DS807 exam

Jakob J. Rasmussen(Jakra16), Jakob H. Olesen(jaole16), Mathias Ø. Hansen(mathh17)

January 30, 2022

Data management fundamentals (Responsible: JJR)

This chapters code is available in Hand_in_NB.ipynb in section "Data management → 12k train, test, val split" & "30K test set, import".
When working with classification of text snippets on a piece of paper, the importing and preparation of data is adjusted to accommodate this data type in a manner that in general should ease computation. We chose to work with the DIDA_12000 data and will be splitting it into separate train, validation and test sets. Images are transformed from RGB to grayscale to decrease complexity and further normalized to ease calculations on inputs by normalizing them to be in value range 0 - 1. All images are resized to induce a uniform format of image resolutions across the board. On average height to width ratio of the images are inspected to be 1:1.9 and all images are reshaped to fit this proportion. The image size is 56 pixels by 106 pixels is applied as it is deemed reasonably large to represent four or more digits within the image, Figure 1. The dimension roughly equates to an image size ten times greater than that of the Mnist dataset and should thus be more than sufficient in representing numbers in the imagery. Images are up-scaled or down-scaled using nearest neighbor interpolation, as is the default of the tensorflow image loader applied in this transformation.

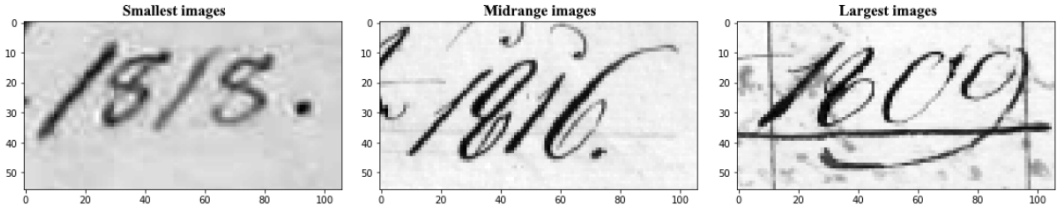

Figure 1: *Shows the rescaling of three different image sizes. The images from left to right represent a smaller image from the dataset, a mid-range and a large image, that have been scaled to the same size.*

Images are labeled according to the system explained and depicted in the exam handout with class CC having two classes and D and Y having 11 distinct classes. Examining the images in the data folder revealed an inherent structure within the order of the images as it is evident that images are indexed in a series like manner with every series being handwritten numbers from the same work of writing. The data upon import is thus shuffled to break up this inherent structure and instantiate an assumed normal distribution in the imported data. The data is split into 7680 training samples, 1920 validation samples and 2400 test samples.
During the course of this project it became apparent how big the imbalance in this dataset really is. The distribution on the 18 or not label is 96,4% and 3,6%. For the decade label it is also bad, with 94% of the dataset in only 4 out of 11 possible categorical labels. This will tried to be solved during the project.

We now turn our focus to the DIDA_30000 unlabelled collection. 1000 images are isolated and labeled manually and can be found in an attached csv file. Import and transformation of the data follow the same protocols as described for the DIDA_12000 data in short. Unlike for the DIDA_12000 data we do not split this 30k set into train, validation and test sets, but let it remain as a whole set to be used for testing CC-D-Y models trained on the DIDA_12000 split. The above is the general procedure unless anything else is specified.

Question 1

1.1 (Responsible: JHO)

SVM

A hyperplane can be used to separate two classes in a linear fashion, by creating a margin between the data points in each class, but this is not always the case. If there is noisy data present or the data is non-separable, then a simple hyperplane is not sufficient. Support Vector Machines (SVM) induce what is known as a soft margin. This allows some points to fall within the margin, by setting a specific parameter that indicates how many points can fall within the margin and by how much. If a linear separation is no longer an option then SVM can perform a kernel trick, which is used to fake a transformation of the variables, allowing the data to become separable and thus, allows for a hyperplane to be constructed. The three most common kernels are: linear, polynomial and radial, Figure 2. Whichever kernel is chosen, will depend on the data in question and which of the kernels would yield the highest accuracy. Due to the nature of SVM, it is not optimal to use when there are more than two classes in question. This is due to the fact that multiple classes are difficult to separate in a multidimensional space.

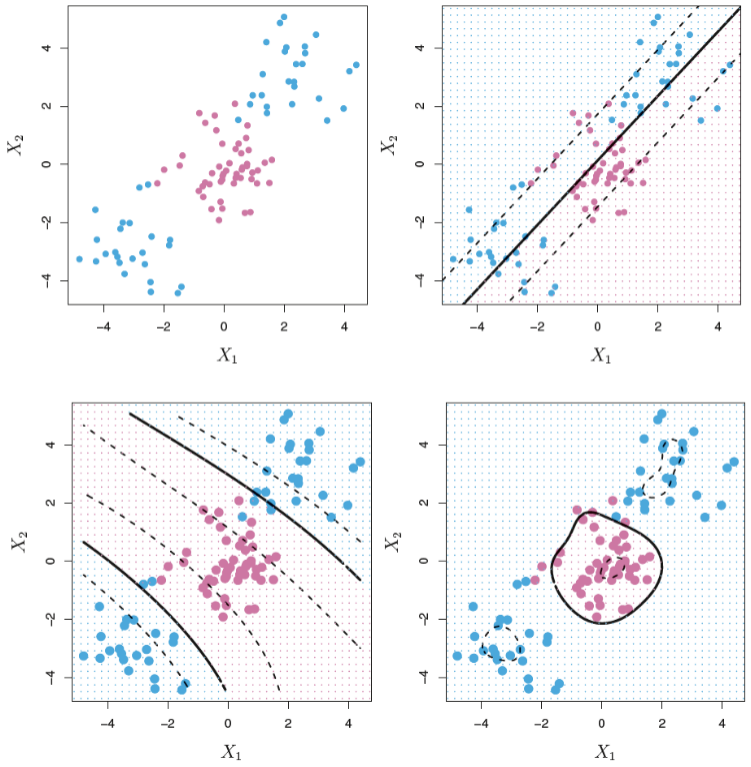

Figure 2: *A dataset in its raw form, one with a linear kernel, polynomial kernel and radial kernel.*

For classification of the CC-D-Y modeling strategy, SVM is not feasible for classifying either D or Y, as these both contain more than two classes. However, since our CC class only contains two classes, then it can be used to classify whether our pictures are from the eighteen-hundreds or not.

Ensemble Methods (Responsible: MØH)

Ensemble methods work by combining multiple models in order to improve the performance. There are different ways to utilize the combination of models:

1. Take an average of the predictions over multiple models, also called bagging.
2. Training a second model on the errors of your first model, also called boosting.

One such model that utilize the average of predictions through bagging are Random Forests, Figure 3. The way a random forest works, is by fitting multiple decision trees on sub-samples of the training dataset through bootstrapping. These decision trees are then part of a series of weak learners which all have been trained on separate, yet overlapping, subsets of the training data. In the end all of these learners will individually predict and their collective average prediction will be the output of the random forest model, Figure 3. This model averaging of outputs is what yields the good performance of random forests models, since each tree with somewhat unique training data compliment each other's strengths and weaknesses to ensemble a strong predictor model.

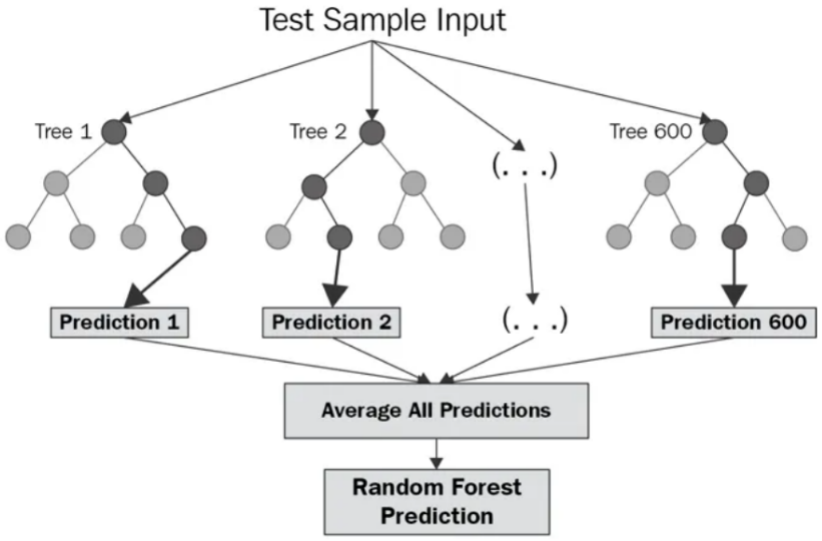

Figure 3: *A visual representation of random trees, showing the different predictions from input to the final random forest prediction [3]*

Other models train several models on top of each other, but rely on the previous models' error measures for learning. The method of utilizing information from previous error measures is called gradient boosting. In contrast to random forests, the gradient boosting model trains its weak learners sequentially and does not apply bootstrapping in training as it is done sequentially. Gradient boosting generally tends to have better performance if it can be tuned correctly. This makes gradient boosting highly applicable for Classifying the multi classes D and Y. There are different implementations of gradient boosting to choose from, as it is a popular model. For this project the model called Catboost was chosen since it appears from public benchmarks to perform well[2].

Gradient boosting methods have different parameters that need to be tuned for the best setup for the model given the select task. One of these parameters is the learning rate, which indicates the increment of the steps the model takes in the negative gradient to reduce the error during training. Another parameter is the depth of the trees that are fitted to each training iteration. The last parameter, will be the number of iterations the model has to run for. Fine tuning these parameters will be how the performance of this model is evaluated. There is a trade off between learning rate and number of iterations. When you decrease the learning rate you need more iterations, because if smaller steps are made, you need to take more steps to get to the minimum value. Increasing the iterations of the model also increases the resources needed for the model, but decreasing the learning rate helps your model from overstepping local minima. In order to fine tune the catboost model, it will be trained on different setups by "cross training" the different parameters in order to evaluate all possible

combinations. This is an extensive parameter search and if you are really thorough it can take many hours to hyper parameter search in this manner.

1.2 (Responsible: JHO)

This chapters code is available in Hand_in_NB.ipynb in section "Question 1.2".
This chapter utilize data as described under "Data management fundamentals".

For model CC we apply an SVM classifier. The classifier is evaluated on parameters; kernel function, L2 regularization factor C, kernel coefficient gamma, decision function and degrees for polynomial kernels. Initially, the model used too many parameters and degrees, and was therefore too computationally expensive and time consuming. After adjusting the parameters to a more suitable level, the model performed well, despite still taking a while to process. After evaluating several model settings, the highest accuracy on the validation set through training was found to be 98.2% when using the radial kernel, a C regularization parameter of 3 and using one-versus-rest as a our decision function shape, Table 1. Applying the determined best settings in training on concatenated training and validation set we receive an accuracy of 97.7% on the test set. This shows a slight decrease in the performance when compared to the accuracy of the validation set.

Accuracy	Kernel	C	Gamma	Decision
0.982292	RBF	3	scale	OVR
0.982292	RBF	3	Scale	OVO

Table 1: *Settings for the final SVM model*

Training the D and Y classes for the catboost model yielded different results for the two single digit classes. The results indicate the best tuning for both models will be doing 500 iterations with a tree depth of 6 and a learning rate of 0.1. This tuning was able to reach an accuracy of 56,6% for the class Y and 64.2% for class D on the validation set, Table 2. Comparing this to the SVM model it might not seem as impressive, but one should keep in mind this model is trained to distinguish between 11 classes instead of just two. It is obvious from accuracies in Table 2 that the model could possibly be improved through further tuning. Increasing tree depth could be significant in improving the models further. This is very time consuming due to the models and data sets size, and not feasible for either the time frame of the project or the available computational resources.

In order to rank and measure the performance of the combined CC-D-Y model two scores will be calculated for each sub model, namely the sequence score and the character score. The character score awards $\frac{1}{3}$ point per right predictions for each model. The sequence model will only be awarded a point if the predicted sequence from all three models is the same as all three labels. This gives a point estimate of the performance of the models, and will be the measure of comparison between the different models.

Character score	SVM CC	Catboost D	Catboost Y
12k test set	781.6 of 800	400.6 of 800	515.3 of 800
30k test set	274.0 of 333.3	135.3 of 333.3	117.6 of 333.3

Table 2: *Character scores of the performance of non-deep learning models*

	Sequence score
12k test set	895 of 2400
30k test set	160 of 1000

Table 3: *Sequential scores of the performance of non-deep learning models*

Looking to the character score, model CC using SVM performs very well when using the DIDA_12000 dataset, Table 2. Testing the DIDA_30000 dataset on the same model gives an accuracy of 82.2% , Table 2. Model D and Y each do not perform nearly as well with model D getting an accuracy of

50% and model Y getting an accuracy of 64.4% on the DIDA_12000 test set. In addition, when these models predict on the DIDA_30000 dataset samples performance drops further. Model D receives an accuracy of 40.68% and model Y 35.28%.

Assessing the sequence score further highlights CC-D-Y model inaccuracies. The score for the DIDA_12000 dataset is only 895, meaning it only received 37.29% of the possible 2400 points. The DIDA_30000 dataset performs even worse by receiving only 16 points, which means only 16% of the total 1000 points were attained, Table 3. The non-deep learning models is deemed unable to predict year-strings from images of handwriting to a satisfactory level.

Question 2

2.1 (Responsible: JJR)

Running neural networks on images can become computationally expensive, especially if there are many images of even a moderate size. Each pixel within the grid has usually three color channels (RGB) and their positioning within the grid and relations to other pixels yields information value. A proper strategy to hone in on the significant information within the network by data transformations is encouraged to reduce complexity of the task objective.

Convolutional neural networks (CNN) can improve the efficiency of computation while still retaining high accuracy and promoting translation invariance. By translation invariance the network is encouraged to generalize better to allow for detection of features within the image grid no matter its position. This is achieved using a kernel in the convolutional computations.

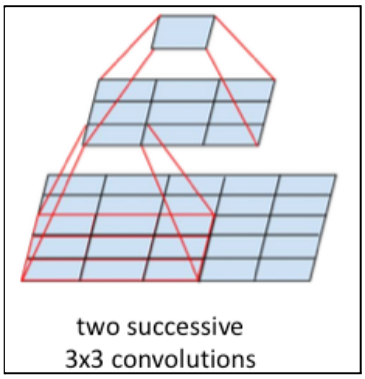

Figure 4: *The principle of convolution depicted by two three by three pixel kernels convolving inputs and reducing dimensionality [5]*

The kernel essentially ties weights together by scanning multiple input pixels at once and computing the output only once. This drastically reduces the number of computations and promotes model speed. This in turn also promotes model invariance. Using the kernel to produce the output, reduces the output dimensionality in comparison to the input dimensions, Figure 4. The output highlights significant values, almost like an intensity heatmap, and due to reduced dimensionality it now becomes faster to apply convolution on these "intense" signal values. Convolution is applied in convolutional layers within the network, denoting the kernel size in pixels and number of filters to apply for each layer. Filters are essentially individual kernels scanning for a certain pattern or mask and whilst striding across the imagery grid will output a value for the "match", the afore mentioned intensity heatmap.

For each convolutional layer in the network, with increasing depth, the layers will hone in on different feature levels, starting with low level features that are shared by multiple classes and later narrow these down to high level features which only fit a few distinct classes. This dialing in on certain features is what allows for classification by differentiating features. This is a brief overview on how CNN's functions. There are several other considerations to take into account such as data augmentation, regularization and other measures to promote performance of the CNN's.

For the CC-D-Y classification task a CNN would be very appropriate as it in theory would allow for the network to generalize on the distinct looks of the numbers depicted. With the numbers being

placed somewhat within the same region of every sample the model should successfully detect these generalized trends easily. The D and CC classes experience a lackluster representation for certain classes in the DIDA_12000 data set. This can impose difficulties of the CNN by being unable to classify these underrepresented classes and increases the risk of overfitting the overly present classes.

2.2

2.2 A (Responsible: JHO)

This chapters code is available in Hand_in_NB.ipynb in section "Question 2.2.A".
This chapter utilize data as described under "Data management fundamentals".

We have chosen to construct the CC-D-Y models with a structure inspired by the VGG16 model, [6]. We have constructed a VGG4 as our initial model with blocks applying MaxPooling to optimize initial computational costs, Figure 5. Pooling itself is an optimization of the CNN as it uses a kernel to hone in on a statistical measure, here the max value within the kernel, and produces a new output with reduced dimensionality for the measure. Pooling is important in reducing dimensionality, filtering out noise and speeding up the CNN calculations and can thus be considered as an optimization in itself. To conduct a search for appropriate optimizers the pooling layers are applied from the start as literature on VGG models emphasizes the importance of their presence in high performing architectures with fast computations. The VGG4 consists of three by three convolutional kernels and two by two pooling kernels in all blocks, Figure 5. Block 1 consists of 32 filters, 2 of 64 filters, 3 of 128 filters and the last block 4 of 256 filters. Two dense layers of 1024 and 64 units follow up the convolutions prior to the last output layer. All layers within the VGG4 use the rectified linear unit (ReLU) for their activation functions. The VGG4 architecture is reused for all CC-D-Y models with the only exception being their classifying output layers. The VGG4 architecture is for all models evaluated on the metric accuracy and model Y and D having loss measures on sparse categorical crossentropy where CC is measures measured on binary crossentropy.

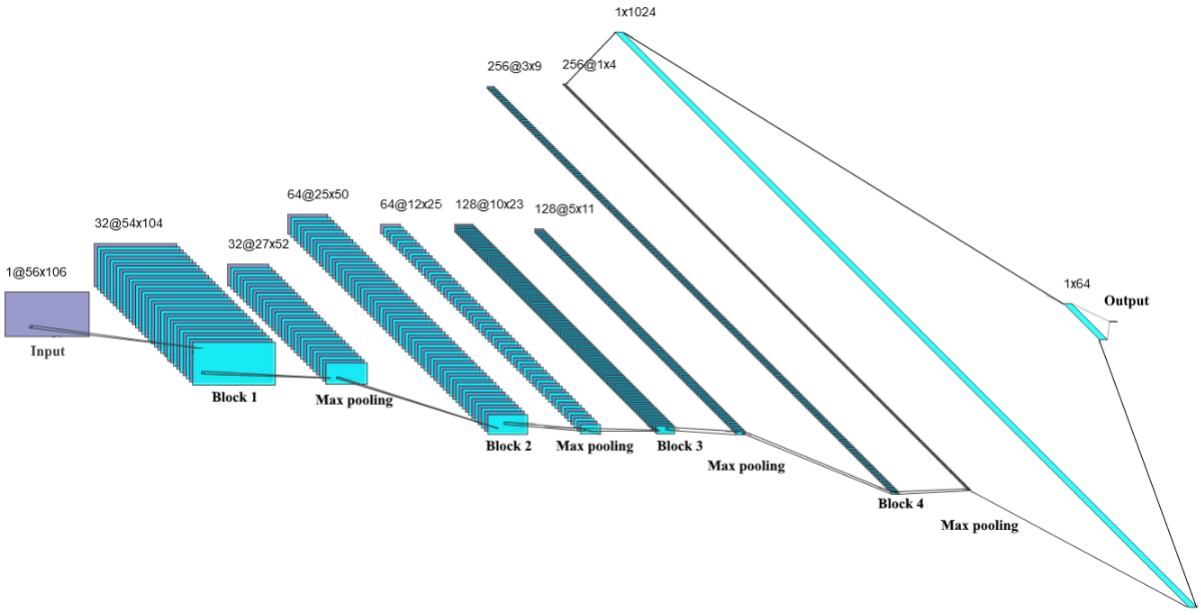

Figure 5: The base VGG4 model applied in classification of CC-D-Y

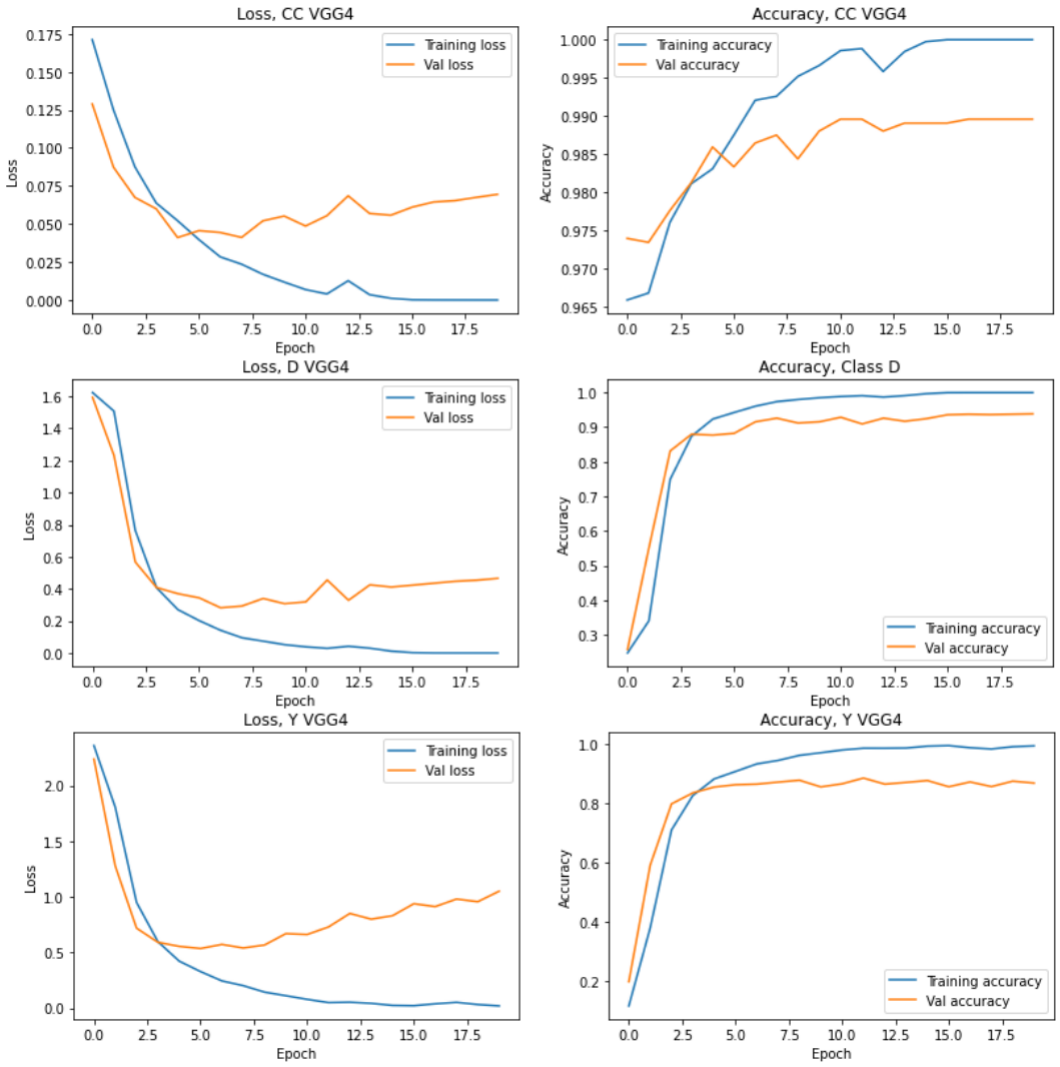

Figure 6: Loss and accuracy curves for training of VGG4 architecture CC-D-Y models. All models are trained for 20 epochs depicting loss and accuracy measures on the training and validation set.

The VGG4 architecture displays reasonable training sessions on loss and accuracy curves, Figure 6, and achieves initial good accuracy and loss levels for the validation set, Table 4. For model CC, 98.96% accuracy leaves little room for improvement however it should be obtainable for D at 93.85% accuracy and Y at 86.87% accuracy. The loss curves for the validation set all start to increase at 6 epochs and implies that overfitting is occurring and that the models could benefit from regularization measures.

	CC	D	Y
Validation accuracy	98,96	93,85	86,870
Validation Loss	0,0695	0,4655	1,0501

Table 4: Accuracy and loss values for the performance of the base line VGG4 model

Different optimizers will be investigated in the upcoming section. Numerous optimizers could be investigated but given the project's time constraints the optimizer search will be kept restrained for few motivized parameters; batch normalization, learning rate adjustments, dropout layers and early stopping.

Batch normalization is an algorithm that makes the training of the neural network more stable and faster. It does so by normalizing the activation vectors inside of the hidden layers, by using the mean

and variance of the batches. By standardizing the batches in the network, it can reduce the amount of epochs necessary and make the network more accurate and stable [4].

Learning Rate is what adjusts the weights and biases of a neural network after each epoch. Learning rate manages the speed at which the model learns thorough minimizing the loss function by gradient descend. Having a learning rate too low, means that the network might converge too slowly towards minima on the loss curve just as having too high of a learning rate can cause the model to converge too quickly and jump around the minima. It is important to note that the learning rate is not technically a form of regularization, but instead a hyper parameter [1].

Dropout in ordinary neural networks sets a chance for any of the neurons in the affected layer to be turned off randomly. In CNN's the convolutional layer has a set chance that filters will be turned off as this equates to having fewer processing units. This chance is rolled every epoch and forces the layers to "stay awake", and prevents layers from co adapting, by inducing noise within the affected layers. Through training, a retaining probability is tuned alongside the weights of each filter and at the end of training the models weights are scaled with this probability to promote select filters importance [8].

Early stopping is an easy monitoring measure that tracks a select parameter, often the validation loss, during training and within a set amount of epochs, it monitors for deterioration of the select parameter. This is costly as the algorithm needs to save model states in memory and rewrite these each time the model improves in order to keep the option to reroll models when the early stopping triggers. Targeting for validation loss prevents overfitting, one could also monitor validation accuracy to maximize this no matter the cost to the loss function. If this is desirable, then early stopping is a great tool to fulfill one's needs.

2.2 B (Responsible: JJR)

This chapters code is available in Hand_in_NB.ipynb in section "Question 2.2.B".
This chapter utilize data as described under "Data management fundamentals".

Overfitting is the phenomenon where the algorithms starts fitting excessively to the training data learning the statistical variance within the training data. This is unfavorable as this decreases the algorithm's ability to generalize on new data. It should be able to predict on new data, but is not because it is fitted to the statistical variance of its training set. Even though the data used in training the algorithm is related to the test set, by being part of the same domain, their statistical variance is unique for every single data set. The term overfitting thus fits this phenomenon well, as we fit the training data beyond our desires and needs, when we instead want to promote generalization. Regularization are measures that induce statistical variance into the CNN thus promoting the algorithm's ability to generalize. For example looking at the simple early stopping, usually monitoring for validation loss, it forces the algorithm to stop training when overfitting occurs. As the training set loss curve decreases and the validation set loss curve in turn starts ramping up, indicating that the algorithm is overfitting on training variance, the validation set and its variance becomes foreign to the algorithm, when it still should be able to recognize the validation set and its inherent variance.

The VGG4 model will be optimized using the reguralizers; batch normalization, dropout and early stopping for each of the three CC-D-Y classifier models. Further the appliance of varying learning rates will be explored. This is not strictly a regularization measure, but a hyperparameter for the cost function. Minimizing the loss curve is promotes algorithm performance and the effect of dropout and early stopping is thus greatly dependant on the learning rate in their own functioning. In tweaking the optimizers three duplicates of each algorithm optimization will be computed and evaluated using scoring of the validation set. This is done to avoid a bad weight initialization being the determining factor in evaluating each model. Optimizations will be based on a further construct of the previously optimized model, meaning that initially the VGG4 will be compared with and without batch normalization, and for the next optimization compare the batch normalized and learning rate tweaked VGG4 to the original unoptimized VGG4, and so on. Optimization choices are primarily based on comparative scoring for the validation set but the training curve will also be taken into account if found significant.

Appendix Figure 20 displays calculated validation accuracies and losses for each model optimization at the end of training. Applying optimizations yields both gain and loss for the CC-D-Y models. Model CC does not change significantly, yet is initialized in VGG4 with astounding accuracy just under 99%

leaving little in turn of possible optimizations. Model D halves its loss through optimization and gains 2% accuracy. Model Y gains 3% accuracy and halves its loss through optimizations and like model D benefits greatly through regularization optimization. All models lose accuracy yet retain reduced losses when applying early stopping (validation loss, patience 10 epochs). This may relate to unlucky initialization of weights, but likely through monitoring validation loss overfitting is prevented, which in turn reduces validation accuracy.

To visualize the effects of regularization one select algorithm for each optimizer will be displayed and commented on. Having explored three settings with three duplicates each for every of the four optimizers there simply are too many graphs to comment; the most interesting ones will be picked and evaluated upon.

Batch Normalization

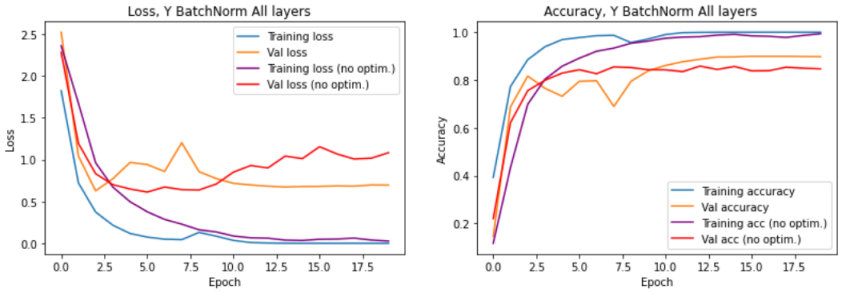

Figure 7: *Loss and accuracy curves for training of model Y with and without batch normalization on all blocks. Both models are trained for 20 epochs depicting loss and accuracy measures on the training and validation set.*

Batch normalization implemented on all blocks for model Y, Figure 7. Batch normalization is seen reducing generalization error at the 8th epoch as the loss curve is minimized and held down in comparison to no batch normalization. The normalization of layer outputs allows for the algorithm to generalize on normalized trends rather than fixating on distinct output values and in this depiction prevents overfitting on the training noise.

Learning Rate

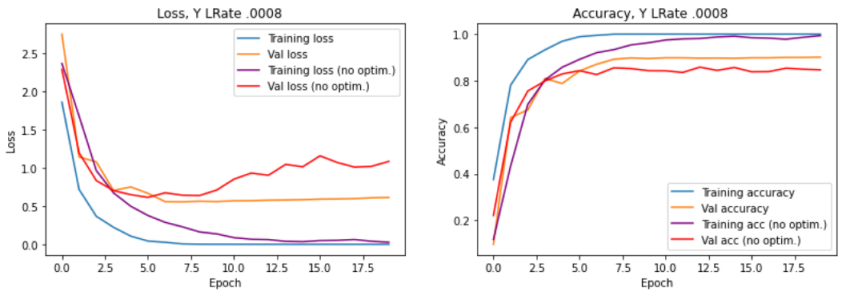

Figure 8: *Loss and accuracy curves for training of model Y with and without learning rate tuned to 0.0008 in contrast to 0.001 on the Adam optimizer. Both models are trained for 20 epochs depicting loss and accuracy measures on the training and validation set.*

Learning rate at 0.008 for the Adam optimizer in the batch normalized model Y smoothe out the accuracy and loss curves, Figure 8. Dialing the learning rate down causes stochastic gradient descent to converge slower with smaller steps allowing for a more controlled minimization of the loss curve. The same gains in accuracy from batch normalization is present but the series conveys more smoothly towards this point rather than fluctuating.

Dropout

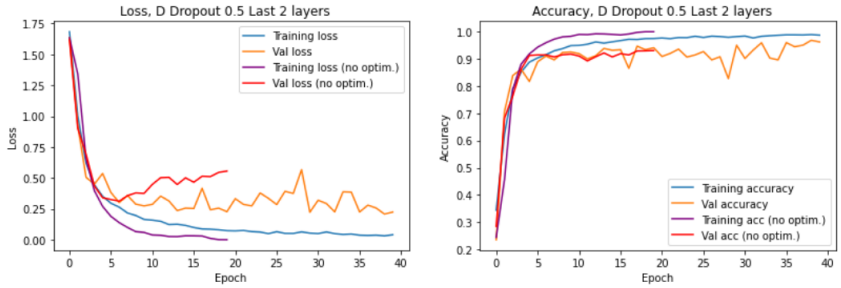

Figure 9: *Loss and accuracy curves for training of model D with and without filter dropout in the last two blocks at a dropout chance of 0.5. The dropout model is run for 40 epochs and the unoptimized for 20 epochs depicting loss and accuracy measures on the training and validation set.*

Dropout applied in the last two blocks of model D with batch normalization on all blocks and a learning rate of 0.0008, Figure 9. Filter deactivations through dropout causes the learning curves to move erratically, yet promotes loss decrease and improves accuracy. Turning filters on or off makes the curve display this jittering tendency, but tuning the weights according to the information gained from the dropout, the model still achieves to promote accuracy.

Early Stopping

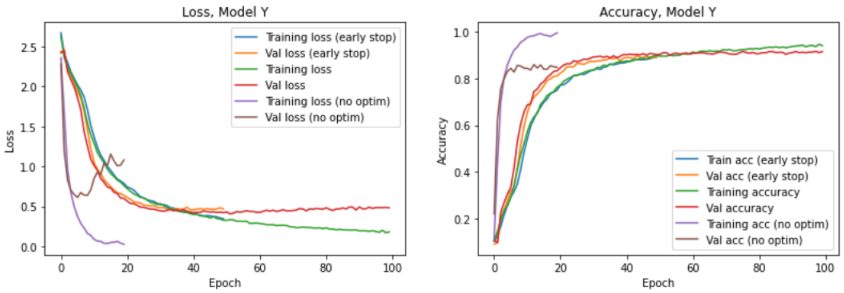

Figure 10: *Loss and accuracy curves for training of model Y with and without early stopping on an optimized VGG4 architecture in comparison to an unoptimized VGG4 architecture. Optimized VGG4's are trained for 100 epochs and the unoptimized for 20 epochs depicting loss and accuracy measures on the training and validation set. The early stopping is set to monitor validation loss with a patience of 10 epochs*

Early stopping implemented on model Y with learning rate 0.0008, batch normalization on all blocks and dropout on the last two blocks, Figure 10. Early stopping is implemented with 10 epochs of patience on validation loss and is executed at 50 epochs of training. It is clear how the model without early stopping stats increases on validation loss and the regularization is thus efficient in hindering overfitting.

2.2 C (Responsible: MØH)

This chapters code is available in Hand_in_NB.ipynb in section "Question 2.2.C".
This chapter utilize data as described under "Data management fundamentals".

Data augmentation relates to the augmentation of your data to increase training pool size. Augmentation is used to recycle data already present by inducing variance into these and essentially synthesizing more data for the algorithm to train on. This is very common in training CNN's classifying on imagery. Images can be rotated, stretched, flipped, have their contrast adjusted, and much more. These augmentations should be implemented in an appropriate manner to retain a valid representation of the class within the imagery. Increasing the size of the data set by augmentation thus induces variance in

the training phase without altering the structure of the CNN. It is a non intrusive and efficient way to minimize generalization error, its effects are similar to what we see when applying regularization. For augmenting images for the CC-D-Y model we decided that applying rotation was the only proper treatment that we had left to apply. The data are already prior to the CNN augmentation deployment augmented through resizing to the common 56 by 106 resolution and all transformed into grayscale imagery. The images vary greatly in their centering of the year string and warping, cropping etc. was deemed too risky in losing information of the object within the image. Flipping the image like so would directly misrepresent the year-string and be an invalid augmentation to deploy. Rotation is thus the only augmenter applied, shifting ever so slightly the numbers to reflect real life variations in human writing, Figure 11. The augmentation is applied using a rotational shift up to 29 degrees from the norm.

Figure 11: *Rescaled image from the training set split of DIDA_12000 depicted when augmented with different rotation settings for a shift up to 29 degrees in the angle.*

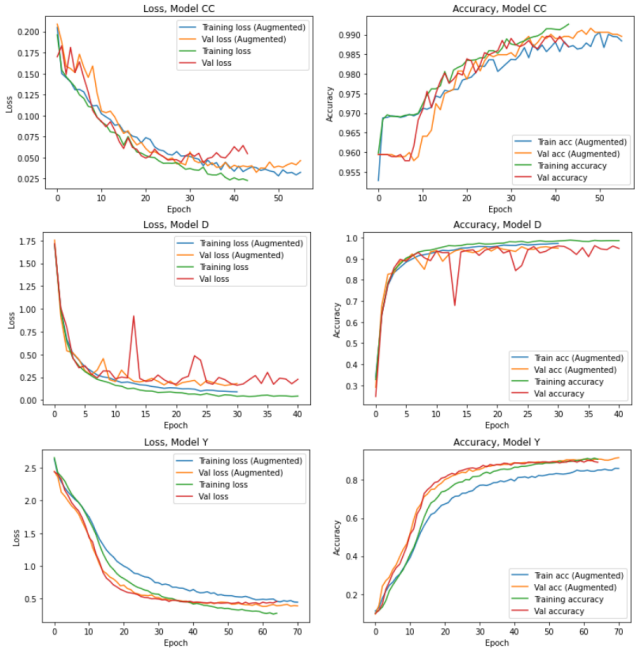

Figure 12: *Loss and accuracy curves for training of optimized CC-D-Y models with and without data augmentation during training. All models are trained for 70 epochs depicting loss and accuracy measures on the training and validation set. Training is initialized with early stopping on validation loss with 10 epochs patience and causes some models to end training before reaching 70 epochs.*

Applying image augmentation in the CC-D-Y models yields accuracies across the board, slightly

for CC and D but significantly with 2% validation accuracy for Y, Figure 12 and Table 5. All models have their loss minimized like so with augmentation enabled and it is evident that the augmentation is efficient in promoting translation invariance. Loss and accuracy curves display trends in minimizing loss and promoting accuracy in comparison to the model without augmentation, Figure 13.

Validation	Model CC	Model D	Model Y
With augmentation	Acc 0.9895 Loss 0.0465	Acc 0.9494 Loss 0.1805	Acc 0.9171 Loss 0.3898
without augmentation	Acc 0.9869 Loss 0.0546	Acc 0.9484 Loss 0.2266	Acc 0.8932 Loss 0.4576

Table 5: *Validation accuracies and loss for for VGG4 optimized CC-D-Y models with and without data augmentation.*

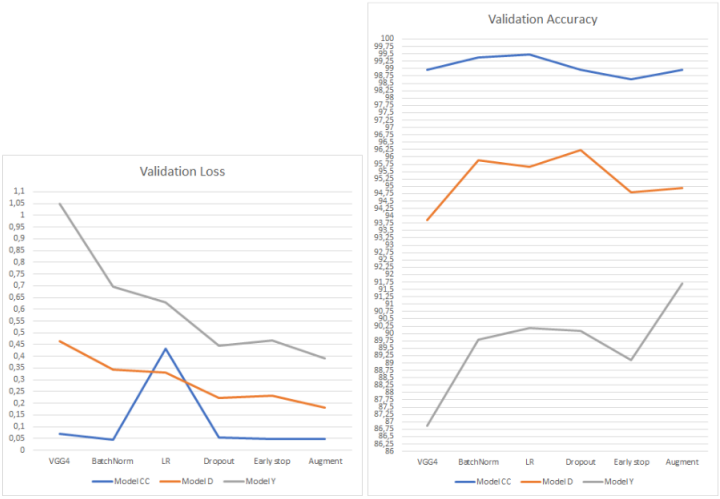

Figure 13: *Validation and accuracy curve trends through optimizing VGG4 models for classifiers CC-D-Y. Loss and accuracy on the validation set is depicted for the model upon applying new optimizers in a sequential manner.*

Optimization of the VGG4 for the CC-D-Y classifiers through regularization and data augmentation yields great gains in terms of minimizing loss, reducing overfitting and improving accuracy, Figure 13. It is concluded that the initial VGG4 architecture, Figure 6, has been regularized properly and achieved better generalization performance as a result of incorporating these measures. Even though for model D and CC accuracies decrease slightly their loss is minimized and reduces risk of overfitting. Decreased accuracy can be due to the random initialization of weights and minimized loss is instead prefered. For classifiers CC-D-Y the model architectures having applied all optimizations are deemed the optimal ones as they successfully minimize loss while promoting accuracy. All final models are based on the VGG4 architecture, figure 13, they differ slightly in their optimizations: All models learn at a rate of 0.0008 using the Adam optimizer and batch normalization occurs in every block. Dropout at a rate of 0.5 is for model D applied in the last two blocks while model CC and Y utilize it in all blocks. Common for all models is that their training on augmented training data is stopped using early stopping on validation loss with a patience of 10 epochs.

2.2 D (Responsible: MØH)

This chapters code is available in Hand_in_NB.ipynb in section "Question 2.2.D".
This chapter utilize data as described under "Data management fundamentals" aswell as a new test set introduced through code in Hand_in_NB.ipynb section "Question 12k, label CC 50/50, import".

Well performing models for image classification often require large amounts of data to train on. In this case there are 12000 images to train on, which might at first seem like quite a big amount, and as it can be seen from the results above, it is enough to produce some well performing models. However, there are some public models that have been trained with very big amounts of data. To compare, there is the imagenet dataset which is a dataset that consists of 14 million images with 1000 classes. The public models, like the VGG16 model, have been trained on this imagenet dataset and achieved an astounding 92.7% accuracy with 1000 classes. This is an amazing performance, but how does this relate to classifying centuries, decades and years? Through transfer learning the weights and biases from the VGG16 model can be transferred over to predict on these classes as well, the idea being that the transferred model does not have to train on millions of new images, but is able to quickly adapt to the new classes. This results in a model that does not necessarily need as much data, and it does not need to run for nearly as long. This will be implemented to try and compete with the previous model implemented for the CNN model with VGG4 setup. There are two ways of implementing a pre-trained model as described by Tensorflow [7]:

1. ***Feature extraction:*** *"Use the representations learned by a previous network to extract meaningful features from new samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained model so that you can repurpose the feature maps learned previously for the dataset."*

2. ***Fine Tuning:*** *"Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task".*

In order to use the pre-trained models from tensorflow, the data will for this task be rescaled to three color channels instead of the single channel used in the rest of the rapport. In this section both implementations of pre-trained models will be explored; one where only a new classifier will be added and another where the first 100 layers will be frozen and the rest will be trained for the DIDA data. The model will be trained with early stopping on validation loss to achieve the best result while avoiding overfitting.

Validation	Model CC test acc, (validation acc)	Model D, (validation acc)	Model Y, (validation acc)
Feature Extraction	99,08%, (94%)	67,9%,(67,08%)	65,9%,(65,2%)
Fine Tuning	98,5%, (98.9%)	67,3%, (68,4%)	60,0% (61,35%)

Table 6: *Test and validation accuracy for the CC-D-Y models initialized through transfer learning on the VGG16 architecture.*

Comparing the results from both models to the performance from the VGG4, the pre-trained model is able to perform on a similar level as on class CC, Table 6. The performance for classifying D and Y is however far from the performance of the VGG4 model, Table 6. Looking at the train and validation accuracies it is clear that the model quickly overfits for the CC label, just as the other models, Figure 14. However looking at the model for the Y label it trains rather nicely, but is still not able to compete with the CNN without overfitting.

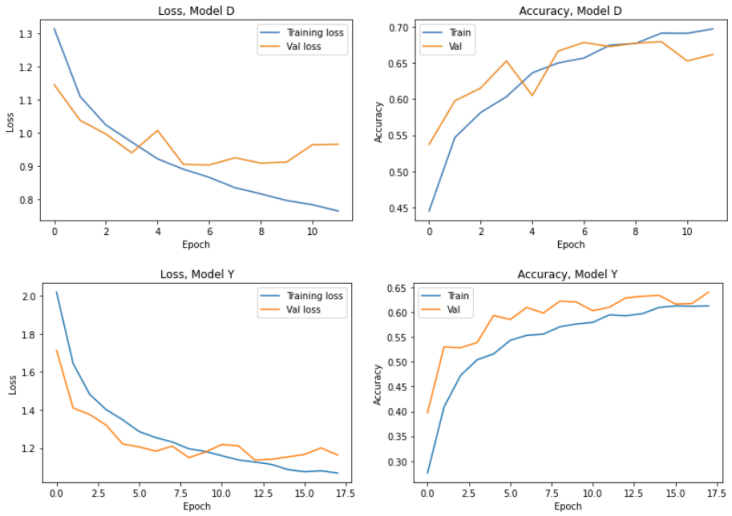

Figure 14: *Loss and accuracy curves on validation data through training on model D and Y using transfer learning.*

In this project since the data for model CC is so imbalanced, transfer learning could be beneficial for predicting the CC label. Out of the 12.000 data points, only around 390 are not in the 1800 class. So the idea for using transfer learning for this class is to make a subset of the original data, which is evenly distributed with the labels for the CC class. We turn our focus to the DIDA_12000 labeled collection and create a 50/50 split dataset with respect to the class CC. All 389 images that do not classify as 18 in CC, class 1, are copied to a new directory and supplemented by 389 images of CC class 0. Importing and transformation of the data follow the same protocols as described in 'Data management fundamentals'. This will be combined with the augmentation layer from task 2.2.C in order to increase the small 50/50 dataset. Then transfer learning is used on augmented 50/50 to see if it can learn enough to classify the 1800 better than the other model. The new dataset is small at 778 samples prior to augmentation. Tackling the training set for model CC in this manner would in theory allow for better utilization of the pre-trained weights of the VGG16-model.

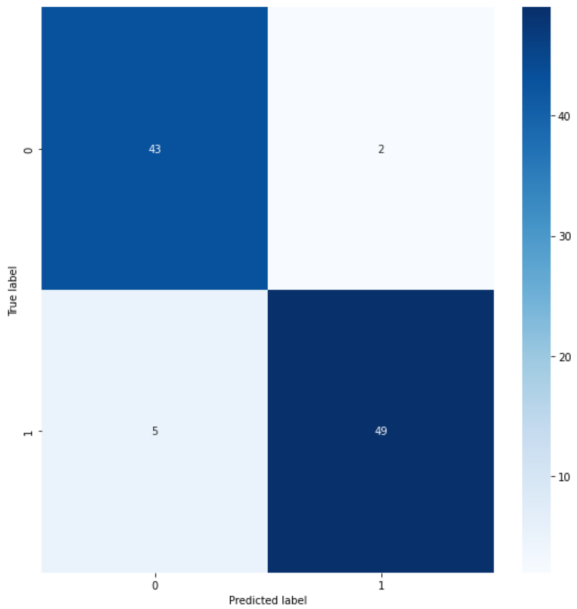

Figure 15: *Confusion matrix for model CC on valdiation data using the new 50/50 data for training.*

The model CC was able to achieve a bit lower accuracy for the CC label using the new 50/50 data. Looking at the distribution of the predictions on the validation set, it is clear that the new model trained with the augmented 50/50 set, is able to misclassify equally between "18 or not", Figure 15. The equal distribution in misclassifications eludes to the model not having any overfitting on one specific label, but it instead being able to generalize. This makes this approach very important, to be able better generalize on foreign data. This perceived ability of generalization supports the argument for the CC transfer model to be incorporated in the collective CC-D-Y classifier.

2.3 (Responsible: MØH)

This chapters code is available in Hand_in_NB.ipynb in section "Question 2.3".
This chapter utilize data as described under "Data management fundamentals".

We chose to proceed with accuracy measurements on test data using our own optimized VGG4 models for the Y and D labels and the transfer model for the CC labels. Model CC gets tested on synthetic RGB whilst the model D and Y are tested on grayscale however all on the exact same image samples. The models are evaluated according to individual character scores and collective sequence scores as described in the exam handout. Character scoring is a mild measure honorating each classifier with a third of a point for correct answers whereas the sequence score only honerates the collective classifier if all three sub classifiers are correct in their prediction. Character scoring can therefore be seen as an ordinary accuracy score of percentage wise correctly predicted samples. Given the solution from 2.2.D, it is not possible to use the DIDA_12000 set as test set, because there might be duplicates in the training sets used in the big test set. Therefore, the scoring will be performed on the self-labeled subset of DIDA_30k as seen applied in non-deep learning, question 1.2.

Character score	Model CC test acc	Model D	Model Y	Overall
30k labeled set	288,6 of 333.3	275.3 of 333.3	262.6 of 333.3	826,5 of 1000

Table 7: *Character score for optimized VGG4 models on DIDA 30k test sets.*

	Sequence score
30k labeled set	620 of 1000

Table 8: *Sequence score for optimized VGG4 models on DIDA 30k test sets.*

The CC-D-Y collective model achieves a sequence score of 620 of 1000 points on the test set, which equates to 62% of the available points, Table 8. Using the sequence measure imposes tough success criteria on the model and the sequence score will never be better than that of the weakest model in the CC-D-Y model composition. So with respect to the weakest classifiers correct predictions, the two remaining classifiers can help calculate the relative collective performance when imposed with this threshold from the weakest model. We can look to the character scores in Table 7 to aid calculating the relative sequence score. Model Y achieves a character score of 78.8% on the DIDA_30k test set and it being the weakest link denotes that with the select CC-D-Y model composition the sequence score will never exceed this threshold. This equates to the CC-D-Y model composition having scored, with respect to the weakest classifiers threshold at 78.8%, a sequence score of 85.2% relative to the scoring threshold imposed.

Question 3

3.1 (Responsible: JJR)

This chapters code is available in Hand_in_NB.ipynb in section "Question 3.1".
This chapter utilize data as described under "Data management fundamentals".

In CNNs it is possible to visualize the output layer from the convolution with respect to each filter, displaying the output as a heat map of excitatory areas of the input images. This is done with regards to a select filter patterns preference in detecting features, Figure 16. This heatmap output of excitatory response can in turn be displayed atop on the original input image with a level of transparency, to then create an overlaying heatmap on the input, Figure 17. The patterns and features were all recognized based on a certain pattern for the select filter and these can be somewhat visualized by generating images that excite the select filter, Figure 18. This allows one to get an idea of what patterns filters scan for in constructing their output feature layers.

These visualization measures provide an insight into how the CNN treats images and classes, but it is not a tool for improving model performance. It is rather a powerful tool in understanding the networks decision making and will allow people unfamiliar with the field of computer science to relate to the work that a CNN produces by providing a visual representation rather than a statistical one.

For model CC we see the last feature map gets excited in several different areas, despite CC only referring to the century of the images, Figure 16. The reason for this left- and right side emphasized spread could be that the model checks whether the year is in the eighteen-hundreds or not, whilst the right side is focused on checking if 18 is part of a year-string.

Models D and Y are slightly different from one another, as they each look for different areas of the images, Figure 16. D focuses mainly on the middle area which is where the decade is and Y as expected looks mainly towards the end where the year number is. Compared to model CC, these two are not as spread out as they do not need to look elsewhere in the images for their goal. They mainly only need to look at their own specific areas, which as the heatmaps indicate, are also the ones that are the most exciting for those models. Worth remembering is that there are hundreds of filters in the network and drawing direct conclusions on single visualized layers should be proposed very cautiously.

Feature maps

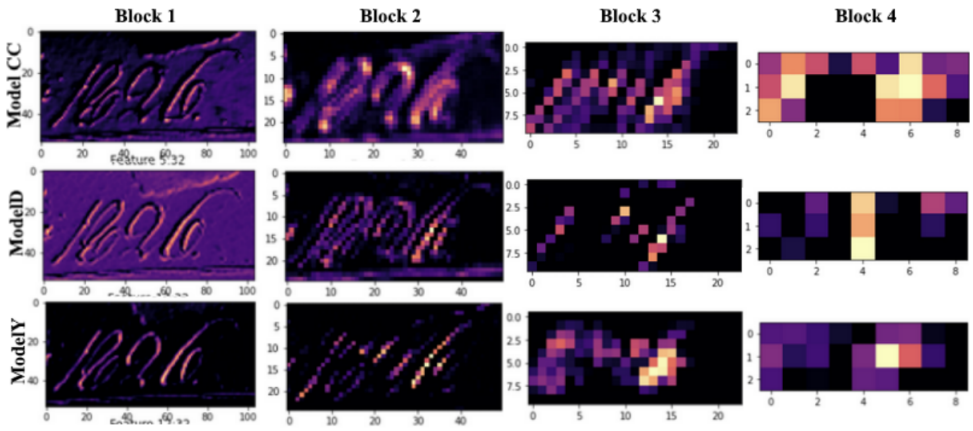

Figure 16: *Visualization of output feature maps for a select filter, in each of the four blocks for all CC-D-Y models. Intensity in outputs is denoted using the magam color ramp, ramping from low entensity purple to high intensity yellow.*

For model CC, we can see the heatmap get excited in several different areas, despite CC only referring to the century of the images. The reason for this left and right spread is that the left one is when the model checks whether the year is in the eighteen-hundreds or not, whilst the right side is focused on checking the number present in the picture. The reason for this is if there are more or less

than four numbers present, the model will assume the number to not be from the eighteen-hundred. This is the reason that model CC gets excited from several areas in the images.

Models D and Y are slightly different from one another, as they each look for different areas of the images. D focuses mainly on the middle area which is where the decade is and Y, as expected looks mainly towards the end where the year number is. Compared to model CC, these two are not as spread out as they do not need to look elsewhere in the images for their goal. They mainly only need to look at their own specific areas, which as the heatmaps indicate, are also the ones that are the most exciting for those models.

Heat Maps

Figure 17: *Visualization of filter activations overlaid as a heatmap on the input imagery in block 4 of the CC-D-Y models. The heatmap color ramps from low intensity blue to high intensity red.*

Heat maps is a good way of visualizing what the models actually sees when its looking at the picture. As it can be seen from figure 16, for model CC the model is looking for something in the left side of the image. The model D heat map shows a layer that is looking for something around the decade digit. The decade digits should be looking for something around the century digit as well, because the decade and year digits require the sequence to be 4 digits or else it categorized in the "wild-card" category. The same with model Y, its looking for something around the 18 and then around the 6.

Pattern Visualization

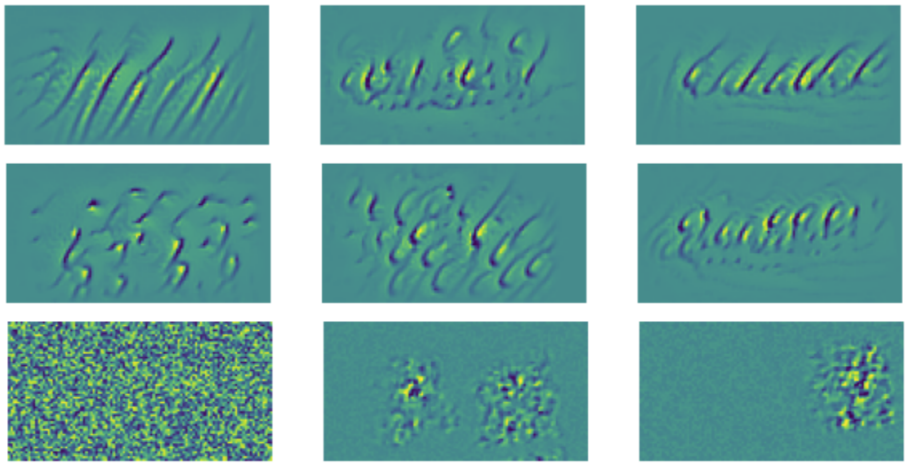

Figure 18: *Visualization of six of block fours 256 filter masks in model D. Images are generated and depicts textures that excite each of the filters individually.*

Visualizing the patterns of filters allows one to get an understanding of their objective, Figure 18. In the middle it could seem the filter is looking for a number 8, which makes sense since all three models are dependent on the picture having a digit in the sequence. Another interesting thing is that the filters are not trying to detect digits close to the borders which could indicate that the model has learned what the approximate placement of year-strings is.

3.2 (Responsible: JHO)

This chapters code is available in Hand_in_NB.ipynb in section "Question 3.2".
This chapter utilize data as described under "Data management fundamentals".

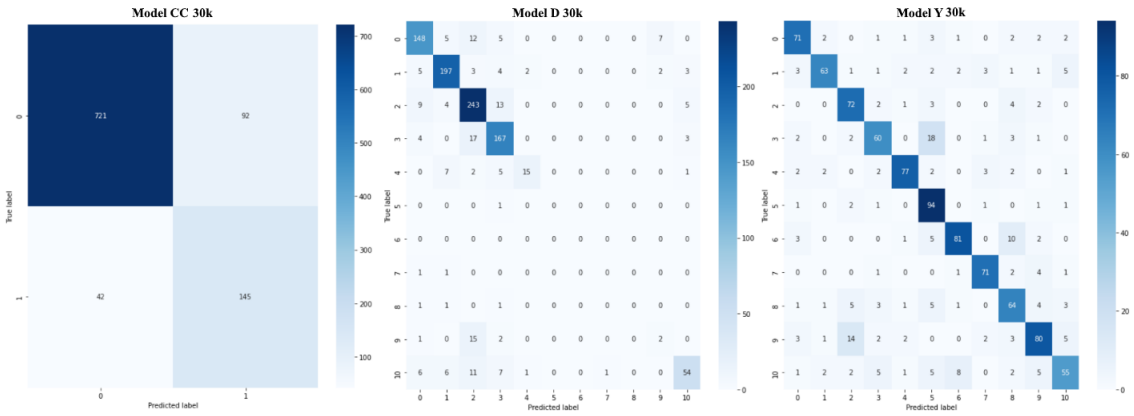

Figure 19: *Confusion matrices for the predictions from the models on the 30k data*

Predictions on the DIDA_30k test set has been visualized in confusion matrices for an improved overview of model predictions, Figure 19. The confusion matrix for model CC shows that on unseen data, the model CC still has issues with labeling data outside of eighteen-hundred, but the model seems to be well balanced in its missclassification eluding to it not having overfit. Having a well balanced model is preferable contra having an overfitting model, that always predicts the same. The matrix for model D shows that the model has a hard time predicting data outside the first four decades, which makes sense from the inherent imbalance in the data set for this labels classes. Further model D looks to have surprisingly overfit to the label 2 as almost all misclassifications occur on this label. In the test set label 2 is massively represented and it might be likely that this is the same case in the DIDA_12000 training data. The Y model appears to be able to categorize the data quite nicely as all 11 classes are well represented. Model Y only seems to be struggling with labels 3, 6 and 9 on the test set but is still able to maintain a high accuracy collectively. Missclassifications once again occurs heavily on label 2 when the true label should be 9. As for model D this may elude to the algorithm having a hard time differentiating the inherent variance of human writing when writing "2" or "9" as the upper hook can deviate in both "2" and "9" while the foot in "2" can at times be vague.
Due to the inherent time constrains imposed by the examination, models and evaluations could still benefit from further investigation. There is a series of optimizations and hypothesises that would be interesting to investigate further. In short, these topics would be:

1. The first thing would be to manually label more of the 30k dataset. This would help the classifiers in different ways. It would allow the possibility of making a more balanced dataset for training, by taking a subset of the 30k dataset that is not 1800 and adding to the training data. More labeling would also benefit the models in being able to mix more hand writings in the training data to make the model generalizing.

2. For the D classifiers the presence of all D classes were not present as observed in figure 19, as the numbers 5, 6, 7, 8 and 9 were barely present in the test sets. With the current implementation, a hypothesis could be raised questioning whether the model D is inadequate in being able to achieve its goal to classify 11 classes and instead is only able to predict for six classes, due to the inherent imbalance of the dataset.

3. Other performance metrics could be incorporated in order to better capture the performance of the models, due to the imbalance of the dataset. Accuracy as a metric does not always capture imbalanced models. Recall as a performance metric should also be considered.

4. Finally, it would have been better to include additional hyperparameter searches within our models; kernel functions, L2 regularization, and others. An extended hyperparameter search

would likely have improved the predictions of our models and thus result in better accuracies and scores.

References

[1] Jason Brownlee. *Learning Rate for Deep Learning Neural Networks*. URL: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[2] Catboost. *Catboost benchmarking*. URL: https://catboost.ai/#benchmark.

[3] CFI. *Random Forest*. URL: https://corporatefinanceinstitute.com/resources/knowledge/other/random-forest/.

[4] Johan Huber. *Batch Normalization in 3 levels of understanding*. URL: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338#b93c.

[5] Jeremy Jordan. *Common Architectures in Convolutional Neural Networks*. URL: https://www.jeremyjordan.me/convnet-architectures/.

[6] Karen Simonyan and Andrew Zisserman. "Very deep Convolutional Networks for large-scale Image Recognition". In: (). URL: https://arxiv.org/pdf/1409.1556.pdf.

[7] Tensorflow. *Transfer Learning and Fine-Tuning*. URL: https://www.tensorflow.org/tutorials/images/transfer_learning.

[8] Haibing Wu and Xiaodong Gu. "Towards Dropout Training for Convolutional Neural Networks". In: (). URL: https://arxiv.org/ftp/arxiv/papers/1512/1512.00242.pdf.

Appendix

Model CC	VGG4	BatchNorm		Learning rate		Dropout 0.5		Early stopping
	acc 0.9896	2 first blocks	acc 0.9906	0.001	acc 0.9942	2 first blocks	acc 0.9890	-
	loss 0.0695		loss 0.0883		loss 0.0414		loss 0.0721	
	-	2 last blocks	acc 0.9921	**0.0008**	**acc 0.9947**	2 last blocks	acc 0.9895	**acc 0.9864**
			loss 0.0436		**loss 0.0431**		loss 0.0646	**loss 0.0466**
	-	**All blocks**	**acc 0.9937**	0.0005	acc 0.9932	**All blocks**	**acc 0.9895**	-
			loss 0.0448		loss 0.0389		**loss 0.0548**	
Model D	**VGG4**	**BatchNorm**		**Learning rate**		**Dropout 0.5**		**Early stopping**
	acc 0.9385	2 first blocks	acc 0.9510	0.001	acc 0.9588	2 first blocks	acc 0.9354	-
	loss 0.4655		loss 0.4738		loss 0.3272		loss 0.4522	
	-	2 last blocks	acc 0.9427	**0.0008**	**acc 0.9567**	**2 last blocks**	**acc 0.9624**	-
			loss 0.4125		**loss 0.3297**		**loss 0.2239**	
	-	**All blocks**	**acc 0.9588**	0.0005	acc 0.9447	All blocks	acc 0.9531	**acc 0.9479**
			loss 0.3446		loss 0.3287		loss 0.2404	**loss 0.2325**
Model Y	**VGG4**	**BatchNorm**		**Learning rate**		**Dropout 0.5**		**Early stopping**
	acc 0.8687	2 first blocks	acc 0.8859	0.001	acc 0.8807	2 first blocks	acc 0.8557	-
	loss 1.0501		loss 0.9817		loss 0.7788		loss 0.9938	
	-	2 last blocks	acc 0.8427	**0.0008**	**acc 0.9020**	2 last blocks	acc 0.8770	-
			loss 1.2308		**loss 0.6301**		loss 0.5904	
	-	**All blocks**	**acc 0.8979**	0.0005	acc 0.8864	**All blocks**	**acc 0.9010**	**acc 0.8911**
			loss 0.6955		loss 0.6771		**loss 0.4438**	**loss 0.4676**

Figure 20: *Optimizer search on VGG4 models. Bold font indicates optimal parameter values.*