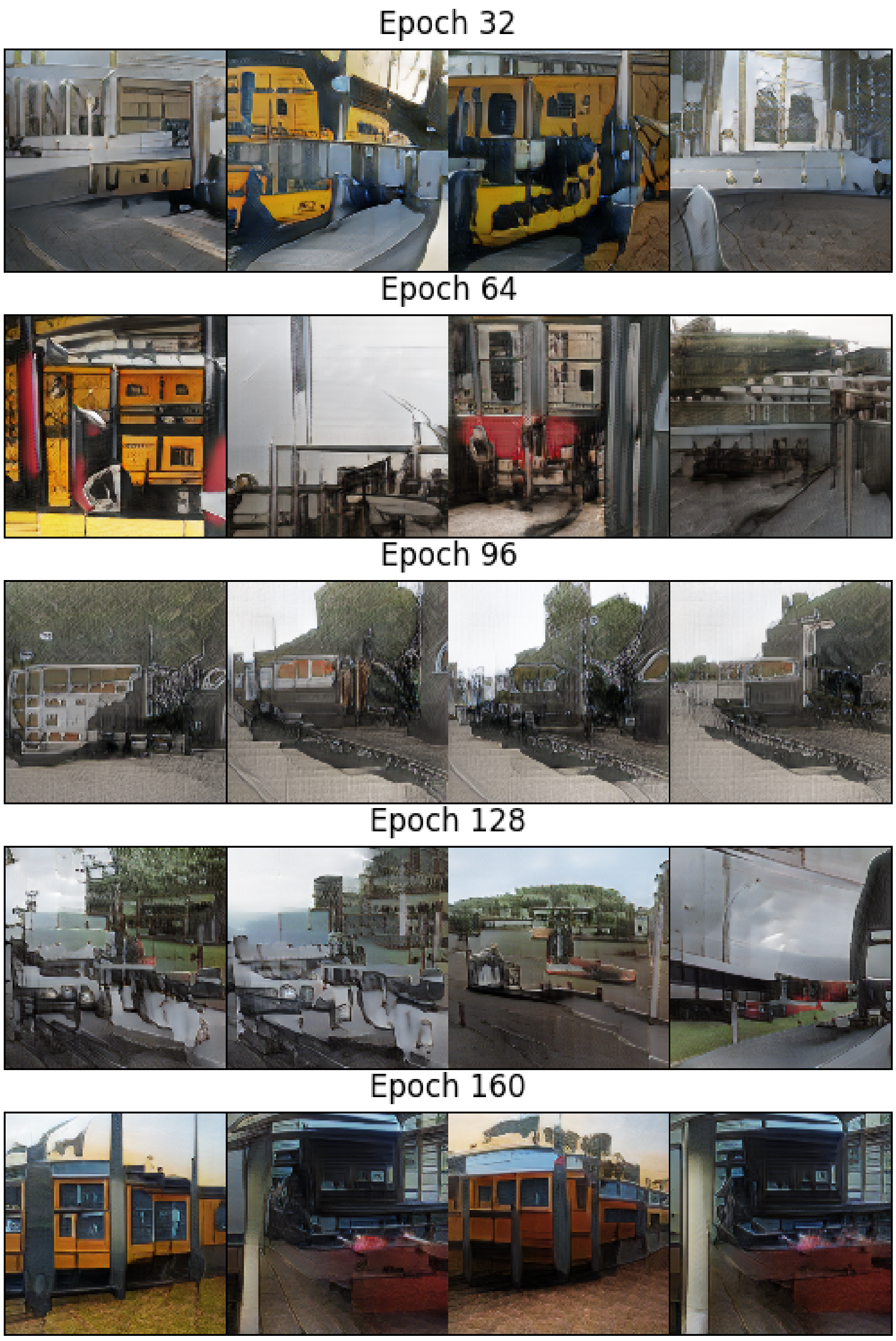


Figure B.13.: New model generated images: “A stationary train with the door wide open.”

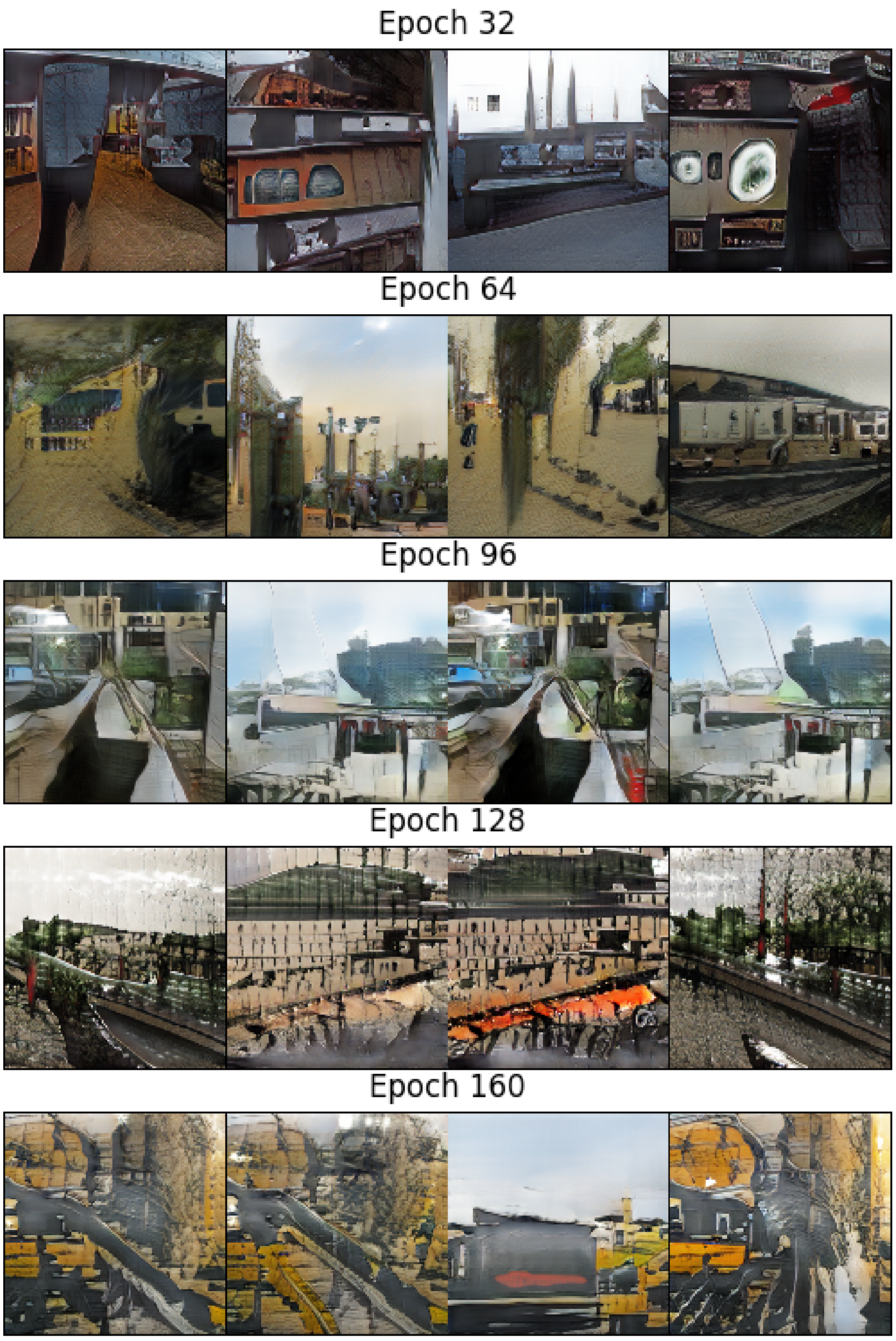


Figure B.14.: Original model generated images: “A stationary train with the door wide open.”

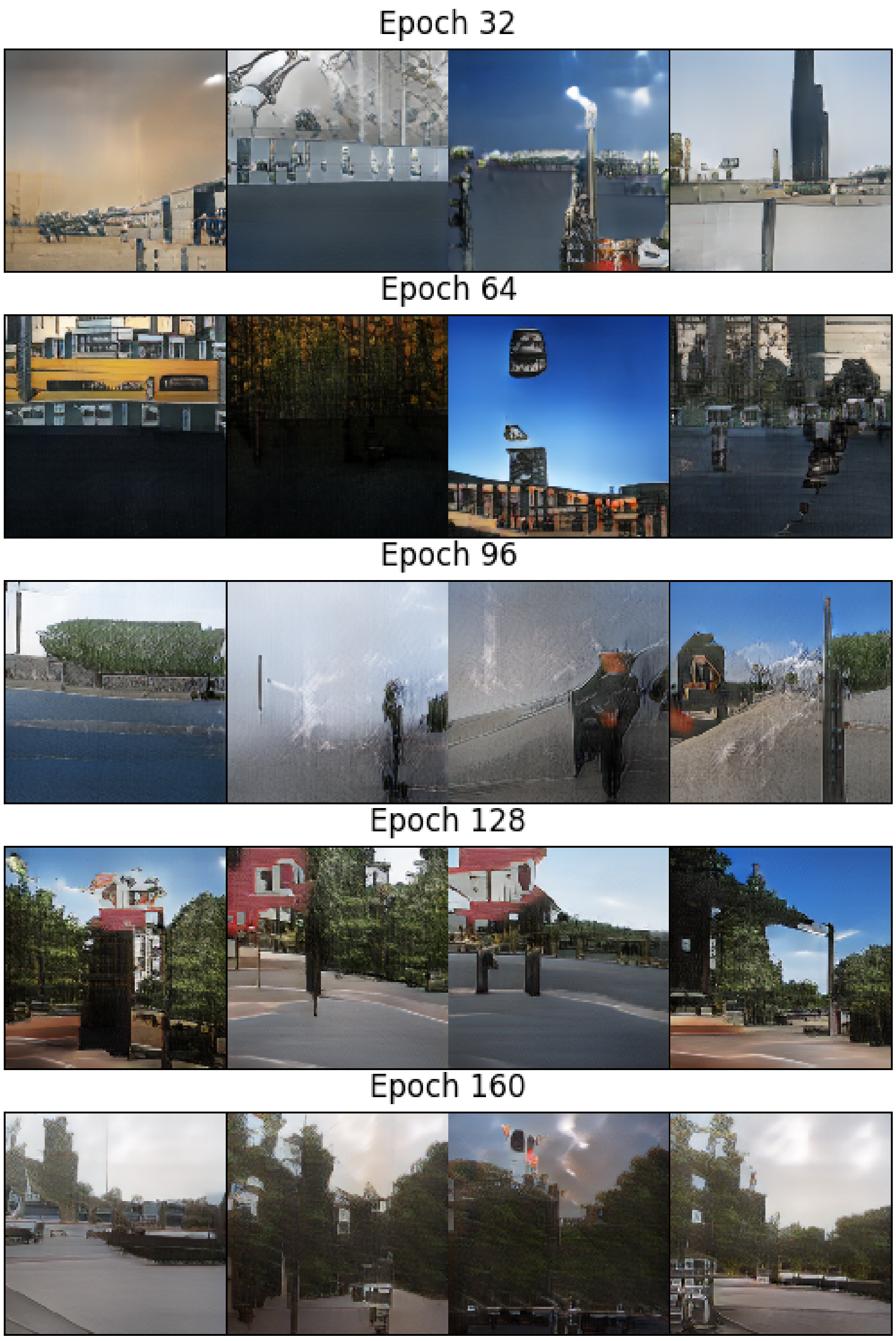


Figure B.15.: New model generated images: “A street that goes on to a high way with the light on red.”

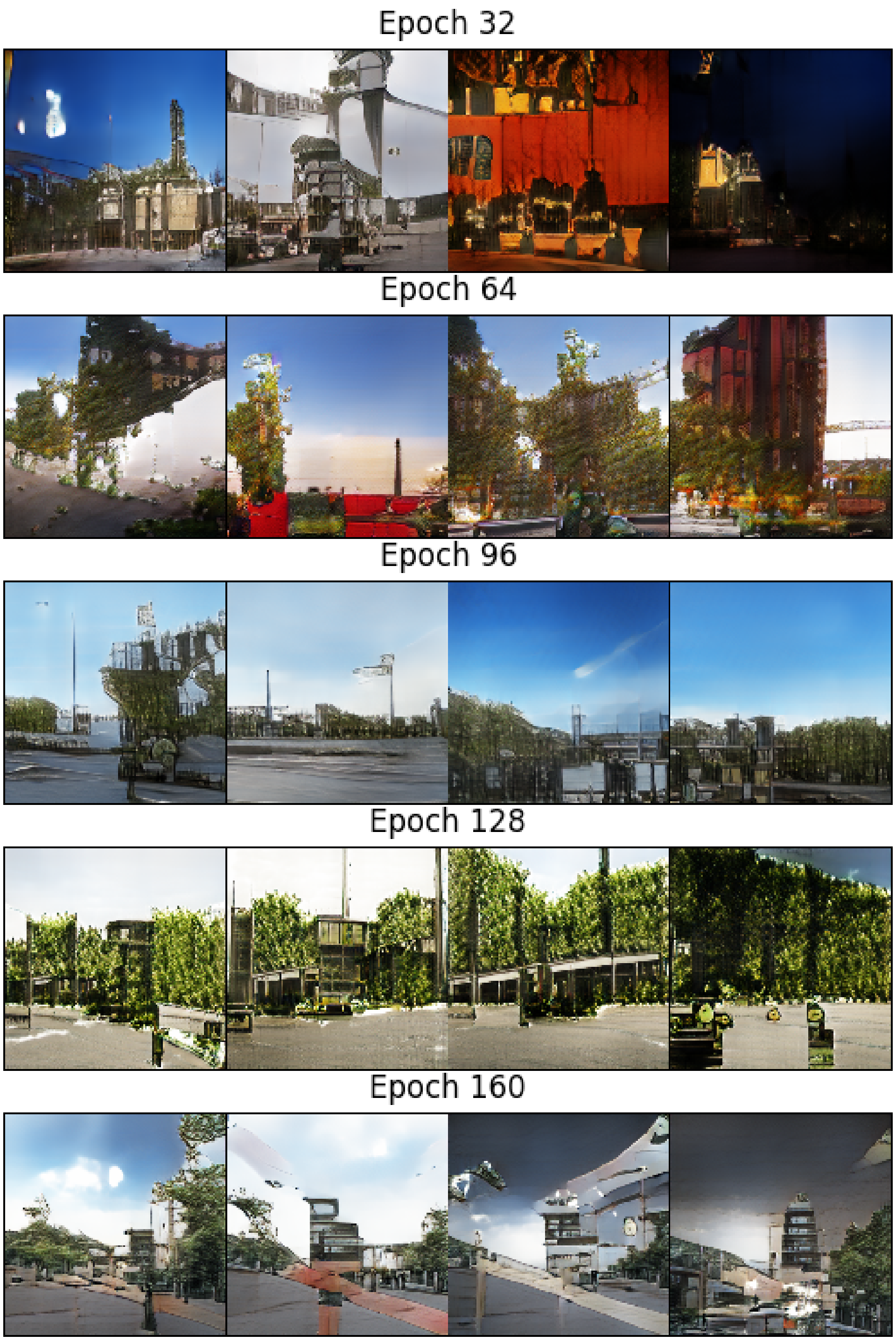


Figure B.16.: Original model generated images: “A street that goes on to a high way with the light on red.”

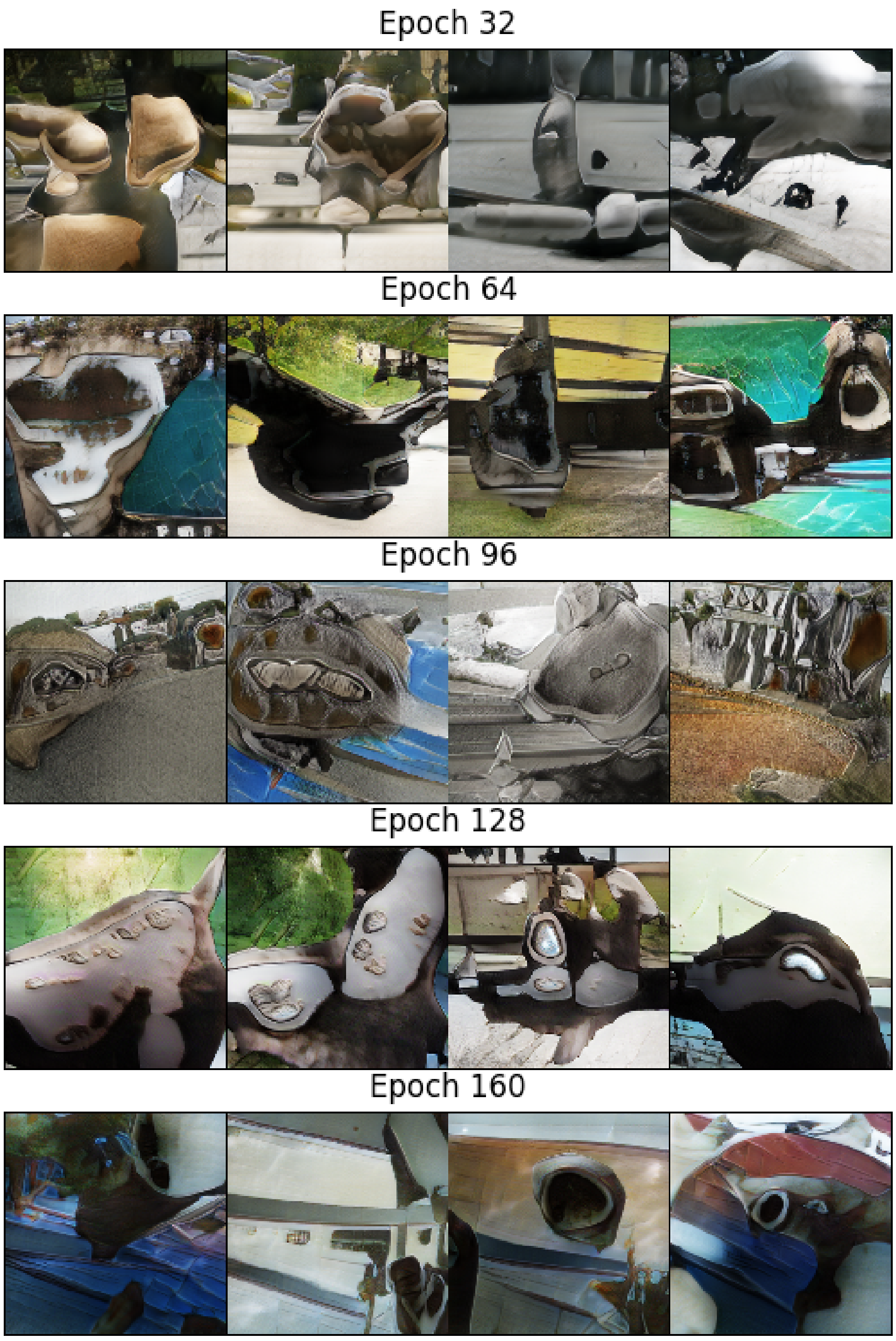


Figure B.17.: New model generated images: “A large white teddy bear sitting on top of an SUV.”

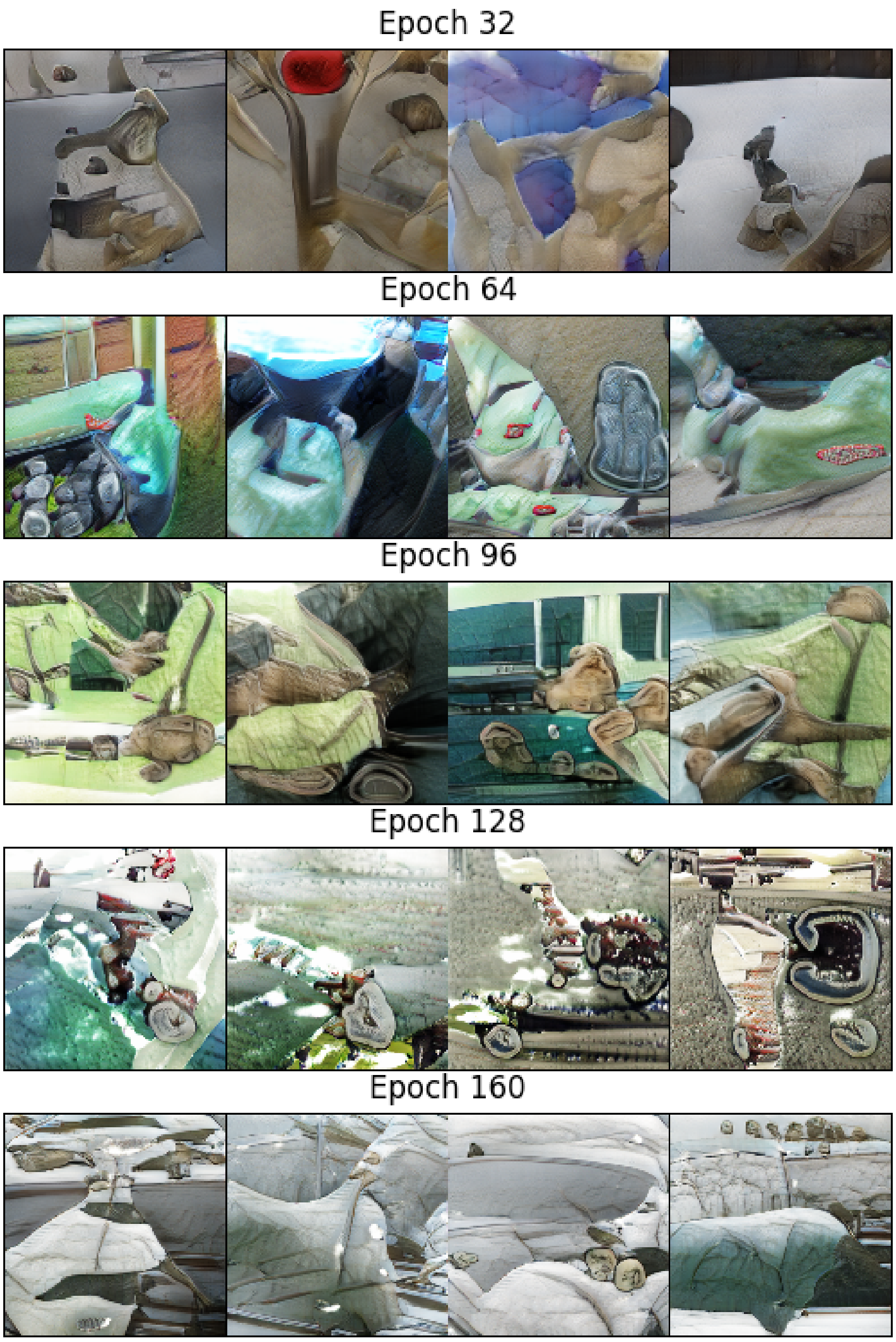


Figure B.18.: Original model generated images: "A large white teddy bear sitting on top of an SUV."

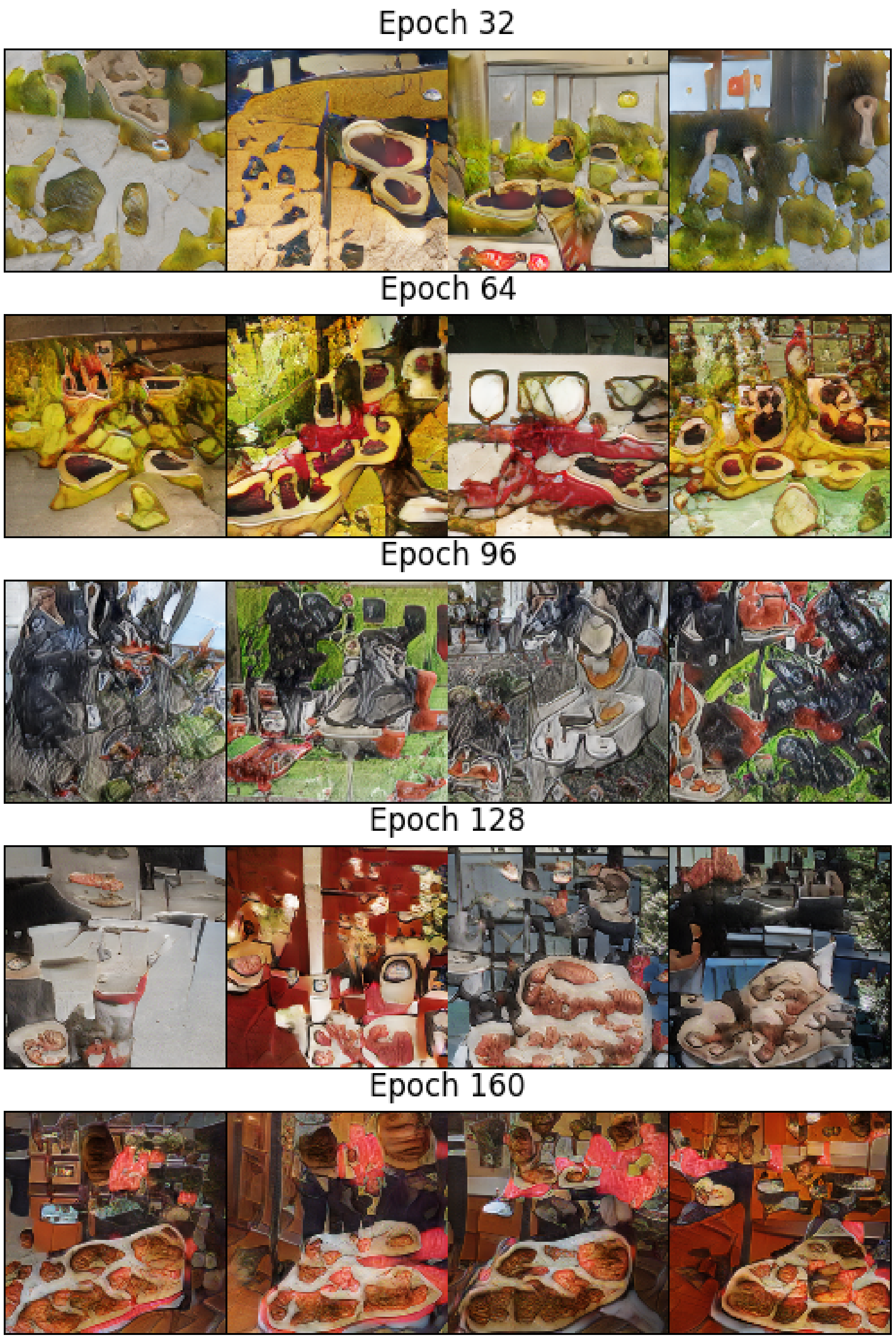


Figure B.19.: New model generated images: “I love going to art galleries and seeing all the beautiful and strange creations they have there.”

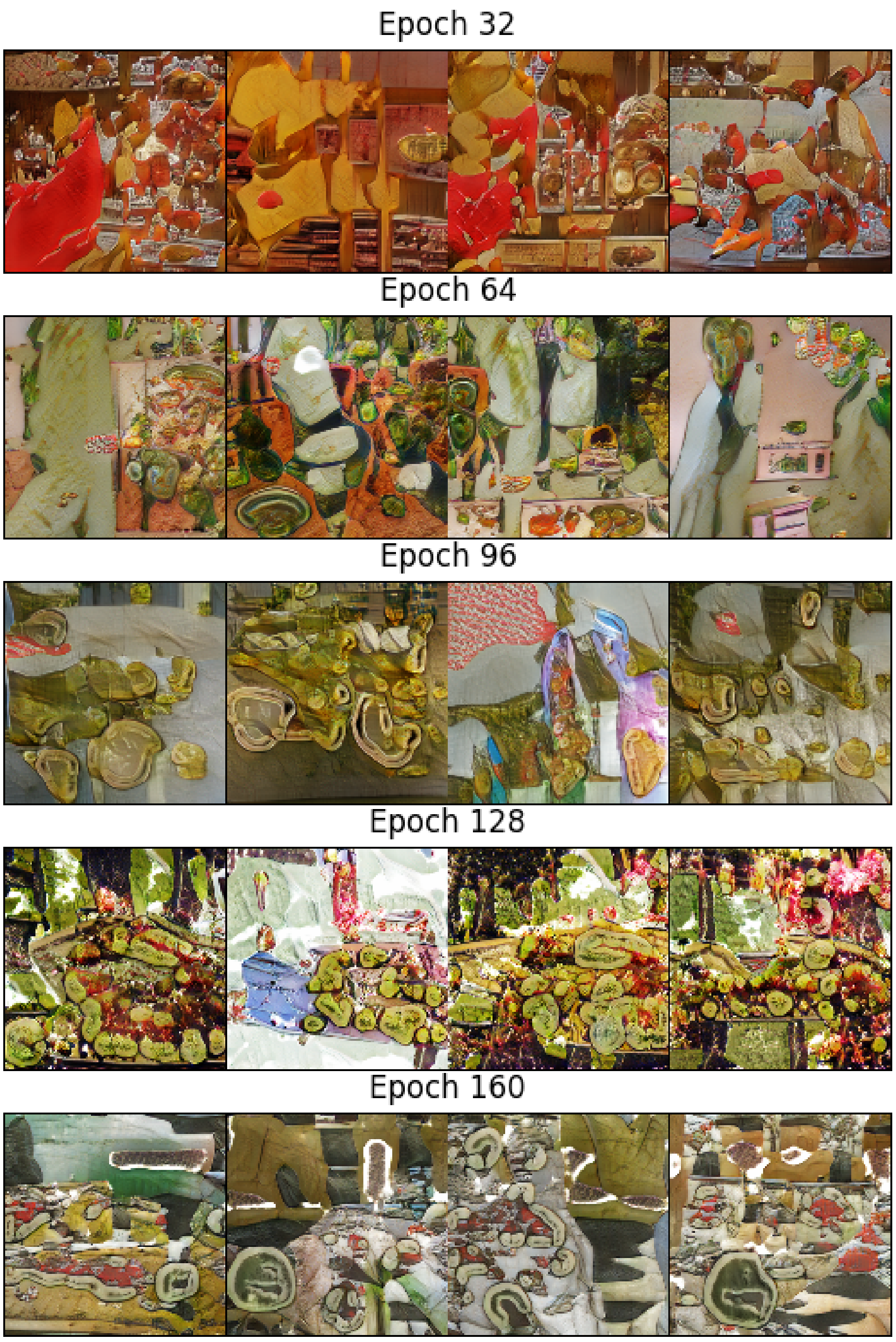


Figure B.20.: Original model generated images: “I love going to art galleries and seeing all the beautiful and strange creations they have there.”

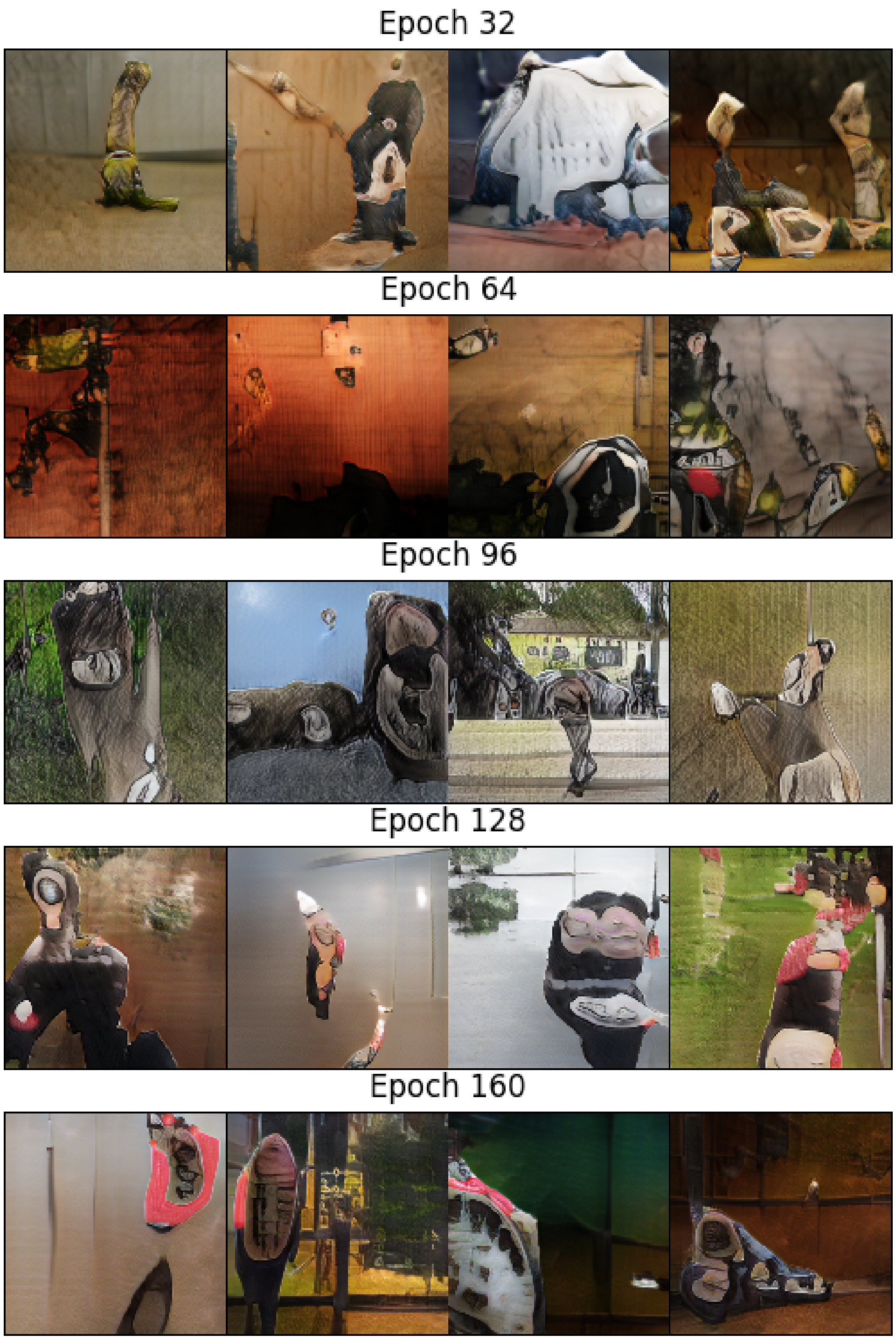


Figure B.21.: New model generated images: “It is utterly terrific that he got accepted to that school.”

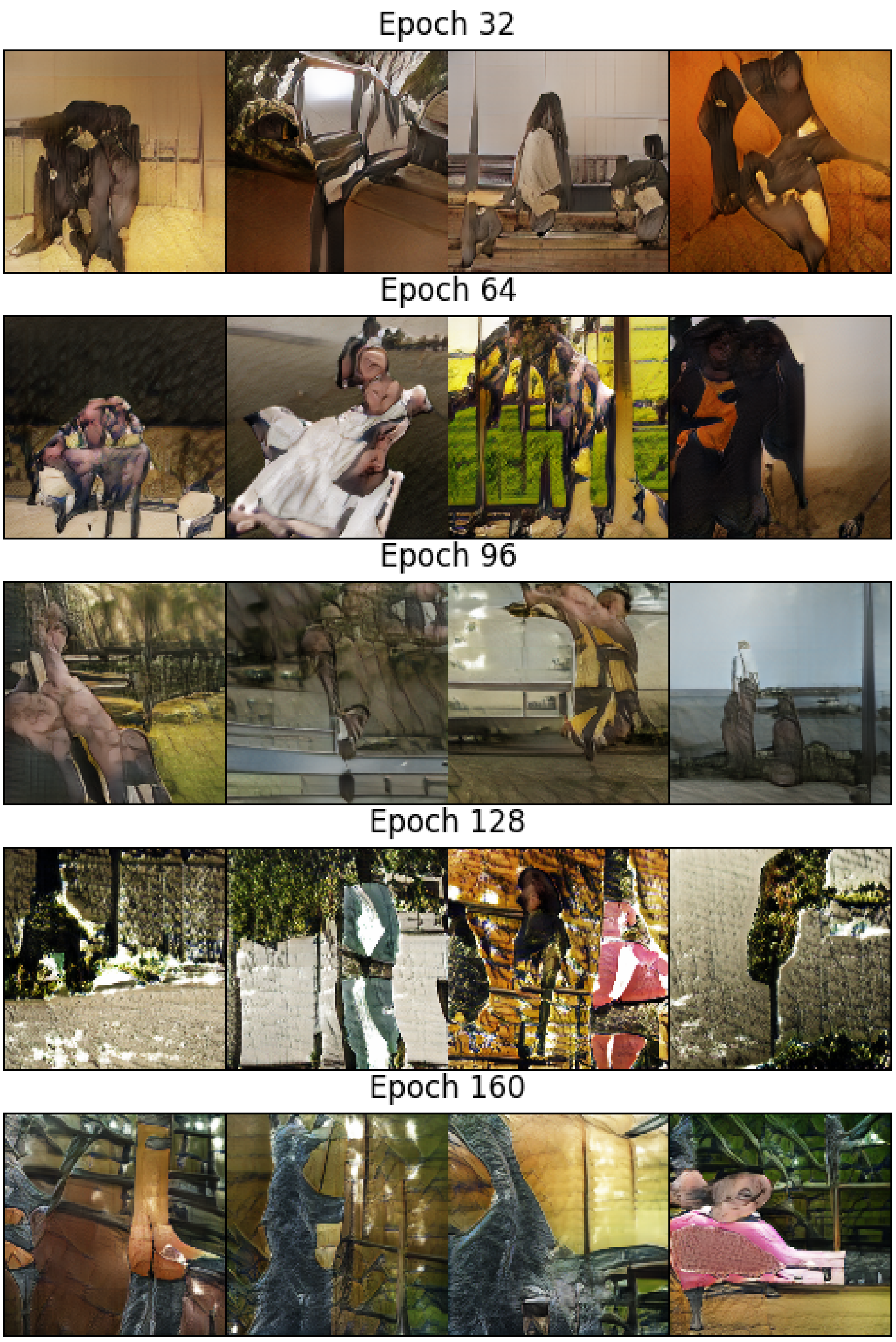


Figure B.22.: Original model generated images: “It is utterly terrific that he got accepted to that school.”

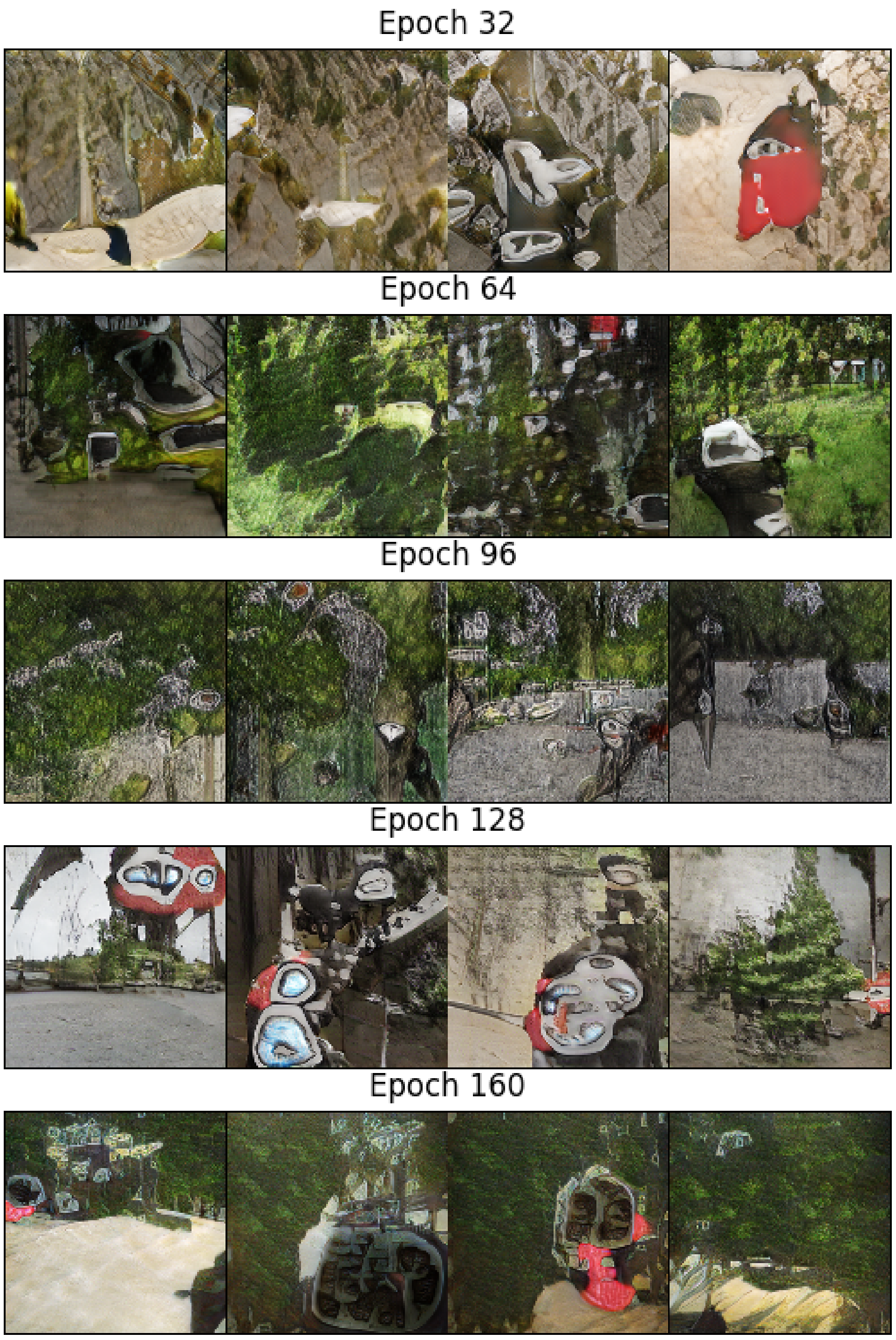


Figure B.23.: New model generated images: “The weather has been really horrible these last few days.”

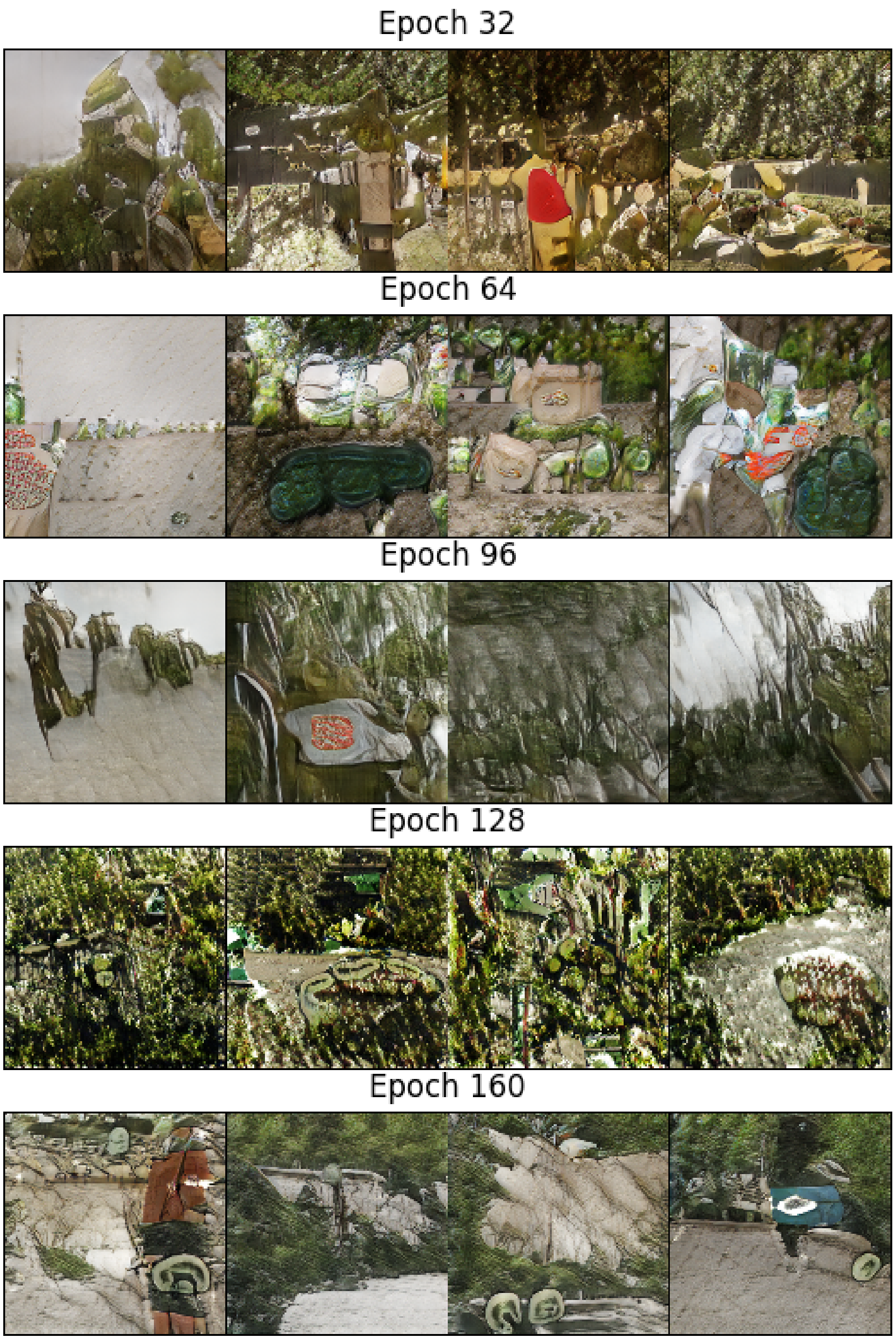


Figure B.24.: Original model generated images: “The weather has been really horrible these last few days.”

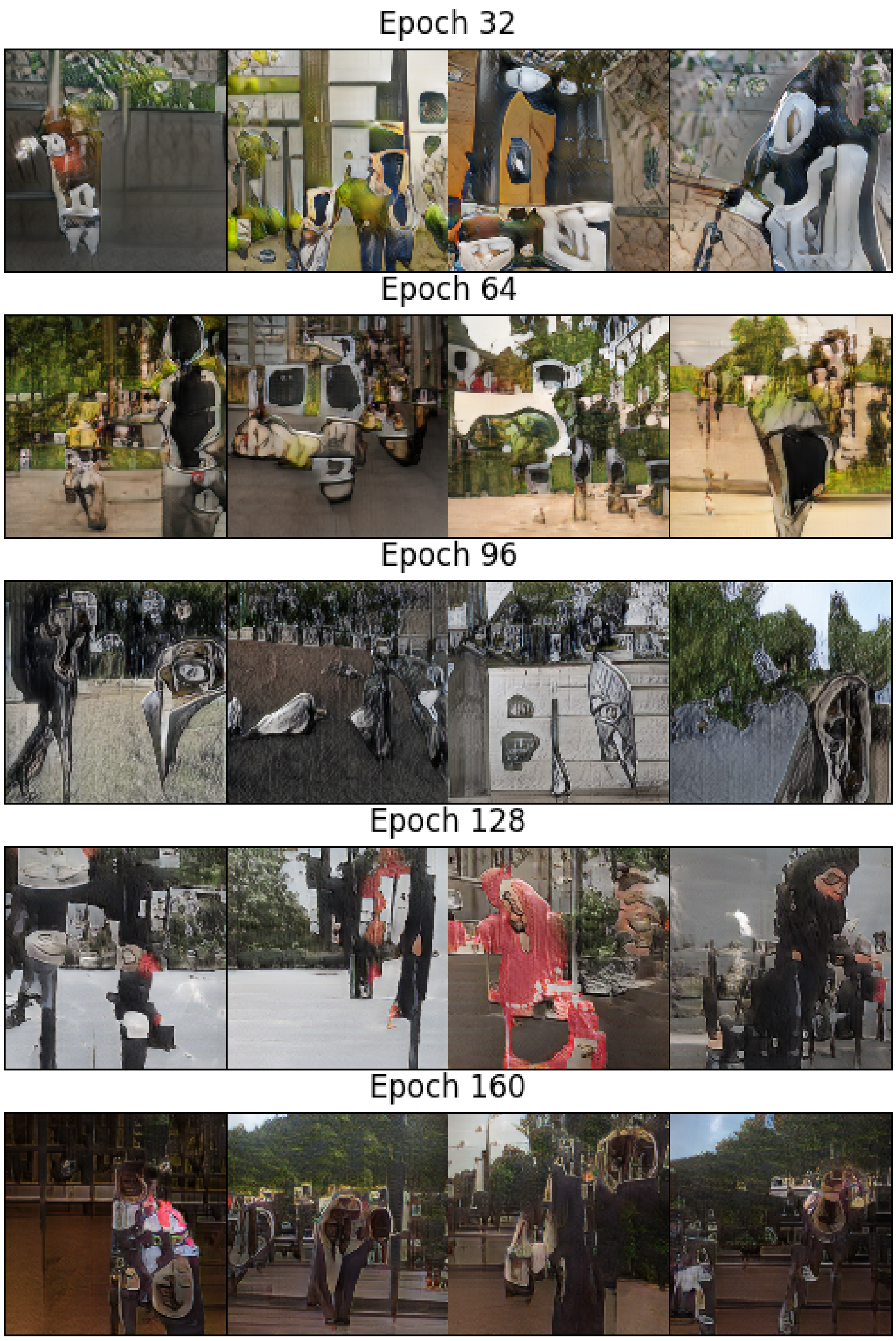


Figure B.25.: New model generated images: “Today we are going to do some shopping for Lucy’s prom.”

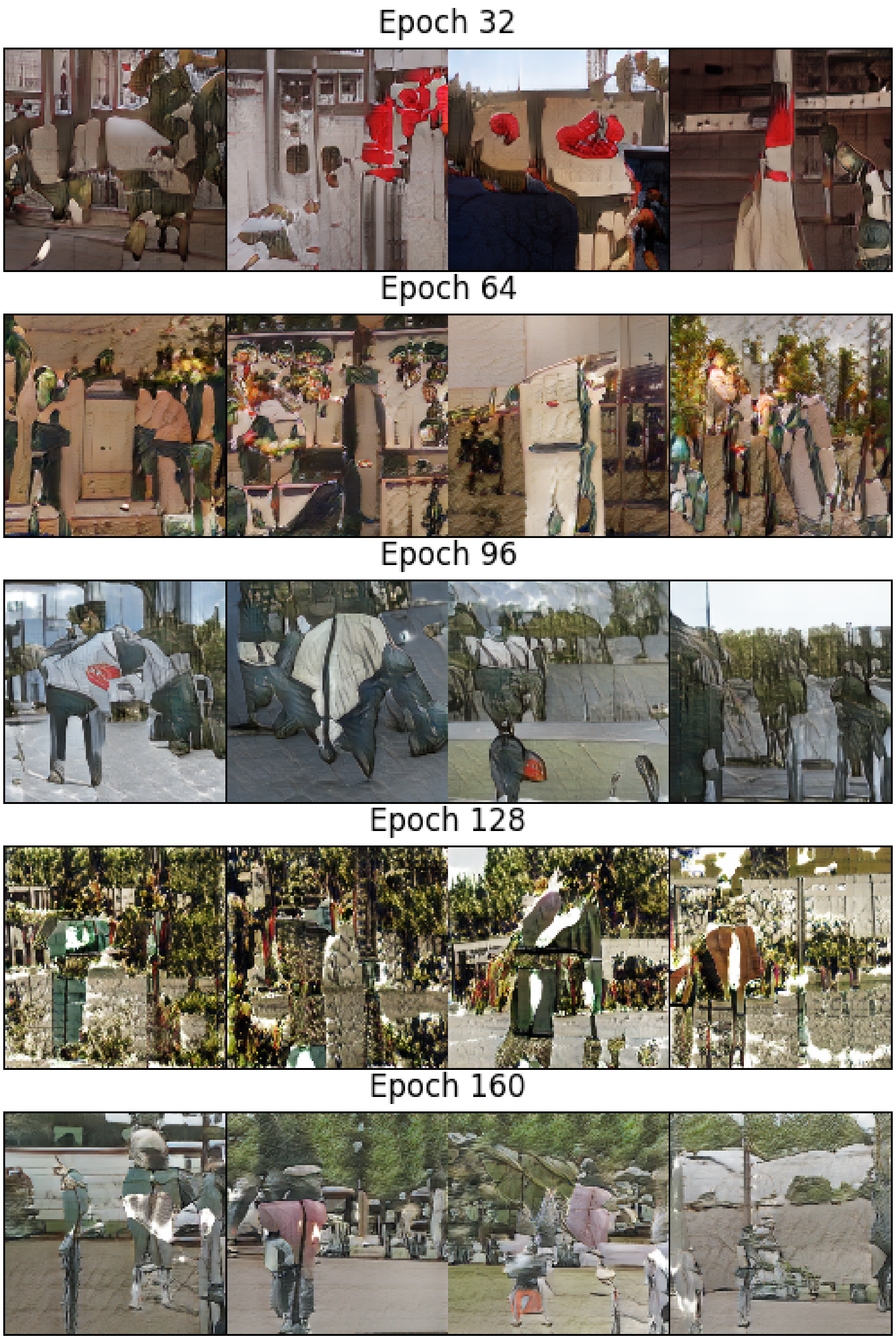


Figure B.26.: Original model generated images: “Today we are going to do some shopping for Lucy’s prom.”

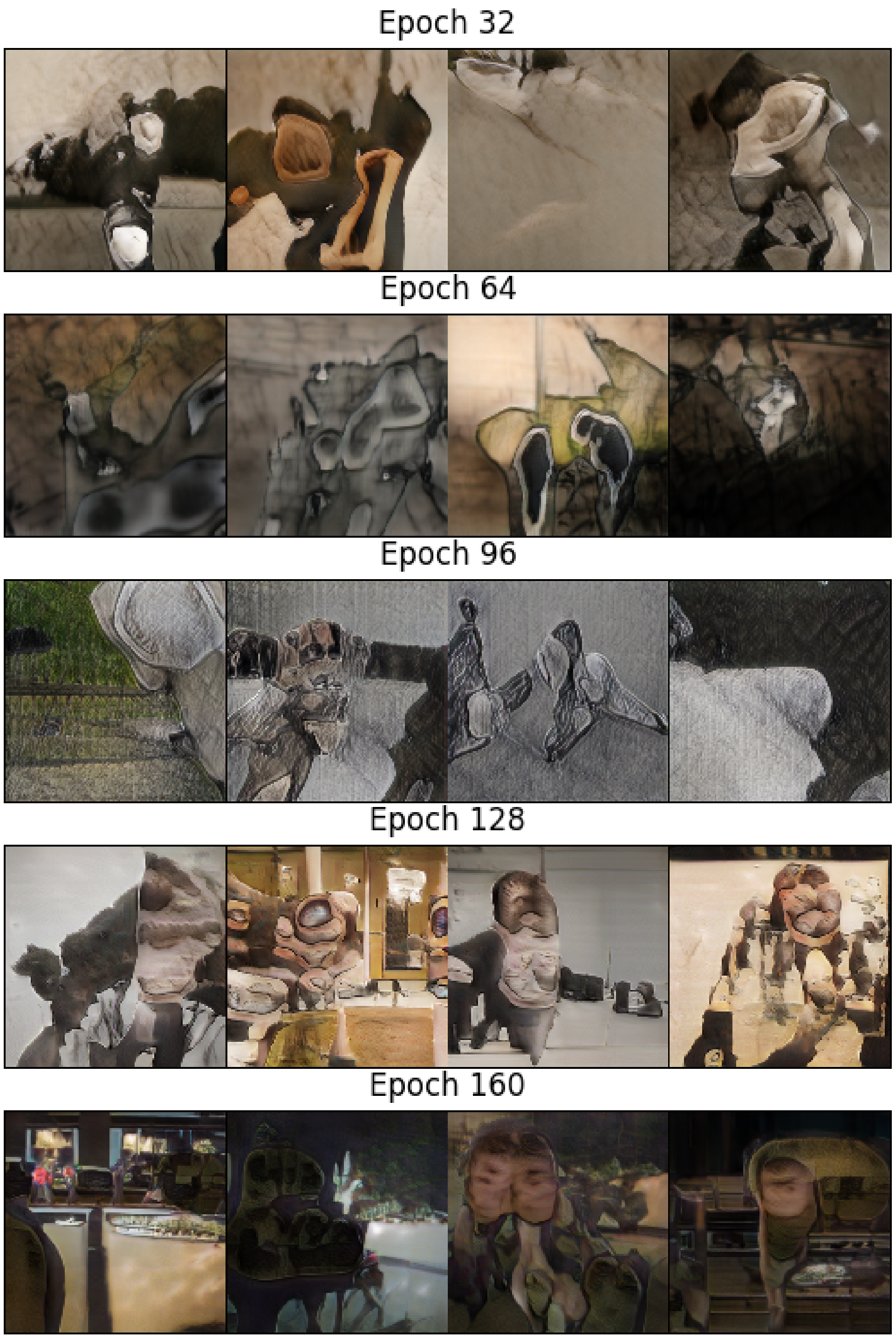


Figure B.27.: New model generated images: “We are about five minutes late, I’m afraid.”

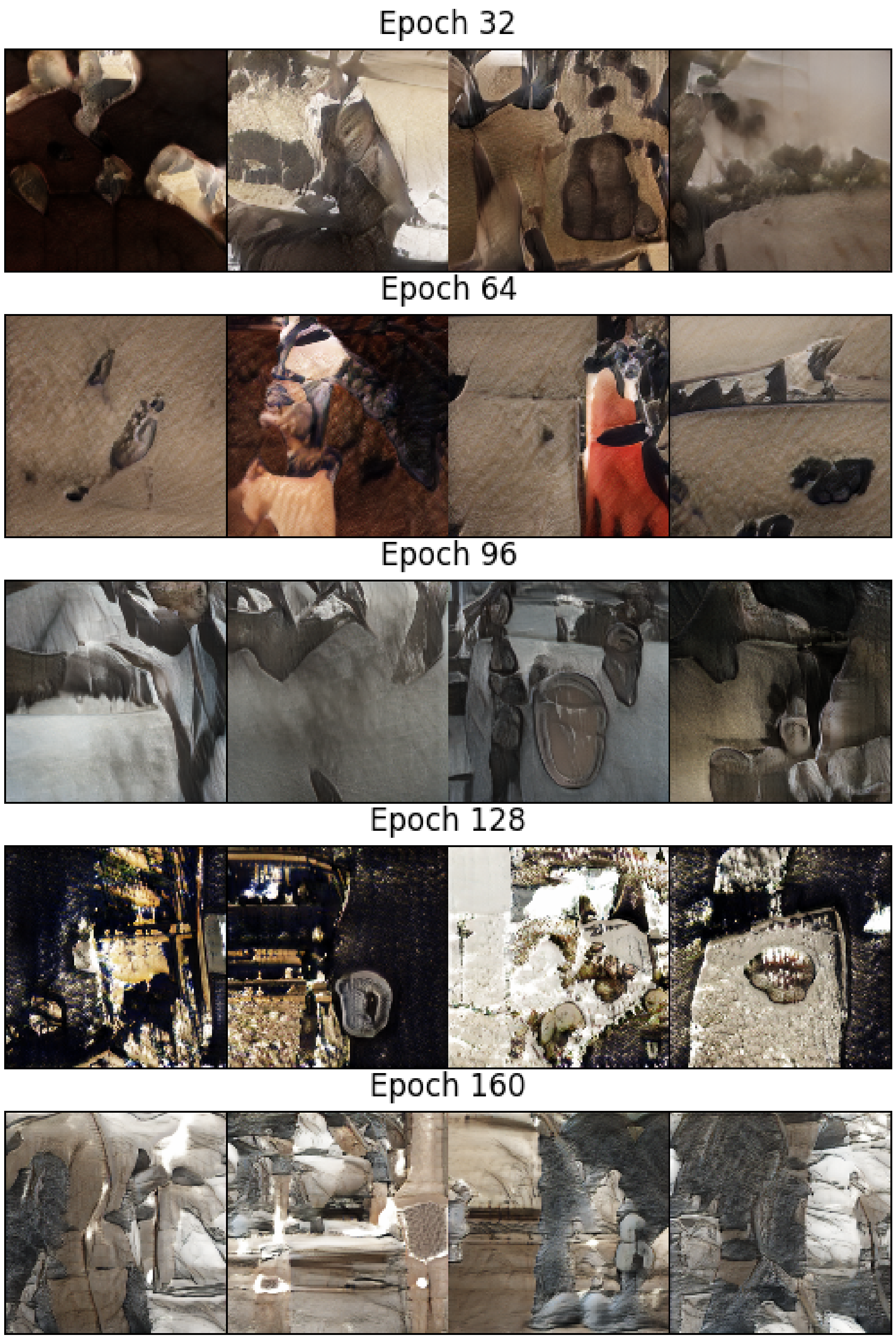


Figure B.28.: Original model generated images: “We are about five minutes late, I’m afraid.”

C. Configuration Settings

This Appendix provides the full set of configuration files applied in the project of this Thesis. The first configuration file serves as a basis, where the other configuration files only change the settings that must differ from the base configuration file for the specific task. Moreover, each subsection provides the name of the current file in the implementation of the project.

C.1. Main Configuration File

`config.py`: The following configuration file is the main configuration file, and provides a base for the subsequent configuration files.

```
from __future__ import division
from __future__ import print_function

import os.path as osp
import numpy as np
from easydict import EasyDict as edict

__C = edict()
cfg = __C

# Dataset name: flowers, birds
__C.DATASET_NAME = 'coco'
__C.CONFIG_NAME = ''
__C.DATA_DIR = ''
__C.DATA_SIZE = 'big'
__C.CUDA = False
__C.DEVICE = 'cpu'
__C.WORKERS = 8
__C.OUTPUT_PATH = ''
__C.RNN_TYPE = 'LSTM'   # 'GRU'
__C.B_VALIDATION = False

__C.TREE = edict()
__C.TREE.BRANCH_NUM = 3
__C.TREE.BASE_SIZE = 64

# Training options
__C.TRAIN = edict()
__C.TRAIN.BATCH_SIZE = 64
__C.TRAIN.MAX_EPOCH = 600
__C.TRAIN.SNAPSHOT_INTERVAL = 5
__C.TRAIN.DISCRIMINATOR_LR = 2e-4
__C.TRAIN.GENERATOR_LR = 2e-4
__C.TRAIN.ENCODER_LR = 2e-4
__C.TRAIN.RNN_GRAD_CLIP = 0.25
__C.TRAIN.FLAG = True
__C.TRAIN.NET_E = ''
__C.TRAIN.NET_G = ''
__C.TRAIN.B_NET_D = True

__C.TRAIN.SMOOTH = edict()
__C.TRAIN.SMOOTH.GAMMA1 = 4.0
__C.TRAIN.SMOOTH.GAMMA2 = 5.0
__C.TRAIN.SMOOTH.GAMMA3 = 10.0
__C.TRAIN.SMOOTH.LAMBDA = 0.0
__C.TRAIN.SMOOTH.LAMBDA1 = 1.0

# Caption_model_settings added by tingting
__C.CAP = edict()
__C.CAP.EMBED_SIZE = 256
__C.CAP.HIDDEN_SIZE = 512
__C.CAP.NUM_LAYERS = 1
__C.CAP.LEARNING_RATE = 0.001
__C.CAP.CAPTION_CNN_PATH = ''
__C.CAP.CAPTION_RNN_PATH = ''
__C.CAP.USE_ORIGINAL = False


# Modal options
__C.GAN = edict()
__C.GAN.DF_DIM = 64
__C.GAN.GF_DIM = 32
__C.GAN.Z_DIM = 100
__C.GAN.CONDITION_DIM = 100
__C.GAN.R_NUM = 2
__C.GAN.B_ATTENTION = True
__C.GAN.B_DCGAN = False


__C.TEXT = edict()
__C.TEXT.CAPTIONS_PER_IMAGE = 5
__C.TEXT.EMBEDDING_DIM = 256
__C.TEXT.WORDS_NUM = 18
```

```
# Vocab indices
__C.VOCAB = edict()
__C.VOCAB.PAD = 0
__C.VOCAB.THRESHOLD = 4

def _merge_a_into_b(a, b):
    """Merge config dictionary a into config dictionary b,
    clobbering the options in b whenever they are also specified in a.
    """
    if type(a) is not edict:
        return

    for k, v in a.items():
        # a must specify keys that are in b
        if k not in b:
            raise KeyError('{} is not a valid config key'.format(k))

        # the types must match, too
        old_type = type(b[k])
        if old_type is not type(v):
            if isinstance(b[k], np.ndarray):
                v = np.array(v, dtype=b[k].dtype)
            else:
                raise ValueError(('Type mismatch ({} vs. {}) '
                                  'for config key: {}').format(
                                  type(b[k]), type(v), k))

        # recursively merge dicts
        if type(v) is edict:
            try:
                _merge_a_into_b(a[k], b[k])
            except:
                print('Error under config key: {}'.format(k))
                raise
        else:
            b[k] = v


def cfg_from_file(filename):
    """Load a config file and merge it into the default options."""
    import yaml
    with open(filename, 'r') as f:
        yaml_cfg = edict(yaml.load(f, Loader=yaml.FullLoader))
    _merge_a_into_b(yaml_cfg, __C)
```

C.2. STEM Pre-Training

`pretrain_STEM.yml`: This configuration file serves to pre-train the STEM module.

```
DATASET_NAME: 'coco'

CONFIG_NAME: 'STEM'
DATA_DIR: '../data/'
DATA_SIZE: 'big'
CUDA: True
OUTPUT_PATH: '../output/'

TREE:
    BRANCH_NUM: 1
    BASE_SIZE: 299

TRAIN:
    BATCH_SIZE: 48
    FLAG: True
    NET_E: '' # No pretrained model
    ENCODER_LR: 0.002  # 0.0002best; 0.002good

TEXT:
    WORDS_NUM: 15
```

C.3. STREAM Pre-Training: New Model

`pretrain_STREAM.yml`: This configuration file serves to pre-train the new version of the STREAM module.

```
DATASET_NAME: 'coco'

CONFIG_NAME: 'STREAM_new'
DATA_DIR: '../data/'
DATA_SIZE: 'big'
CUDA: True
OUTPUT_PATH: '../output/'

TRAIN:
  BATCH_SIZE: 32
  MAX_EPOCH: 4
  RNN_GRAD_CLIP: 5.
  FLAG: TRUE

CAP:
  EMBED_SIZE: 256
  HIDDEN_SIZE: 512
  NUM_LAYERS: 1
  LEARNING_RATE: 0.0004
  CAPTION_CNN_PATH: '' # No pretrained model
  CAPTION_RNN_PATH: '' # No pretrained model
  USE_ORIGINAL: False

TEXT:
  WORDS_NUM: 15
```

C.4. STREAM Pre-Training: Original Model

`pretrain_STREAM_original.yml`: This configuration file serves to pre-train the original version of the STREAM module.

```
DATASET_NAME: 'coco'

CONFIG_NAME: 'STREAM_original'
DATA_DIR: '../data/'
DATA_SIZE: 'big'
CUDA: True
WORKERS: 8
OUTPUT_PATH: '../output/'

TRAIN:
  MAX_EPOCH: 4
  BATCH_SIZE: 32
  RNN_GRAD_CLIP: 5.
  FLAG: TRUE

CAP:
  EMBED_SIZE: 256
  HIDDEN_SIZE: 512
  NUM_LAYERS: 1
  LEARNING_RATE: 0.001
  CAPTION_CNN_PATH: '' # No pretrained model
  CAPTION_RNN_PATH: '' # No pretrained model
  USE_ORIGINAL: True

TEXT:
  CAPTIONS_PER_IMAGE: 5
  EMBEDDING_DIM: 256
  WORDS_NUM: 15
```

C.5. MirrorGAN Training: New Model

`train_coco.yml`: This configuration file serves to train the new version of the MirrorGAN model, specifically specifying the use of the new version of the pre-trained STREAM module.

```
CONFIG_NAME: 'MirrorGAN_new'
DATASET_NAME: 'coco'
DATA_DIR: '../data/'
CUDA: True
OUTPUT_PATH: '../output/'

TREE:
    BRANCH_NUM: 3

TRAIN:
    BATCH_SIZE: 64  # 22
    MAX_EPOCH: 650
    SNAPSHOT_INTERVAL: 1
    DISCRIMINATOR_LR: 0.0002
    GENERATOR_LR: 0.0002
    FLAG: True
    NET_E: '../models/big/STEM/text_encoder115.pth'
    NET_G: '' # No pre-trained model

CAP:
    EMBED_SIZE: 256
    HIDDEN_SIZE: 512
    NUM_LAYERS: 1
    LEARNING_RATE: 0.001
    CAPTION_CNN_PATH: '../models/big/STREAM/new/encoder_epoch4'
    CAPTION_RNN_PATH: '../models/big/STREAM/new/decoder_epoch4'
    USE_ORIGINAL: False
```

C.6. MirrorGAN Training: Original Model

`train_coco_original.yml`: This configuration file serves to train the original version of the MirrorGAN model, specifically specifying the use of the original version of the pre-trained STREAM module.

```
CONFIG_NAME: 'MirrorGAN_original'
DATASET_NAME: 'coco'
DATA_DIR: '../data/'
CUDA: True
OUTPUT_PATH: '../output/'

TREE:
    BRANCH_NUM: 3

TRAIN:
    BATCH_SIZE: 64  # 22
    MAX_EPOCH: 650
    SNAPSHOT_INTERVAL: 1
    DISCRIMINATOR_LR: 0.0002
    GENERATOR_LR: 0.0002
    FLAG: True
    NET_E: '../models/big/STEM/text_encoder115.pth'
    NET_G: '' # No pre-trained model

CAP:
    EMBED_SIZE: 256
    HIDDEN_SIZE: 512
    NUM_LAYERS: 1
    LEARNING_RATE: 0.001
    CAPTION_CNN_PATH: '../models/big/STREAM/original/encoder_epoch4'
    CAPTION_RNN_PATH: '../models/big/STREAM/original/decoder_epoch4'
    USE_ORIGINAL: True
```

C.7. Calculate BLEU: New Model

`validate_STREAM.yml`: This configuration file serves to calculate the BLEU scores of the new STREAM.

```
DATASET_NAME: 'coco'

CONFIG_NAME: 'STREAM_new'
DATA_DIR: '../data/'
DATA_SIZE: 'big'
```

```
OUTPUT_PATH: '../output/'
CUDA: True

TRAIN:
  FLAG: False
  BATCH_SIZE: 32
  RNN_GRAD_CLIP: 5.

CAP:
  EMBED_SIZE: 256
  HIDDEN_SIZE: 512
  NUM_LAYERS: 1
  CAPTION_CNN_PATH: '../models/big/STREAM/new/encoder_epoch4'
  CAPTION_RNN_PATH: '../models/big/STREAM/new/decoder_epoch4'
  USE_ORIGINAL: False

TEXT:
  WORDS_NUM: 15
```

C.8. Calculate BLEU: Original Model

`validate_STREAM_original.yml`: This configuration file serves to calculate the BLEU scores of the original STREAM.

```
DATASET_NAME: 'coco'

CONFIG_NAME: 'STREAM_new'
DATA_DIR: '../data/'
DATA_SIZE: 'big'
OUTPUT_PATH: '../output/'
CUDA: True

TRAIN:
  FLAG: False
  MAX_EPOCH: 4
  BATCH_SIZE: 32
  RNN_GRAD_CLIP: 5.

CAP:
  EMBED_SIZE: 256
  HIDDEN_SIZE: 512
  NUM_LAYERS: 1
  LEARNING_RATE: 0.0004
  CAPTION_CNN_PATH: '../models/big/STREAM/original/encoder_epoch4'
  CAPTION_RNN_PATH: '../models/big/STREAM/original/decoder_epoch4'
  USE_ORIGINAL: True

TEXT:
  WORDS_NUM: 15
```

C.9. Calculate FID: New Model

`fid_new.yml`: This configuration file serves to calculate the FID scores of the new MirrorGAN model.

```
CONFIG_NAME: 'FID_new'
DATASET_NAME: 'coco'

DATA_DIR: '../data/'
DATA_SIZE: 'small'
CUDA: True
OUTPUT_PATH: '../output/experiments/'

TRAIN:
  NET_E: '../models/big/STEM/text_encoder115.pth'
  NET_G: '../models/big/MirrorGAN/new/netG_epoch_160.pth'
  BATCH_SIZE: 32
```

C.10. Calculate FID: Original Model

`fid_original.yml`: This configuration file serves to calculate the FID scores of the original MirrorGAN model.

```
CONFIG_NAME: 'FID_original'
DATASET_NAME: 'coco'

DATA_DIR: '../data/'
DATA_SIZE: 'small'
CUDA: True
OUTPUT_PATH: '../output/experiments/'

TRAIN:
  FLAG: False
  NET_E: '../models/big/STEM/text_encoder115.pth'
  NET_G: '../models/big/MirrorGAN/original/netG_epoch_160.pth'
  BATCH_SIZE: 32
```

C.11. Generate Images: New Model

`gen_imgs_new.yml`: This configuration file serves to generate images from the specified sentences through the new MirrorGAN model.

```
CONFIG_NAME: 'new'
DATASET_NAME: 'coco'

DATA_DIR: '../../data/'
DATA_SIZE: 'big'
CUDA: True
OUTPUT_PATH: '../../output/experiments/'

TRAIN:
  NET_E: '../../models/big/STEM/text_encoder115.pth'
  NET_G: '../../models/big/MirrorGAN/new'
  BATCH_SIZE: 32
```

C.12. Generate Images: Original Model

`gen_imgs_original.yml`: This configuration file serves to generate images from the specified sentences through the original MirrorGAN model.

```
CONFIG_NAME: 'original'
DATASET_NAME: 'coco'

DATA_DIR: '../../data/'
DATA_SIZE: 'big'
CUDA: True
OUTPUT_PATH: '../../output/experiments/'

TRAIN:
  NET_E: '../../models/big/STEM/text_encoder115.pth'
  NET_G: '../../models/big/MirrorGAN/original'
  BATCH_SIZE: 32
```