Kevin Andre Helgeland

Classification of characters using modular techniques

Bacheloroppgave i Computer engineering

Veileder: Ole Christian Eidheim

Mai 2021

Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Kevin Andre Helgeland

Classification of characters using modular techniques

Bacheloroppgave i Computer engineering
Veileder: Ole Christian Eidheim
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Classification of characters using modular techniques

Kevin Andre Helgeland

Preface

This project was chosen to have a closer look into how we could better create an Optical Character Recognition for character sets that have a large variety of different characters. To do this, we used Japanese, since they use both chinese characters called kanji, hiragana and katakana. This made Japanese a good fit for our project.

Whereas we first intended to make a fully functional Optical Character Recognition for japanese, we soon realized that exploring a modular approach for character recognition would be interesting enough to explore without looking too much into Natural Language Processing and object detection.

This project was originally started as a part of a project this Autumn where we ended up with similar results. Because of this, the pre-processing used in the previous project is still used in this project. All models made and described during this project have been made for this project, as we redesigned the models for better results because we now have access to the Idun HPC server hosted by NTNU.

Finally, we would like to thank Ole Christian Eidheim for being an excellent supervisor for this project that has provided invaluable guidance for this project with quick responses. Thanks to the experience and fast, concise answers provided by Ole, this project would not have been possible without Oles support and guidance. We would also like to thank NTNU for allowing access to the Idun High Performance Computing group for their GPU resources. We would not have been able to initiate the experiments we did without this.

Signature: 27.05.2021, Trondheim, Kevin Andre Helgeland

Assignment text

This project shall explore the possibilities for character recognition using a modular architecture for a divide and conquer strategy for recognising japanese kanji and hiragana. We will create models from just standard linear architecture using CNN and also use architectures that are modular in nature like Mixture of Experts (Jacobs *et al*., 1991) and Negative Correlation Learning (Liu & Yao, 1999) to look for a better result.

Summary

During this project, we decided to have a look at modular architecture for training a model for classifying kanji and hiragana that are being used in japanese writing. The japanese generally use multiple types of character sets simultaneously, where they use hiragana, katakana and kanji for different uses. Katakana and hiragana use about 71 different characters each that represent sounds that often contain a consonant followed by a vocal. Some examples are の (hiragana) and ノ (katakana) that represents the sounds “no”.
On the other hand, a kanji represents ideas rather than sounds. Because they represent ideas or things, there has to be a lot of characters that have to be used. JIS recommends 6000 characters for everyday use in Japan, and has separated them into level 1 and level 2. We will be training for JIS level 1 to get most of the important characters.
When looking at CNN with a linear architecture, we see that others have managed to get up to 99.5% accuracy with 878 characters (Tsai,2016). As we will come to know, when we tested the same model on 3036 characters, we only reached 40% accuracy. Because we only reached a marginal accuracy, we decided to try a divide and conquer tactic to use multiple expert models that are specialized in their own problem space.
The first method we tried is Mixture of Experts (Jacobs *et al*, 1991) where we train 12 experts, where we use a gating model that distributes importance for each expert for each instance. This will create models that are specialized on separate parts of the problem space. The second method that we try is using Negative Correlation Learning (Liu & Yao, 1999). This method creates unique experts by calculating the correlation between the activation of the models, and rewarding negative correlations.

Table of contents

Chapter 1: Introduction 3

Chapter 2: Earlier work 4

2.1 Batch Normalization 4

2.2 Adams optimizer 5

2.3 ReLU family 5

2.4 Similar solutions 5

2.5 Mixture of Experts 6

2.6 Negative Correlation Learning 8

Chapter 3: Methods 8

3.1 Dataset 8

3.2 Pre-processing 10

3.3 Vanilla Concurrent Convolutional Network (CNN) 12

3.4 Mixture of Experts 14

3.5 Negative Correlation Learning 16

Chapter 4: Results 16

4.1 Accuracy 16

4.2 Top N accuracy 19

Chapter 5: Discussion 20

5.1 Solution 20

Chapter 6: References 21

Chapter 1: Introduction

Machine learning for detecting handwritten characters has come far, and can with very high accuracy detect characters. The MNIST dataset is a common dataset to train models for recognizing handwritten numbers for lab settings (Deng, 2012). In this paper, we shall have a look at classifying characters when there are thousands of classes to be classified. This is a substantially more difficult challenge, because the model will now have to account for more details than earlier. The dataset that is being used is provided by the ETL-9 Character Database, and contains all level 1 kanji and all hiragana. This all adds up to 3036 classes in the end. Where machine learning can accurately predict characters with low class count, a single model will eventually become too big to cheaply train with a monolithic model.

To solve this problem we have decided that we should test the feasibility of using a modular approach for training our model. When the problem becomes more complex, the size of the models increases along with it. This means that when a problem becomes big enough it becomes infeasible to use one monolithic model to solve everything. Because of that we are looking at a divide and conquer strategy to solve problems for big classed classification.

This report will use techniques such as Mixture of Experts (Jacobs *et al*, 1991) and Negative Correlation Learning (Liu & Yao, 1999) to create an Optical Character Recognition for all 3036 characters in the ETL-9 Character Database, and see if a modular approach can improve the results versus only using a linear architecture.

Chapter 2: Earlier work

During this chapter, we will give a rough explanation of all techniques used during this project. We will describe techniques used to standardise activation between layers like PReLU (He *et al*., 2015), ELU (Clevert, Unterthiner and Hochreiter, 2015) and Batch Normalization (Ioffe and Szegedy, 2015). In addition, we will show how well solutions in the past have done in the same problem space. Finally, we will describe how the Mixture of Experts (Jacobs *et al*, 1991) and Negative Correlation Learning (Liu & Yao, 1999) architectures are built.

2.1 Batch Normalization

Batch Normalization is a technique cited by Sergey Ioffe, Christian Szergedy (Ioffe and Szegedy, 2015) often used in machine learning to stabilize the neural network and to make the model faster as a result. Batch Normalization does this by reducing the internal covariance shift by using mini batches to normalize the activation vectors of the layers Batch Normalization is used in. Batch Normalization normalizes the data by calculating the variance and mean value of the minibatch and scaling the output by adding the variance and mean value.
When using batch normalization it is sensible to use higher learning rates in your model, since this method naturally prevents exploding or vanishing gradients by scaling the values in layers either up or down. Because of the inherent nature of batch normalization regulating any large or small values into a normalized output.

2.2 Adams optimizer

Adam optimizer is the optimizer proposed by Diedrik P. Kingma and Jimmy Lei Ba (Kingma. and Ba, 2014) that uses stochastic gradient optimization that uses adaptive learning rates during training. Adam optimizer combines the Adagrad (Duchi *et al*, 2011) and RMSProp (Tielman & Hinton, 2012) in an attempt to get the best of both worlds, where Adam optimizer adds a bias correction that prevents large initial steps.

2.3 ReLU family

During this project there will also be mentions of PReLU (He *et al.*, 2015) and ELU (Clevert, Unterthiner and Hochreiter, 2015). These are both variations of ReLU. Whereas ReLU will output $y = x$if $x > 0$ and $y = 0$if $x <= 0$to create a non linear activation, it has a problem of dying ReLU and exploding gradient. Dying ReLU happens when too many activations are set to 0, and therefore prevents a large portion of the neurons from ever activating. This has been solved by using PReLU, that uses $y = b * x$ if $x <= 0$where b is a very low number (He *et al*,

2015). This prevents the layers from setting too many neurons to 0, which means that the model no longer kills portions of the neurons.
Furthermore, ELU was also found to converge the model with fewer epochs if it used $y = b * (a^x - 1)$ if $x <= 0$because it saturates negative values. This allows us to get the benefits from sigmoid activation without the vanishing gradients.

2.4 Similar solutions

There have been multiple previous solutions when solving our problem space. Many of which we found used techniques to reduce the feature space before training. One of which used PCA to reduce the feature space (Hyaman *et al*., 1991), which managed to get an accuracy of 89.1% when using 160 principal components when classifying 956 characters.
Other methods for recognizing characters have also been proposed to recognize kanji by detecting strokes and radicals like radical level representation using CNN (Ke & Hagiwara, 2018). This model detects radicals inside a kanji, and uses LSTM to detect the kanji. Furthermore, it uses highway layers instead of dense layers to employ some natural language processing as well. A model like this requires many types of data, and would be expensive to train, even though effective with 93.5% accuracy.
Furthermore, Takuya Okada and Kazuhiro Takeuchi(Okada & Takeuchi, 2018) tried applying a sparse autoencoder to automatically detect strokes in a character. This would be a cheaper option to training the model to recognise radicals. They chose to use a dataset that consists of 3036 characters to train their model. The sparse autoencoder managed to reach an accuracy of 85%.
In this report, we will be using a Convolutional Neural Network (CNN) as a gold standard for classifying Japanese characters. According to Charlie Tsai (Tsai, 2016), he could make CNN recognize up to 878 classes for kanji with a 99.5% accuracy by using a CNN directly. He did this by basing his model on VGGNet (Simonyan and Zisserman, 2014), and is currently one of the better architectures for image recognition out there at the moment. Comparing this model in our setting, we can see that the model only reaches an accuracy of 38% when there are few changes to the model at hand. The reason our method likely reaches a much lower accuracy than Tsais model, is that we are working on a training set consisting of 3036 classes whereas Tsais is working with 878 classes despite being an otherwise similar dataset.

2.5 Mixture of Experts

Robert A. Jacobs (Jacobs *et al*, 1991) proposed a new model architecture for allowing for a more modular approach when classifying instances. This approach uses two separate types of training blocks to train the model. The first block is called the gating model. This is trained to make a coarse decision in the early layers of the model, and thus adjust the error on the experts so that only the experts that are predicted to be relevant for the instance will be trained.

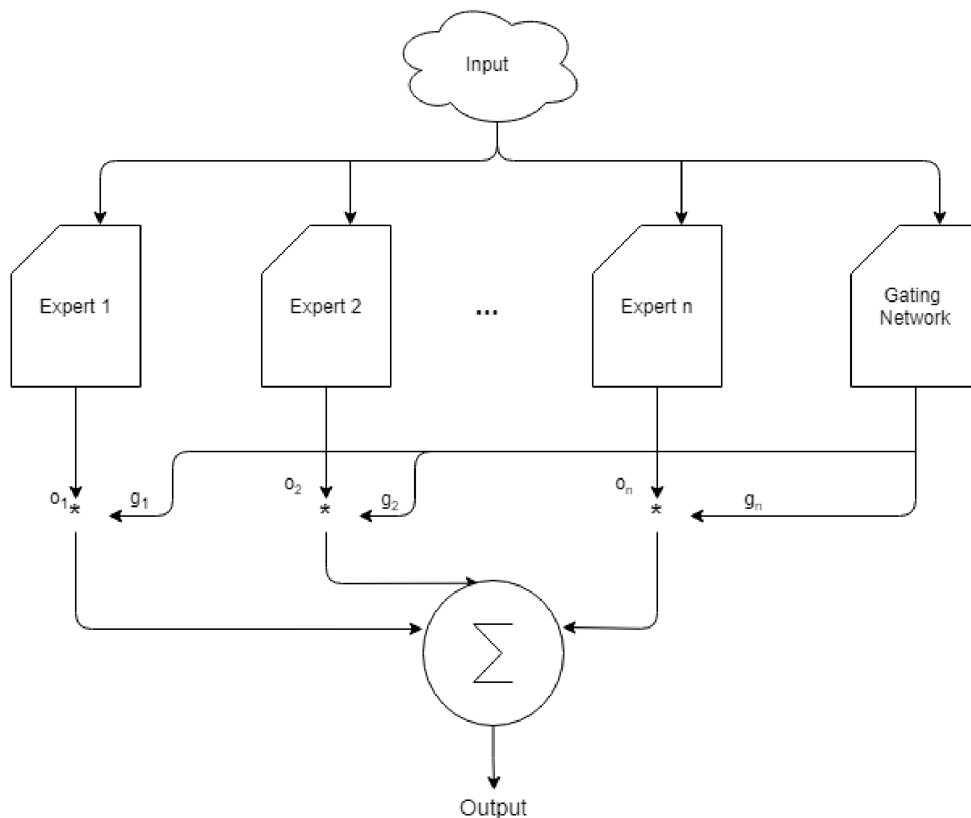


Figure 1: This figure represents the architecture of Mixture of Experts (Jacobs et al, 1991). We can see that the input is being used equally in all sub-networks in the model. The gating network thus decides the significance of each expert by itself, returning a significance value between 0 and 1.

The second block is the expert block. In this block, the model is making the finer choices for the model. When the error is calculated, we are using the formula below to train the experts into localized models. This is great since localized models make each expert model as modular as possible as opposed to cooperative experts that need to work together to average to the correct answer.

$$E = - \log \sum_i g_i e^{-\frac{1}{2}||y - o_i||^2}$$

Formula 1: This is the error function for MoE where i indicates a single expert, y is the true value, o is the output from an expert and g is the gating value for that expert in this instance.

2.6 Negative Correlation Learning

Another method for solving a machine learning problem with modularity is Negative Correlation Learning (NCL) (Liu & Yao,1999). The idea is similar to MoE in that we are creating expert networks to solve a subset of the problem, but unlike MoE, this method does not require a gating block to regulate the experts. NCL will instead impose error penalties based on the correlation to other models. This ensures that experts are as negatively correlated as possible, and each model is trained on different problems.

$$E_i = \frac{1}{N} \sum_{n=1}^{N} E_i(n) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2} (F_i(n) - d(n))^2 + \frac{1}{N} \sum_{n=1}^{N} \lambda p_i(n)$$

Formula 2: This is the error function used by NCL to train a single expert. N is the total number of experts and "i" determines a single expert. λ is a number between 0 and 1 that signifies how strongly the correlation should play an effect. "p" is the correlation function.This allows us to adjust how much the correlation should play a role.

$$p_i(n) = (F_i(n) - F(n)) \sum_{j \neq i} (F_j(n) - F(n))$$

Formula 3:This is the correlation function that is being used to determine the correlation between an instance and the current expert. This will be used along with λ, to reward negative correlated instances and punish strongly correlated instances.

Using the two functions above, we can successfully determine the correlation between each expert so that we can stimulate the experts to train on different parts of the problem space.

Chapter 3: Methods

This chapter will first discuss the dataset we are working with, how we have decided to pre process the data for use in the models and how the models were built. We will also describe each

of our experiments that consists of three vanilla CNN, one Mixture of Experts model and one Negative Correlation Learning model.

3.1 Dataset

This project is going to classify handwritten Japanese kanji for use in OCR. Japanese writing generally consists of 3 character systems; hiragana, katakana and kanji. Hiragana and katakana both use a similar system, where the characters usually represent a consonant and a vowel with exceptions. For instance な and ナ both represent the sounds *na*, where the first character is a hiragana character, and the latter is katakana. The style for each of the character systems are slightly different, where katakana are written with more straight lines and clear edges, and hiragana are more curvy.
There are 46 different characters in hiragana and katakana respectively, with 25 characters that are variations of some of the characters that are called *dakuon* and *handakuon*, making hiragana a total of 71 classes to classify. An example on the variation is は (ha), ば (ba,dakuon) and ぱ (pa,handakuon). These can be exceptionally difficult to classify without a logic that looks for the dots or circle on the edge. However, in this project we will choose to ignore this, since they represent a smaller portion of the dataset, and we will instead løook at the dataset as a whole.
Kanji on the other hand is an entire system entirely. Kanji is an inherited system from China that consists of thousands of characters that represent entire words or ideas of words. One character consists of multiple radicals that are repeating patterns. For example, can we look at the character 字 that means *character* and the kanji is built off of of the radicals 宀 (roof) and 子 (child). This allows humans to more simply learn mnemonics like “A child living under a roof can read characters”. We hope that these recurring patterns coming from radicals can also be used by neural networks on a lower level despite not detecting individual radicals before classifying the characters, since teaching the network about radicals requires much more granularity in the data.
The dataset we are using for this project is provided by the Electrotechnical Laboratory, Japanese Technical Committee for Optical Character Recognition, and we will be using the ETL-9 Character Database. This contains handwritten kanji and hiragana, the dataset is using the JIS X 0208 for labeling the characters along with a serial number for easier use in indexing. The kanji

is limited to the characters in JIS X 0208 Level 1 class,meaning the dataset consists of 2965 kanji and 71 hiragana with a total of 3036 classes.
The data comes in a 127x128 pixel format in a greyscale where each pixel is a number between 0 and 255 to represent the whiteness.

Character	Classes	Writers	Records	Resolution
Kanji	2965	200	593000	127x128
Hiragana	71	200	14200	127x128
Total	3036	200	607200	127x128

Table 1: This is a description of the dataset. The dataset were made by having 200 separate writers write each character once. We get 71 different hiragana and 2965 different kanji with a total of 3036 characters. This gives us a total of 200 instances of each character written by different people.

3.2 Pre-processing

For this project, we decided to pre-process each of the images so that the images are all the same size and in the center of the image as often as possible. Since we get a resolution of 127x128, we need to resize the resolution to at least 64x64. The reason we want to resize the images is because we want to save resources when training without losing much details. This preprocessing for kanji was developed for the autumn project for the autumn project during the TDAT3025 subject and reused for this project (Helgeland,2020).

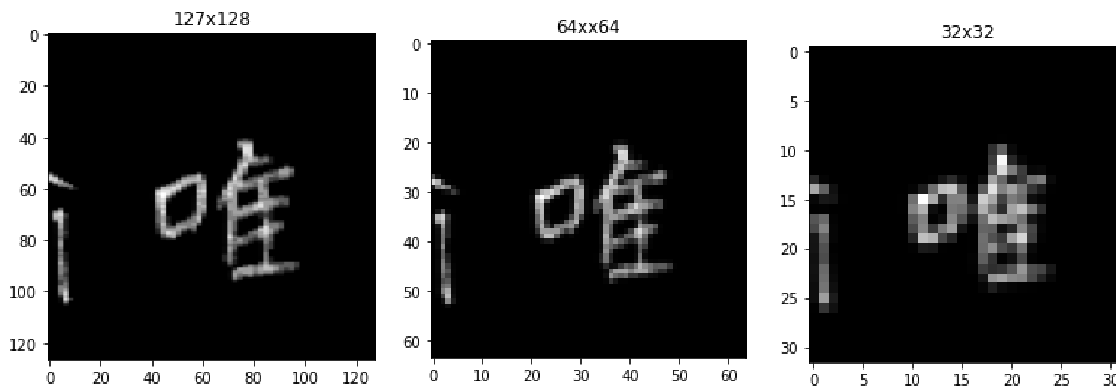


Figure 2: Here we see the transformation when resizing the images directly from 127x128, 64x64 and 32x32. We can see that 32x32 loses a lot of details on even a pretty clear image once the character turns out small. 64x64 seems to be doing fine so far.

As we see in figure 2, there is a lot of space on the image that remains unused. This could potentially affect the classification negatively, because the deeper layers will have less details to work with. The image will be clumped into the center and the dimensions are being reduced with maxpool. To solve this problem, we will first crop the image before we resize the image to reduce the empty space area.

Cropping is implemented by detecting whole rows and columns that are empty, and removing them. This cropping method works fine in most cases, but has a weakness with characters that use empty space as a part of the character. An example of a character that uses empty space is こ (ko, hiragana), where the character consists of two horizontal lines with space between. That character will become flattened, and therefore distorted. However, for our case flattening should not provide any issues, because all of the instances of characters with rows and columns of space inside will be distorted equally.

Since we are already cropping the image, we decided to also try to crop away noise from the images. We noticed that the noise almost alway came from edges followed by space. We then decided to crop any noise that starts at the edges until whole columns or rows that are empty. This method is dangerous with the risk of cropping away the entire image if there is a lot of noise. In the scenario of too much noise, we have decided to add a fail safe that cancels the noise reduction altogether if the function tries cropping more than 25% of the image from each of the sides. Stopping cropping might affect noisy instances negatively, but with that much noise the instance will likely be difficult to use anyway.

Further along noise reduction, we also treat all pixel values under 55 as 0. This action crops fait smudges with no known repercussions.

If we were to resize now, we would risk distorting the image by stretching it. There is no guarantee that the image has equal dimensions. This problem can easily be solved by padding the lower dimension sides to make each dimension equal. We choose to pad equally on each side, so that the character is centered in the middle of the image.

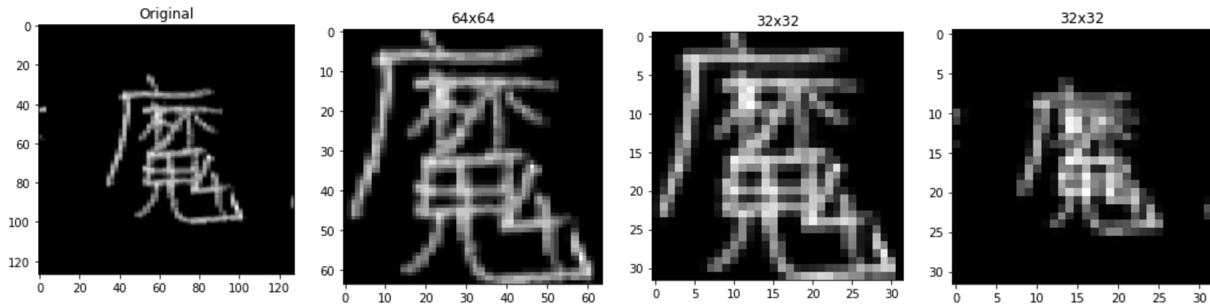


Figure 3: This figure shows before and after resize. To the left, we see the original image for reference. In the middle-left, we can see the cropped and resized image at 64x64. We can see that the image is mostly clear if not a little bit blurry. This happens because it crops to smaller than 64x64, and has to upscale the image. To the middle-right, we see an image that is 32x32, and can see that the details are still not lost. To the right, we can see the image resized to 32x32 before cropping for reference.

We can see that we get overall good results from cropping the image, and it is entirely possible to downscale to 32x32 when we use figure 3 as reference. However, in our experiments, we decided to use 64x64 instead. The main reason we decided this is because we wanted to use and compare our results to Charlie Tsais (Tsai, 2016) model that uses 64x64. In addition, our cropping method is known to have a fail safe where it will cancel it's cropping. If this were the case, we can see based on both figure 2 and figure 3 that uncropped 32x32 is not usable. In order to keep the few uncropped images usable, we will use the 64x64 resolution. This is further enabled by getting access to a High Performance Computing Group hosted by NTNU.

3.3 Vanilla Concurrent Convolutional Network (CNN)

For the purpose of comparing the modular methods to a linear approach, we used a model based on VGGNet pulled from one of Charlie Tsai's (Tsai, 2016) methods. This is because it's a tried and tested method for a problem space with 878 kanji. We first tried training the model as is, but we also tried increasing the dense depth to 8192 because we have a bigger output space than Charlie Tsai did to see how much we can gain by doubling the size.

	Left model
Convolutional layers	Conv 64
	Conv 64
	Maxpool 32x32
	Conv 128
	Conv 128
	Maxpool 16x16
	Conv 256
	Conv 256
	Conv 256
	Maxpool 8x8
	Conv 512
	Conv 512
	Conv 512
	Maxpool 4x4
	Conv 512
	Conv 512
	Conv 512
	Maxpool 2x2
Fully connected layers	Dense 4096
	Dense 4096
	Dense 3036
	softmax

	Right model
Convolutional layers	Conv 64
	Conv 64
	Maxpool 32x32
	Conv 128
	Conv 128
	Maxpool 16x16
	Conv 256
	Conv 256
	Conv 256
	Maxpool 8x8
	Conv 512
	Conv 512
	Conv 512
	Maxpool 4x4
	Conv 512
	Conv 512
	Conv 512
	Maxpool 2x2
Fully connected layers	Dense 8192
	Dense 8192
	Dense 3036
	softmax

Figure 4: Here we can see the way we have set up the models that we have called "Vanilla". The model to the left is the closest one to one of the models from Charlie, where The left one is an extended model.

The models are implemented in python using PyTorch. Each Convolutional layer uses kernel size of 3 and a padding of 1. This means that each layer outputs the same resolution as the input, and a kernel size of 3 is a very normal size. One could argue for kernel size 5 in the earlier layers to make the model faster, but we chose to stay at the granularity of size 3. On a higher level, after each Convolutional layer we use Batch Normalization followed by a PReLU layer. This prevents exploding gradients in our model, seeing as it is a rather deep model.

The dense layers are only followed by PReLU/ELU, as Batch Normalization seemed to slow down the model significantly. Another variation of the model to the right in figure 4 was where we used ELU instead of PReLU, since that has been shown to converge with fewer epochs, though a bit slower.

Each model was trained using a learning rate of 1e-4, because that seemed to be the highest learning rate that kept a stable loss. At the same time, we chose to train our model using a minibatch size of 64. Normally a minibatch size of 16 or 32 would be preferred, but because we wanted to speed up the training process, we chose to use 64 instead. The learning rate and minibatch size is consistent in all models.

3.4 Mixture of Experts

Seeing how our problem space seemed very big, and had been solved at a smaller problem space, we decided to try out modular methods to go for a divide and conquer technique by creating multiple smaller networks dedicated to solving a sub part of the problem space. Mixture of Experts (MoE) seemed to be a good fit for this problem, seeing as it trains multiple localized networks that are gated by a separate gating network (Jacobs *et al*,1991). Each expert needs to be as negatively correlated as possible to be as effective as possible (. We tried two general strategies for solving our problem, where the first uses a static pre-trained gating network and the other uses a dynamic gating network that is being trained along with the experts.
If we were going to use the static gating method, we would have to first decide what characters are most negatively correlated(Yuksel, Wilson, & Gader, 2012; Masoudnia & Ebrahimpour, 2014). This could be done by extracting features by using either a pre-trained CNN or by using an auto encoder to detect strokes of the characters, and then use a k-means to cluster the experts into negatively correlated classes. Once we have the classes ready, we could train the gating model based on the clusters, and freeze it when training the rest of the experts. This would require more manual work, but could also provide a model that is easier to debug, since we can directly have a look at what experts are struggling with, and what characters those are trained on. Despite this, we realized quickly that a model this rigid did not train well, because it is difficult to cluster the characters into negatively correlated clusters. There are also no cooperation between networks that happen to actually be somewhat correlated
On the other hand, the dynamic model seems much more reliable, as it can dynamically find the best experts to endorse based on the early layers in the model that it trains on. Since these layers usually tell the model about individual lines and strokes, it has a good leverage for deciding which experts are likely to output the lowest error. This allows the gating model to adjust to the expert models as they are training on their own. This technique does have the weakness of not

guaranteeing that the experts are training on different “expertises”, which means that experts could be trained to be correlated in the worst case scenario.

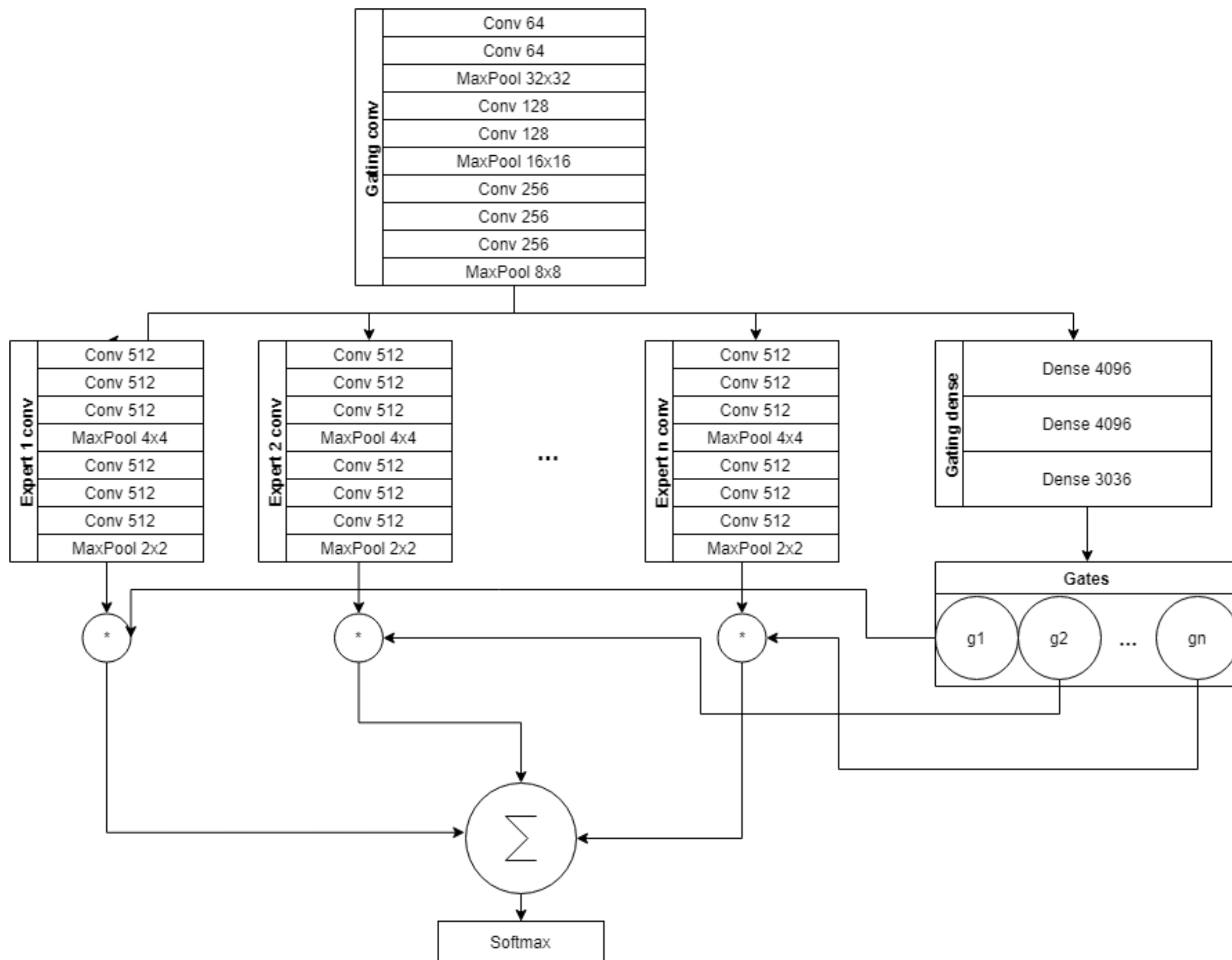


Figure 5: This is a model of the Mixture of Experts implementation that we used. We use the output from the gating convolutional layers as input for both the gating dense layers and the expert convolution layers. Because the early layers mostly detect simple shapes like lines, curves and edges, so we can reuse the output from those layers.

The gating values are normalized to a number between 0 and 1 with the formula 3 as described in the previous work chapter. Doing this will reduce expert strength for experts that are not well trained for a certain instance which allows the experts to compete to be used. Just like the vanilla CNN, we are using Batch Normalization and PReLU after each convolutional layer and PReLU after each dense layer.

3.5 Negative Correlation Learning

Since Negative Correlation Learning (Liu & Yao, 1999) is created to be able to create negatively correlated experts, we decided to implement this as well as a response to our Mixture of Experts not being able to guarantee a negative correlation to see if that helps our case. Each expert uses the same structure as the right model in figure 4 in vanilla CNN. In addition, we tried to separate a block of the earlier layers into a shared block that we call a stem. The experts branch out from the stem for the last layers. We do this because the earlier layers find the same simple features anyway, whereas the later layers are where the uniqueness happens anyway. Using a common stem could reduce computer resources that are being used.

When training Negative Correlation Learning and Mixture of Experts, we chose to use 12 experts. This seemed to be the limit of the computer resources of what we got. This would optimally give each expert the domain over 253 classes if there were no overlap.However, when using models that are competing, there is bound to be overlap. However, seeing as how previous work seems to handle character recognition fine when categorizing 878 characters, this should be sufficient.

Chapter 4: Results

This chapter will talk about the results we have managed to accumulate during these experiments. During this project, we have created three different CNN models to function as a reference to how well the modular models are working. As for the modular methods, we shall compare the results from Mixture of Experts (MoE) and Negative Correlation Learning (NCL).

4.1 Accuracy

The models are trained on NTNUs hpc server called idun using tesla P100 GPU. Each model was given 10 epochs to train where one epoch corresponds to using all training data. The training data is 80% of the full dataset. We choose to use 80% as training data because the dataset has inherently a wide problem space, and need as many varieties of writers as possible for a more stable model. The last 20% of the dataset will be used as a separate test dataset to verify the accuracy.

Model:	**Accuracy**	**Training time:**
Vanilla CNN FC:4096 w/PReLU	39.8%	44 hours
Vanilla CNN FC:8192 w/PReLU	39.8%	45 hours
Vanilla CNN FC:8192 w/ELU	38.5%	46 hours
Mixture of Experts w/stem	40.5%	61 hours
Negative Correlation Learning	40.7%	46 hours
Negative Correlation Learning w/stem	40.7%	60 hours

Table 2: This is a table showing the final results for each model. The training time shows how much time it took to train the model, including calculating the accuracy. When looking at training time, the models were trained on the NTNU hpc cluster using Tesla P100 GPU. The times should be read with a grain of salt, since my process could have been interrupted by another more prioritised process. However, it does give a ballpark estimate.

The idea of stem seems to be making the models that I use it in slower, since it is used in both Mixture of Experts and the second Negative Correlation Learning model. The reason for this added time despite less calculations could be that PyTorch works the stem synchronously. If this is the case, the fact that the cuda cores are somewhat prohibited from working fully asynchronously across the entire model. However, a stem will reduce resource usage in terms of memory usage.

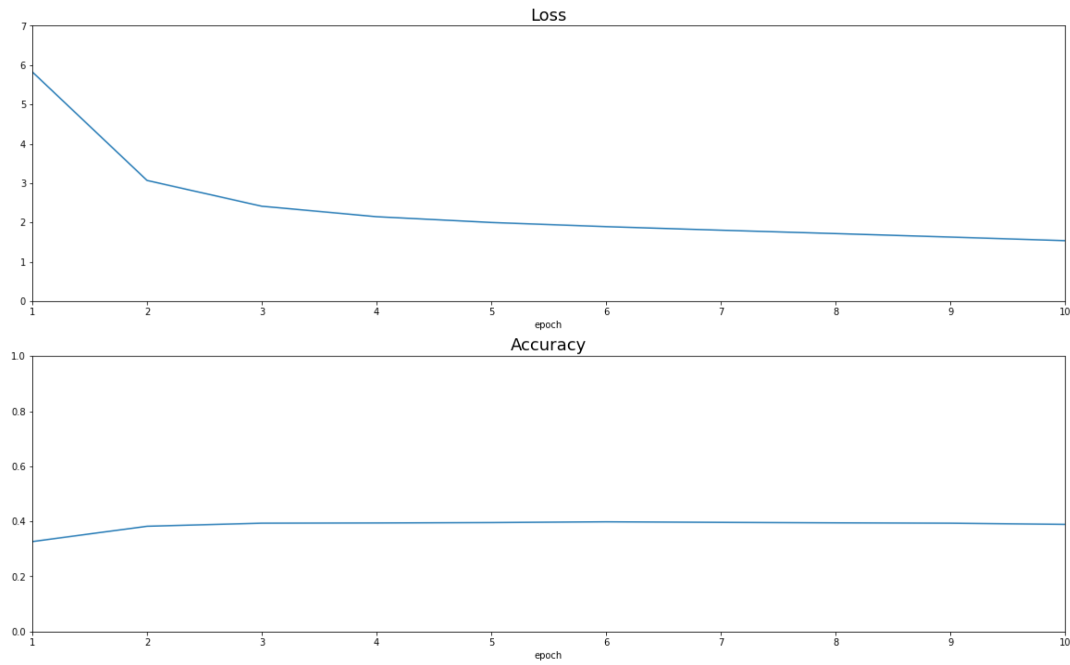


Figure 6: In this figure, we can see the accuracy growth during the training of the vanilla CNN model. The model seems to converge after one and two epochs and stay stable. All the vanilla models had the same trend regardless of model depth and activation function.Doing this again would prefer to measure accuracy mid epoch after for more granularity.

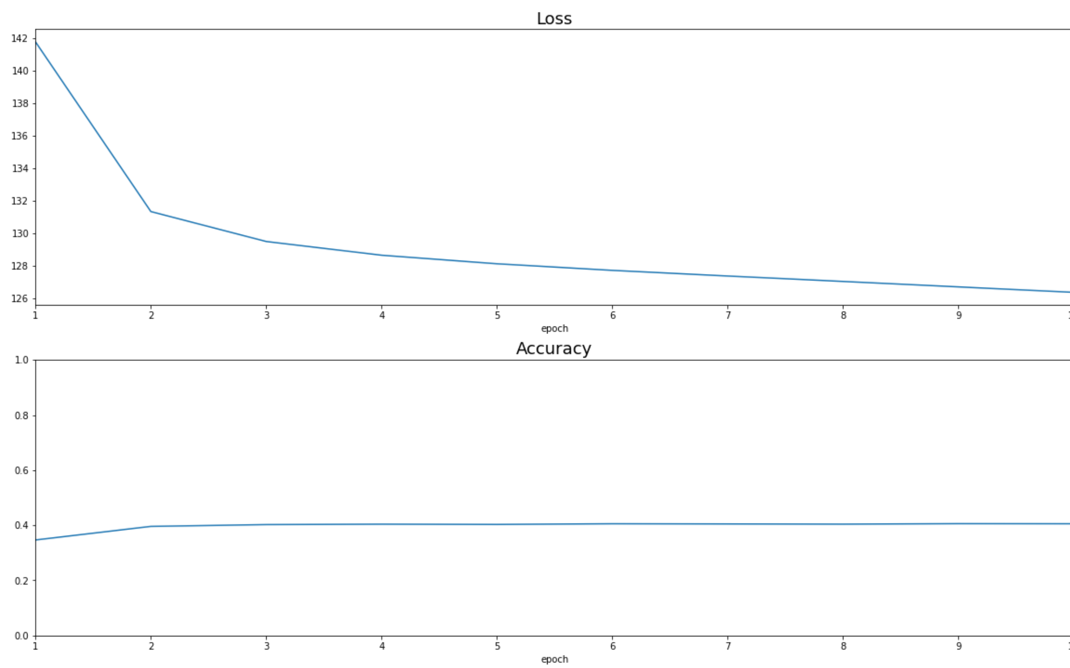


Figure 7: This is the model for displaying the loss and accuracy for the Mixture of Expert model for each epoch. As one can see, it outputs similarly to Vanilla CNN. This tells us that Mixture of Experts does not seem to have any other setbacks for working having multiple models working separately and also having to train a gating model.

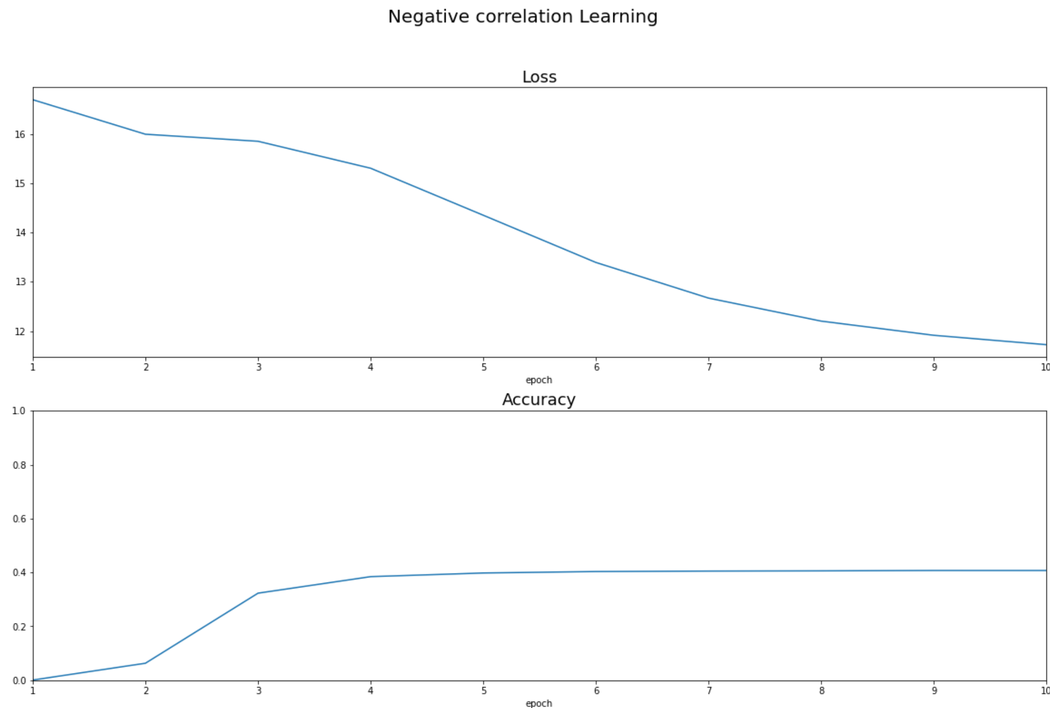


Figure 8: this is the model for displaying the loss and accuracy for the Negative Correlation Learning during training for each epochWe can see that the model trains slowly at first, bus speeds up after 2-3 epochs. This is likely because the experts are very correlated in the beginning. In this case, the models are encouraged to find their own problem space instead of increasing accuracy right away.

4.2 Top N accuracy

Because these models have a generally low accuracy as seen in table 2, we decided to have a look at top n accuracy. If we look at the accuracy when looking at the top n guesses, we would like to see how much the accuracy increases. This is to have a look at the possibility for Natural Learning Processing to correct the incorrect accuracies later when context is applied in the sentences.

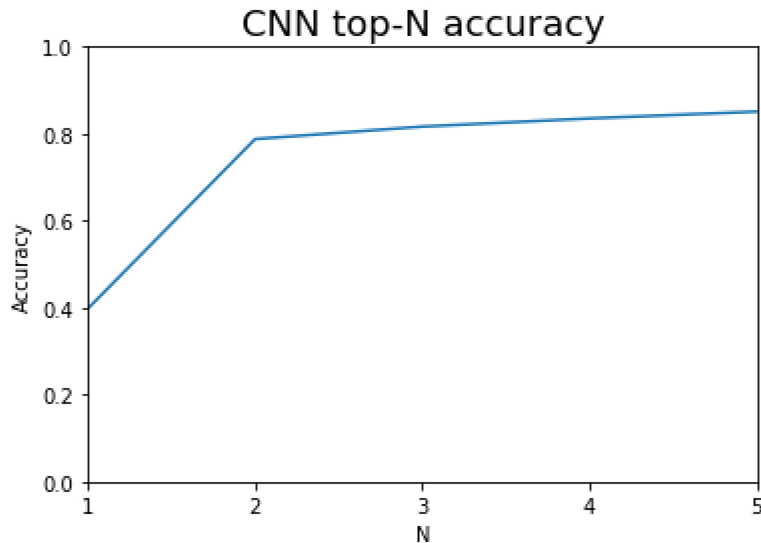


Figure 9: This figure displays how the accuracy increases as we increase N for checking top N accuracy of the model for Vanilla CNN. As we can see, if we accept the top 2 characters as an answer, the accuracy reaches 79% accuracy, and almost linearly increases past that point by 2% per N past that point.

Chapter 5: Discussion

This chapter will discuss the results that we got during this project. We will compare the results from previous work to what we have achieved and discuss the usages for the different architectures based on our findings. We will end this chapter by reflecting on what could be better and future work.

5.1 Solution

As we see in the results chapter, the results have been less than adequate for an OCR that should be somewhere between 80-90% if it were to be used. Instead we have an accuracy of 38-40% across all models. On the other hand, we can see from the figure 9 that if we introduce top N accuracy, the model then reaches an adequate accuracy. This tells us that techniques that employ context on top of the results like Natural Language Processing could employ our model and still get decent results.

Another observation we would like to make is that when classifying kanji with a large number of output classes a very common strategy for doing this is by using techniques that encode the

characters into smaller characters like radicals or strokes by some kind of autoencoder. Tanuya Okada and Kazuhiro Takeuchi (2018) managed to reach 80-90% accuracy with the same dataset with 3036 classes using autoencoder, whereas we managed to get to 40% accuracy with Mixture of Experts and Negative Correlation Learning. Our model does not detect smaller components, despite the fact that it would have likely benefited from an autoencoder. This strengthens the idea that using smarter feature extractions with a monolithic model seems to be a better way of approaching a big problem space rather than dividing the problem into multiple models when using the current techniques.

Whereas Mixture of Experts and Negative Correlation Learning might not be the solution we had hoped for, the fact that the results stayed mostly the same compared to vanilla CNN tells us that it might still be used for dividing the problem into multiple computational units. As we're solving complex problems, we might encounter problems that require huge models. In this case a modular architecture will more easily allow for a horizontal scaling instead of a vertical scaling, which tends to be cheaper.

As we made some tests with PReLU versus ELU, we found that PReLU worked best in our problem. This could be that when there is a big problem space, PReLU retains information about which classes are just bad, while others are very bad. On the other hand, ELU will not retain this information to a high degree because of its logarithmic properties. However, while there is an increase in accuracy, this could also be a case of bad luck for ELU. If the start bias for the ELU test were unlucky, the model could find a local minima that the other two vanilla tests did not.

Chapter 6: References

Tsai, C., 2016. *Recognizing handwritten Japanese characters using deep convolutional neural networks.* University of Stanford in Stanford, California, pp.405-410.

Yuksel, S. E, Wilson, J. N & Gader, P. D, 2012. Twenty Years of Mixture of Experts. *IEEE transaction on neural networks and learning systems*, 23(8), pp.1177–1193.

Masoudnia, Saeed & Ebrahimpour, Reza, 2014. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2), pp.275–293.

Liu, Y & Yao, X, 1999. Ensemble learning via negative correlation. *Neural networks*, 12(10), pp.1399–1404.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E., 1991. Adaptive mixtures of local experts. *Neural computation*, *3*(1), pp.79-87.

Ke, Yuanzhi & Hagiwara, Masafumi, 2018. CNN-encoded Radical-level Representation for Japanese Processing. *Transactions of the Japanese Society for Artificial Intelligence*, 33(4), pp.D-I23–1-8.

Okada, Takuya & Takeuchi, Kazuhiro, 2018. Comparing Sparse Autoencoders for Acquisition of More Robust Bases in Handwritten Characters. *Integrated Uncertainty in Knowledge Modelling and Decision Making*, 10758, pp.138–149.

He, Kaiming et al., 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015 *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp.1026–1034.

Clevert, D.A., Unterthiner, T. and Hochreiter, S., 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Hyman, S.D et al., 1991. Classification of Japanese Kanji using principal component analysis as a preprocessor to an artificial neural network. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, i, pp.233–238 vol.1.

Deng, L., 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6), pp.141–142.

Electrotechnical Laboratory, Japanese Technical Committee for Optical Character Recognition, *ETL Character Database*, 1973-1984

Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Duchi, John, Hazan, Elad & Singer, Yoram, 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12, pp.2121–2159.

Tieleman, T. and Hinton, G., 2012, Lecture 6.5 - RMSProp, *COURSERA: Neural Networks for Machine Learning. Technical report*, .

Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
Helgeland K. A., 2020, *Kevinah rapport_ Gjenkjenning av japansk håndskrift*, TDAT3025

Appendix:

Project architecture

```
|
|____Vanilla
|         |___main.py
|         |___main_topN.py
|         |___Vanilla_CNN_M4.py
|         |___Models\
|
|____MixtureOfExperts
|         |___train_MoE.py
|         |___main_topN.py
|         |___MixtureOfExperts.py
|         |___Models\
|
|____Vanilla
|         |___train_NCL.py
|         |___main_topN.py
|         |___Vanilla_CNN_M4.py
|         |___Models\
|
|____OCR_preprocessing.ipynb
```

Figure 10: this is the project structure used during this project.Because the models were trained separately, the training scripts were copied instead of creating scripts that take in parameters to unify the architecture.

Installations and running

Running the scripts require the following dependencies:

Python 3.8.6

PyTorch 1.7.1

CUDA 11.1.1*

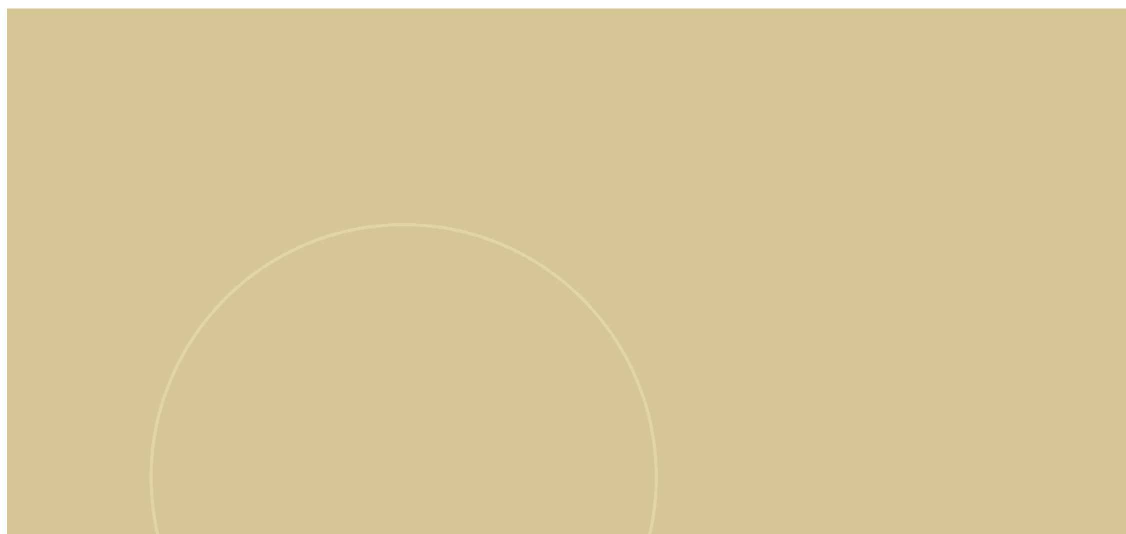
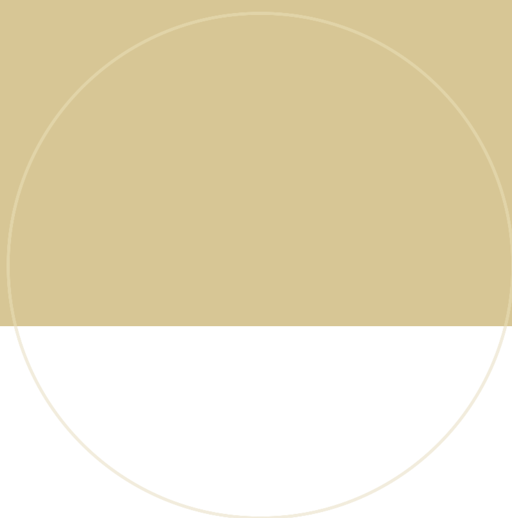
*Cuda also requires some Nvidia specific hardware to use. The model is trained on the Tesla P100 GPU, and assumes that a similar or better GPU is equipped.

When running the scripts, the working directory should be the same as the main.

Main.py, train_MoE.py and train_NCL.py are the scripts that should be run to train the models.

Remember to set the training and test data paths accordingly in the get_data functions.

OCR_preprocessing.ipynb requires a jupyter notebook to run and uses both matplotlib and numpy for visualization ona rescaling.

NTNU

Kunnskap for en bedre verden