Report 2

Deep Learning and Neural Networks

Yizhen Dai (s2395479),
Anne Liebens (s1838458),
H.C. (Bram) Otten (s2330326)

April 26, 2020

Contents

1 Introduction **1**

2 Task 1 **2**
- 2.1 Base Model . . . 2
- 2.2 Initialization optimizers . . . 2
- 2.3 Activation . . . 3
- 2.4 Complexity . . . 3
  - 2.4.1 Regularization . . . 3
  - 2.4.2 Model Size . . . 4
- 2.5 Learning Rate . . . 4
- 2.6 Random Walk . . . 4
- 2.7 Results on test data . . . 4

3 Task 2 **4**

4 Task 3 **5**
- 4.1 Base models . . . 5
- 4.2 Changing from two outputs to one output . . . 6
- 4.3 Loss function . . . 7
- 4.4 Image pre-processing . . . 7
- 4.5 Final model . . . 7

5 Conclusion **8**

References **9**

1 Introduction

In this assignment, we used existing modules, Keras and TensorFlow, in Python to implement neural network models. The key objectives of this assignment are:

- Learning the basics of existing modules Keras and TensorFlow,
- Practicing with simple Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs),
- Developing a CNN for the “tell-the-time” problem.

In the first task, instead of building the classification algorithm from scratch as in assignment 1, we constructed MLPs and CNNs for the well-known MNIST and perhaps less known Fashion MNIST datasets with the Keras and Tensorflow libraries in Python. Both datasets contain 70 000 gray-scale images of size 28x28, split into 10 categories. They are both split into 60k images for training and 10k for testing. MNIST is a famous dataset consisting of handwritten digits and is often used to illustrate examples of image classifiers (*MNIST* 2020). Fashion MNIST is a dataset of article images from shopping website Zalando (*Fashion MNIST* 2020).

The following task was to change the input by permuting 28x28 pixels of all images with help of a single random permutation of all 784 pixels of both datasets (MNIST and Fashion MNIST). We tested both networks on permuted versions of the training sets and compared their accuracy on the test sets.

In the last task, we developed a deep CNN on a big collection of images of a traditional clock to tell the time shown on these images. The provided dataset consists of 18000 150x150 gray scale images of analog clocks in different orientations. Labels first indicate the hour and then the minute.

2 Task 1

For task 1 we were asked to experiment with the MLP and CNN implementations on both MNIST and Fashion MNIST data with different options of initialization, activation, optimizer, regularization (including dropout), and network structures (e.g., layer quantity, type and size), as well as hyperparameters using the Keras package.

The purpose of this task is to gain some practical experience with tuning networks, running multiple experiments and documenting the findings.

A train/validation split of 0.15 was used in the model. The validation data are used for tuning hyper-parameters and avoiding overfitting. After model selection, we used test dataset to validate our choices of parameters.

A large batch size was applied in the beginning of our experiments. If training was unstable or the final performance was disappointing, we would use a small batch size instead.

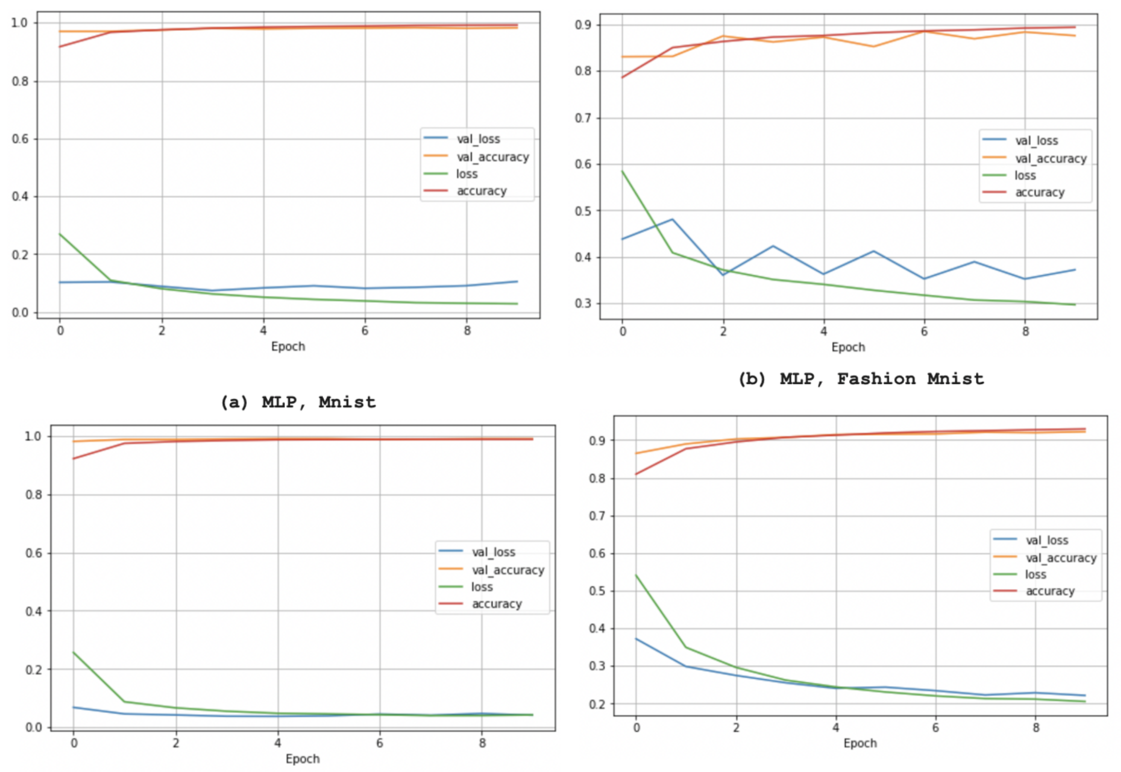
The default number of epochs is fairly low, only 10. Based on the loss curve of train and validation data, we adjusted the model. We tried to avoid relying on early stopping since it simultaneously affects how well our models fit training set and validation set.

2.1 Base Model

We used the examples shown in `github.com/keras-team/keras/` as the base models with the following set up:

- No L1 or L2 regularization but with Dropout
- `Glorot uniform` initialization
- Rectified linear units (ReLU) as activation function except the output layer
- `RMSprop` as optimizer

The results are shown in Figure 1. CNNs perform better on both datasets, with 0.990 and 0.922 accuracy on MNIST and Fashion MNIST validation data, compared to 0.982 and 0.876 for MLPs. Also, models converge more quickly on MNIST than on Fashion MNIST, which is not surprising since pictures in fashion data are more complicated than handwritten digit figures.



(a) `MLP, Mnist`

(b) `MLP, Fashion Mnist`

(c) `CNN, Mnist`

(d) `CNN, Fashion Mnist`

Figure 1: Loss and accuracy for base models on train and validation dataset.

2.2 Initialization optimizers

Since our models are not too complicated nor deep, we assumed that initialization and optimizers would not play a significant role in the models. We tested following initialization and optimizers on both dataset and both models, focusing on the accuracy and training time for converging.

- Initialization:
  - Glorot uniform (base)
  - Standard normal
  - Glorot normal
  - LeCun uniform
- Optimizers:
  - `RMSprop` (base)
  - Stochastic gradient descent optimizer (`SGD`)
  - `Adam`, a combination of `RMSprop` optimizer and momentum
  - `Adadelta`, an improved version of the `Adagrad` optimizer

`SGD` is the most basic one and practically not used often nowadays. It is expected to work decently since the variables in our cases are pixels, which are of the same scales. `Adadelta` introduces penalization so it decrease the learning rate for those parameters that are updated frequently. It was shown to work well with sparse data, which is not our case.

The experimentation shows that different initialization does not make a huge different in our case, which is consistent with our expectation.

Also, results are similar for different optimizers. However, `SGD` and `Adadelta` do not converge as quickly and stably as `RMSprop` and `adam` on MLPs for Fashion MNIST. This may be due to the default learning rate is a bit off. We test learning rate later.

2.3 Activation

Activation is an important part of neural network layers since it handles the non-linearirty. This way, neural networks can model more complicated relationships rather than mere linear combinations. We tested the following activation functions: `ReLU` (base), `tanh`, `sigmoid` and `linear`. The accuracy for validation set is shown in Table 1. As shown in the table, `ReLU` performs best for both CNN models. This is consistent with the general recommendations for CNNs. We can also see that `linear` does not perform well with MLPs, indicating mere linear combinations of input variables are not sufficient for correct categorizing.

	MLPs		CNNs	
	MNIST	Fashion MNIST	MNIST	Fashion MNIST
ReLU	0.982	0.876	0.99	0.922
tanh	0.978	0.887	0.985	0.91
sigmoid	0.977	0.88	0.983	0.882
linear	0.921	0.824	0.978	0.895

Table 1: Accuracy for different activation functions on the validation dataset.

2.4 Complexity

In this part, we adjusted the model complexity with regularization and structures of the models.

2.4.1 Regularization

First, we tested regularization (L1, L2, Dropout) based on the base models.

- No L1 or L2 but with Dropout (base)
- No L1 or L2 and no Dropout
- L1(0.01) with Dropout
- L1(0.1) with Dropout
- L2(0.01) with Dropout
- L2(0.1) with Dropout

Dropout means that a certain percentage of particular layer will be deactivated during training. Theoretically, this improves generalization because it forces the layer to learn with different neurons the same "concept". During the prediction phase the dropout is deactivated.

L1 and L2 regularization is commonly used in the regression problem. It combats overfitting by penalizing too big parameters.

The experimentation shows there is hardly any overfitting problem in the base models. Decreasing the complexity of the base models by adding L1 or L2 regularization terms hurt the model performance on both datasets. And, more remarkably, L1 regularization could decrease the validation accuracy from about 0.9 to about 0.1. This is because L1 can actually set the parameters to 0. If there is no overfitting issue, the L1 regularization term might dramatically decrease the model performance. Similarly, we see that getting rid of Dropout increase the validation accuracy for all the models, indicating a potential underfitting problem in the base models.

Based on our experiments, dropout is preferred to L1 and L2 regularization when it comes to preventing overfitting. With increasing numbers of parameters, dropout may play a significant role in combating overfittig. We would further experiment with it in Task 3.

2.4.2 Model Size

We tried different numbers and sizes of convolution filters for CNNs using cross validation:

- Number of convolution levels: 1 or 2
- Number of neurons per convolution level: 10, 100 or 500

The ranking of validation accuracy for combinations are shown in Figure 2. As shown in the table, two layers with 10 neurons performs best, followed by 2 layers of 100 neurons. One layer with 500 neurons performs worst. Generally, we get better accuracy by increasing the number of layers instead of the number of neurons per layer. In our case, increasing the number of layers give us better results than increasing the number of neurons per layer.

	10 neurons	100 neurons	500 neurons
1 layer	5	3	6
2 layers	1	2	4

Table 2: The rank of validation accuracy for combinations of different number of layers and neurons

2.5 Learning Rate

Based on the recommendation from the Deep Learning course from Standard Univeristy (taught by Andrew Ng), tuning number and sizes of layers and not as important as learning rate α.

The optimal learning rate, based on the textbook, is generally about half of the maximum learning rate (i.e., the learning rate above which the optimizer diverges). We started with a very low learning rate (e.g., 10^{-5}) and gradually increasing it up to a very large value (e.g., 1). We chose the learning rate at which the loss starts to climb as the optimal learning rate, and re-train the model with the chosen learning rate.

The experiments show that the validation accuracy for all models are below 0.3, with a big learning rate as 0.1. It means the model does not reach optimal but rather diverge from it. With a small learning rate of 10^{-4}, the model converges smoothly but also slowly. Based on the experimentation, we selected 10^{-4} as the learning rate for MLP Fashion MNIST, and 10^{-3} for other models.

2.6 Random Walk

After all the above mentioned experiments, we would like to find our optimal models. We did not use a grid search for every setting. To be specific, we followed “coarse to fine” strategy, i.e. trying random values at first. This way, we could test more options for each parameter. Another detail in our experiments is that we used the appropriate scale for each parameter based on our prior knowledge.

The best settings based on the validation accuracy during our trials are shown in Table 3.

After changing the layers and parameters, we re-tune the learning rate. Based on the validation accuracy, we selected the learn rate 0.001 for CNN model on digit data, and 0.0001 for all other three models.

2.7 Results on test data

In summary, we tested the influence of different settings. Then we picked models with more layers and neurons and cut it down based on the validation accuracy. The accuracy results are shown in Table 4. The results show that all the selected models perform as well as or better than the base models on the test data. Also, CNNs work better than MLPs, which is not surprising since we are working on image data.

3 Task 2

The second task focuses on the performing difference between MLPs and CNNs. Both digit and fashion datasets are permuted randomly. The new permutated data was trained using the selected models in Task 1. The accuracy on the test sets are shown in Table 5. It is shown that the decrease of accuracy is bigger for CNNs than MLPs. This is consistent with our expectation. In fact, the test accuracy even increase a little bit for permuted data using MLPs.

	Digit data	Fashion data

MLP	Dense(1024, activation='relu',)
	BatchNormalization()
	Dropout(0.1)
	Dense(524, activation='relu')
	Dense(10, activation="softmax")
		Dense(1024, activation='relu')
		BatchNormalization()
		Dense(524, activation='relu')
		Dropout(0.1)
		Dense(524, activation='relu')
		Dense(10, activation="softmax")

CNN	Conv2D(32, (3, 3),activation='relu')
	Conv2D(64, (3, 3), activation='relu')
	Conv2D(64, (3, 3), activation='relu')
	BatchNormalization()
	Dropout(0.1)
	Conv2D(64, (3, 3), activation='relu')
	MaxPooling2D(pool_size=(2, 2))
	BatchNormalization()
	Flatten()
	Dense(128, activation= 'relu')
	Dense(10, activation='softmax')
		Conv2D(64, (3, 3),activation='relu')
		BatchNormalization()
		Conv2D(64, (3, 3),activation='relu')
		BatchNormalization()
		Dropout(0.1)
		Conv2D(64, (3, 3),activation='relu')
		BatchNormalization()
		Conv2D(32, (3, 3), activation='relu')
		MaxPooling2D(pool_size=(2, 2))
		BatchNormalization()
		Dropout(0.1)
		Flatten()
		Dense(128, activation= 'relu')
		Dropout(0.1)
		Dense(10, activation='softmax')

Table 3: The layers of MLPs and CNNs for both digit and fashion data

		Base Models	Selected Models
Digit Data	MLP	0.980	0.981
	CNN	0.990	0.990
Fashion Data	MLP	0.869	0.877
	CNN	0.918	0.921

Table 4: Accuracy of MLPs and CNNs on test data for base models and selected models

In theory, convolutional layers in CNNs perform like feature detectors, such as vertical edges detectors. This is extremely useful for the image data since such features may work for different parts of the images. Therefore, an object is the same to the CNN models no matter where it appears. Permuting pixels ruins the patterns for such features and therefore hurts the performance for CNNs. Such influence will not occur for MLPs since permuting pixel functions like exchanging variables in the regression model, i.e. changing the order the variables will not affect the model performance.

4 Task 3

For task 3 we were going to implement a model that could read the time from pictures of analog clocks. 18000 150x150 grayscale images were provided as train and test data. The data was split into a train and test set, where 20% of the data was part of the test set and 80% of the data was part of the training set. Next, the training set was split up further in order to create a validation set that consisted of 10% of the examples in the training set.

We had some difficulties initially and ended up approaching certain aspects of the task differently. This will become clearer in the final subsections.

4.1 Base models

The problem can be viewed as either a regression problem or a multi-classification problem. The base model was a multi-class classifcation model based on a model used to classify images from the CIFAR-10 dataset, which consists of small images (*Train a simple deep CNN on the CIFAR10 small images dataset.* 2020). The model is based on convolutional layers, maxpooling and dropout.

The output consists of two parts with the first part for the hours and the other part for the minutes. Two versions of this model were created:

- Classification model: the output is integers between 0 and 12 for hours, and integers between 0 and 59 for the minutes;

		Original Data	Permutated Data	Difference
Digit Data	MLP	0.981	0.982	-0.001
	CMM	0.990	0.974	0.016
Fashion Data	MLP	0.877	0.889	-0.012
	CNN	0.921	0.890	0.031

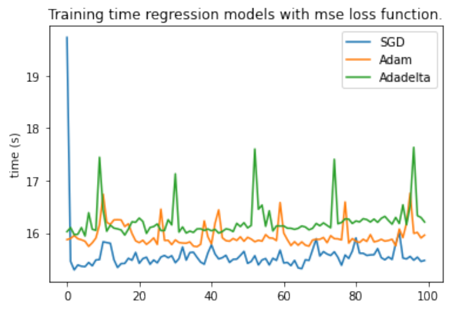
Table 5: The accuracy on the test sets for both original and permutated data

- Regression model: the output values are not integers but decimals between 0 and 1.

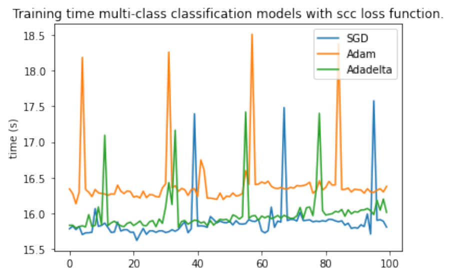
For both versions, different optimizer was tested: `SGD`, `Adam` and `Adadelta`. (*Quick Notes on How to choose Optimizer In Keras* 2020). We selected the optmizer based on the convergence speed and validation accuracy. Based on the experimentation of Task 1, we expected `Adam` would be a good option.

The models were tested with different loss functions: (sparse) categorical crossentropy (scc) for the classification models since we are dealing with a multi-class problem; mean squared error (mse) for the regression models.

The first experiment was a timing experiment in which the first training epoch of all models was run 100 times. Fig. 2 shows that for regression models the adadelta loss function ensures the fastest training, while for multi-class classification models the model with the `adam` function trains fastest. It cannot be concluded that either regression or multi-class classification trains faster than the other, additional (statistical) testing would be needed in order to draw such conclusions, which is not within the scope of this assignment.



(a) Regression models with mse loss function.



(b) Multi-class classification models with ssc loss function.

Figure 2: Training time of first epoch.

Next, the models were trained for 500 epochs and the loss value throughout training is shown in Fig. 3.

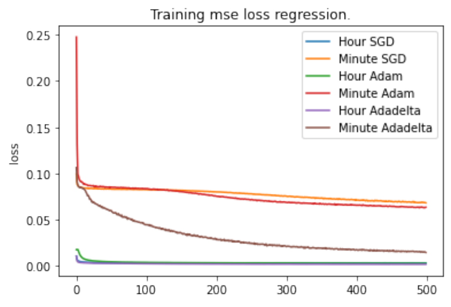
For the regression model it can be seen that there is quick convergence for the hour variable for both the training set and the validation set and also for both the mse loss function and the custom loss function. The minute variable shows slower convergence.,

For the multi-class classification models with the scc loss function the minute loss was not be recorded for unknown reasons. However, the hour loss shows quick convergence for the training set, although there is still a lot of oscillation. For the validation set, the scc loss increases, which is not good.

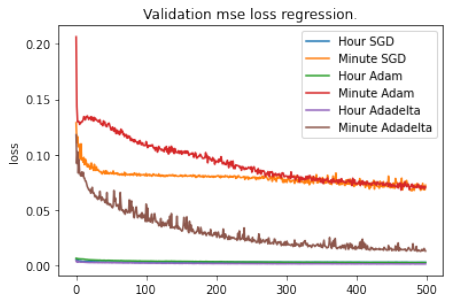
Finally, the models were evaluated on the test set. From the multi-class classification models, the models with scc loss have a loss for hour of 2.485. For the regression models the mse loss for hour is 0.003, 0.003, and 0.002 for `SGD`, `adam`, and `adadelta` respectively; and 0.0738, 0.0725, and 0.0142 (for `SGD`, `adam`, and `adadelta` respectively) for minute. The accuracy was recorded for all models. For the multi-class classification models the accuracy is 0.08 for hour and 0.17 for minute for all three models. For the regression models with mse loss, the accuracy for hour was 0.003, 0.003, and 0.002 (for `SGD`, `adam`, and `adadelta`) and 0.07, 0.07, and 0.01 (respectively for `SGD`, `adam`, and `adadelta`) for minute. It is interesting to see that while the loss is generally not very high, the accuracy is very low, which is consistent with our expectation. Thus, we need to introduce common sense error. Common sense error is the minutes between the actual time and predicted time. For example, the error in classifying 3:59 as 4:01 was two – the same as it would be for classifying 4:01 as 4:03.

4.2 Changing from two outputs to one output

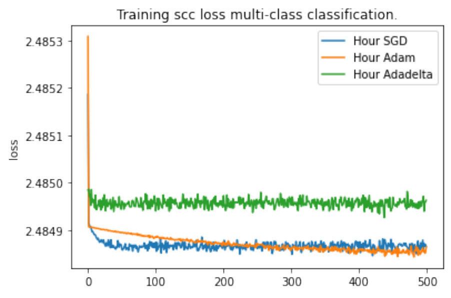
Firstly, the output of the models was redefined. Instead of classifying hour and minute separately, one output layer with 720 nodes was created to classify each possible time in the multi-class classification model. For the regression model, the output would still fall between 0 and 1 but this model also had only one output channel, and not two separate output channels for both hour and minute. Initial experimentation showed that the loss values for multi-class classification remained the same for each of the 20 training epochs. Accuracy on the training



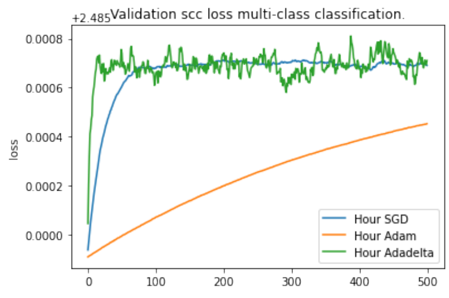
(a) Regression mse loss on training set.



(b) Regression mse loss on validation set.



(c) Multi-class classification scc loss on training set.



(d) Multi-class classification scc loss on validation set.

Figure 3: Loss values.

set remained very low and oscillated, it did not show improvements. The accuracy on the validation set did not change at all and remained 0 at all times. For the regression model, the same problem arose: the loss did not change and the accuracy on the validation set remained 0. Therefore, experimentation with this model was not increased to a bigger scale as the model did not seem promising enough for this regarding time and hardware constraints tied to this assignment.

4.3 Loss function

We tried our own loss function for CNN models, which is based on common sense error and computes loss for hour and minute separately. However, it slowed down the training process and failed to converge. Additionally, it was very hard to debug because they work on tensors instead of simple arrays we can print the values of, and we were unsuccessful in combining both hour and minute predictions into one anyway.

We also applied different weights to the loss of the hour and minutes. To be specific, the hour output is weighted more heavily than that of the minute output for two reasons: the minute is less important compared to hour and there are five times as many minute categories as hour categories. This approves successful during our experimentation.

4.4 Image pre-processing

The background of all the clocks is bright, whereas the hands and digits are dark. This darkness is not the same everywhere, because the lighting of images differs. We can increase the contrast, but we only have to make these dark elements that are of interest more different from the brighter elements. This can be achieved very quickly by just “subtracting” brightness from the already somewhat dark pixels, and subtracting even more from the already quite dark pixels (since we are more confident they correspond to actually dark elements, i.e. *not* the white background of the clock). One example of changing the contrast is shown in Figure 4. The clock on the left is the original one and the one on the right is updated one with increased contrast.

Another kind of pre-processing that we do not implement is increasing the size of our training set by creating slightly distorted and, for example, rotated variations of the images. We have the necessary code, but can not run it on all of our data. This problem can be alleviated imperfectly by casting the data to a smaller data type – a much bigger dataset simply does not fit in the RAM of the environment we used.

4.5 Final model

After testing out different set up of layers and activation functions, the CNN model we eventually obtained with the best results with has a number of parameters in the order of 600 000. It consists of several convolutional layers that are all followed by max pooling, then a fully connected layer, then a regular dense layer, and finally a differentiation into an hour and minute branch. These branches are very similar to each other; for both the hour and minute branch they consist of one relatively small or relatively large dense layer (respectively) and a 12 or 60

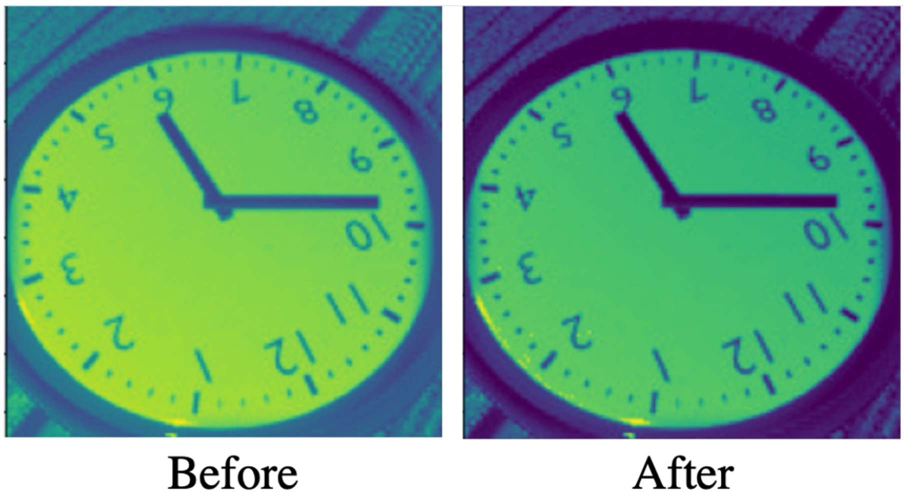


Figure 4: Example of the clock figure before and after increase the contrast

sized final dense layer (idem). The activation functions are always rectified linear, except for this final layer where the softmax is used. A dropout of differing magnitude occurs after each layer. The structure is shown in Figure **??**

As mentioned in Tasks 1 and 2, dropout, convolution and maxpooling play an important role in our model adjustment:

- Dropout: it is significant since our model is deep. It improves generalization and the validation accuracy on the validation data.
- Convolution: acting as feature detectors, convolutions layers help decreases the parameter numbers and increase the prediction accuracy.
- Pooling: it reduces the size of the array with two hyperparameters – f for size and s for stride. We used maxpooling as generally recommended. It looks at each f times f square of the array with a step of s strides, and keep the biggest number, which is ideally the most interesting bit.

Categorical cross entropy is used as loss function with weights of 6 and 4 for hour and minute. The `adam` optimizer with a relatively small learning rate (around 0.0003) is used. The batch size is kept small (around 24) but the number of epochs is relatively high – the environment we used does not let use go over 300 but it may be beneficial to try this in the future.

Unlike what was mentioned for the base model, the validation split ratio is adjusted downward from 0.10 to 0.05, so we train over only 75% of the data we have been provided with. Validating over just 900 images seems sufficient, especially as we really are interested in the common sense loss that could not be monitored during training.

Perhaps due to all our dropouts, cross entropy loss and accuracy on our validation sets is consistently higher than on our training sets (during training). This is definitely a good property of our setup, and indicates that creating the variations of the images for a bigger training set may not be especially beneficial.

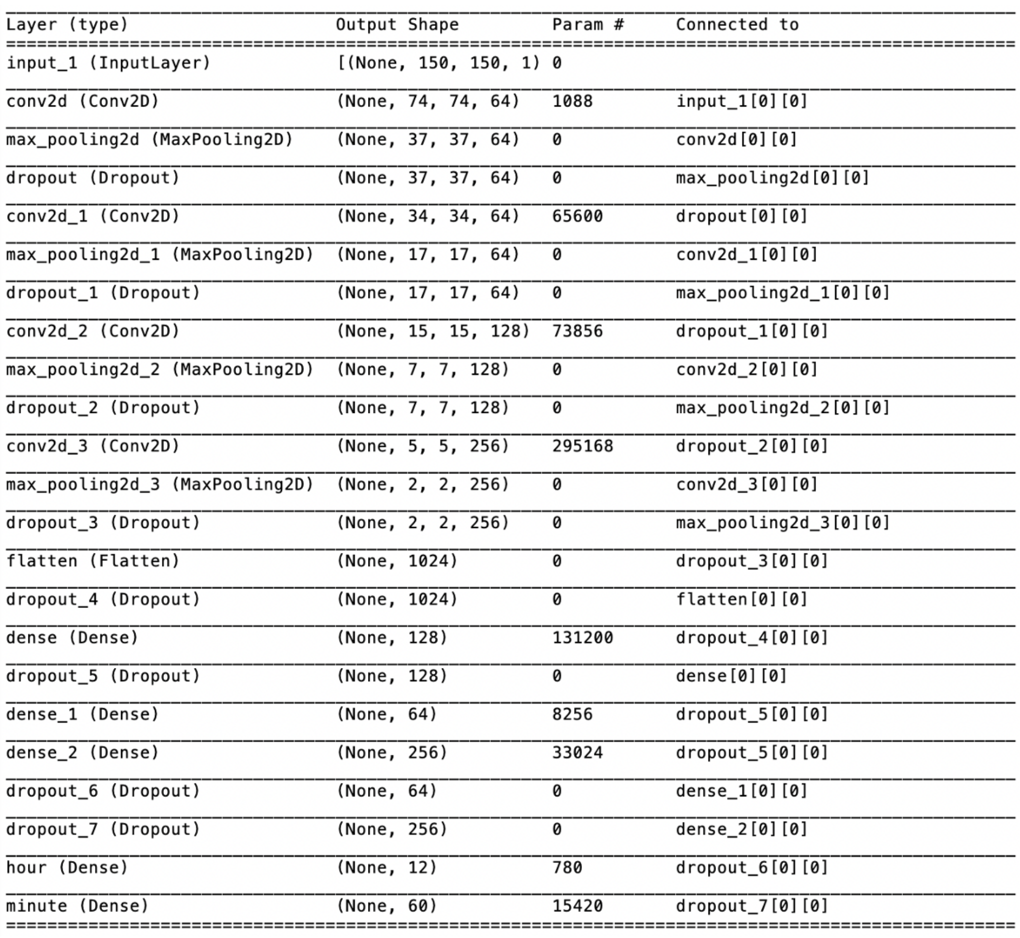
The results of the loss and accuracy on training and validation data is shown in Figure **??**. The accuracy jumps around 20 epochs.

We tested the model on the test data. A random sample of 10 predictions on the test set is shown in Table 6.Note that the first prediction in this list has a common sense loss of 60 minutes. This is is not the type of error a human is likely to make. The hour hand is indeed very close to 1, but the minute hand is at 59, and the latter actually gives a lot of information about the hour hand. This "fact" is presumably represented in the network in some manner, but could be made more explicit by perhaps merging the branches for a single final layer.

The common sense loss – i.e., the mean absolute number of minutes wrong on the test – we obtain is a little under two minutes. This seems like a good result, since the model is trained on quite limited hardware and the times displayed in these images are quite hard to read more precisely for humans as well.

5 Conclusion

For this assignment, we experimented with MLPs and CNNs using Keras and Tensorflow.



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 150, 150, 1) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 74, 74, 64)   1088        input_1[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 37, 37, 64)   0           conv2d[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 37, 37, 64)   0           max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 34, 34, 64)   65600       dropout[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 17, 17, 64)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 17, 17, 64)   0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 15, 15, 128)  73856       dropout_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 7, 7, 128)    0           conv2d_2[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 7, 7, 128)    0           max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 5, 5, 256)    295168      dropout_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 2, 2, 256)    0           conv2d_3[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 2, 2, 256)    0           max_pooling2d_3[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1024)         0           dropout_3[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 1024)         0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          131200      dropout_4[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 128)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           8256        dropout_5[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          33024       dropout_5[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 64)           0           dense_1[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 256)          0           dense_2[0][0]
__________________________________________________________________________________________________
hour (Dense)                    (None, 12)           780         dropout_6[0][0]
__________________________________________________________________________________________________
minute (Dense)                  (None, 60)           15420       dropout_7[0][0]
==================================================================================================
```

Figure 5: The structure of the selected final CNN model

Firstly and most importantly, we found that CNNs work much better than MLPs for our image data. Convolutions layers definitely take advantage of the characteristics of images and function well as the feature detector for any part of the image. It saves a number of parameters and improves the prediction accuracy.

For simple datasets such as MNIST and Fashion MNIST, a simple model works. There is no need for many layers and regularization terms. Also, different activation functions and optimizers do not have a significant influence on our simple dataset, except for the linear activation function. This is also expected since the linear activation function does not take the advantage of Neural Networks that theoretically can model any non-linear relationships.

For a complicated problem like Task 3, the optimizer plays an important role in finding the optimal solution of minimizing the loss function. In our case `Adam` perform better since it treats parameters differently and uses momentum for smooth diverging. Adding more layers in the model absolutely helps to improve the results for Task 3. However, dropout is also found to play a significant role in this case. It helps combat the issue of overfitting and gives us a high validation accuracy.

The fact that 0 and 59 and 0 and 11 are in reality close together on a clock provides a challenge as it is hard to convey this information to the model so that it can be exploited during the training phase. Therefore it is tricky to use accuracy for Task 3. For example, one minute difference between the actual time and predicted time will not be a big issue in common sense but affect the accuracy equally as any other difference.

The model we derived for Task 3 achieved a common sense error of 1.89 minutes on the test data. We believe with data augment (image rotation, shift and/or mirroring) and higher resolution will further improve the performance. Sadly, this is hardly done due to the hardware limit. With the increasing data set and more advanced Neural Network models in the near future, computation power will in great need.

References

Fashion MNIST (2020). URL: `https://www.kaggle.com/zalando-research/fashionmnist` (visited on 04/20/2020).

MNIST (2020). URL: `https://www.tensorflow.org/datasets/catalog/mnist` (visited on 04/20/2020).

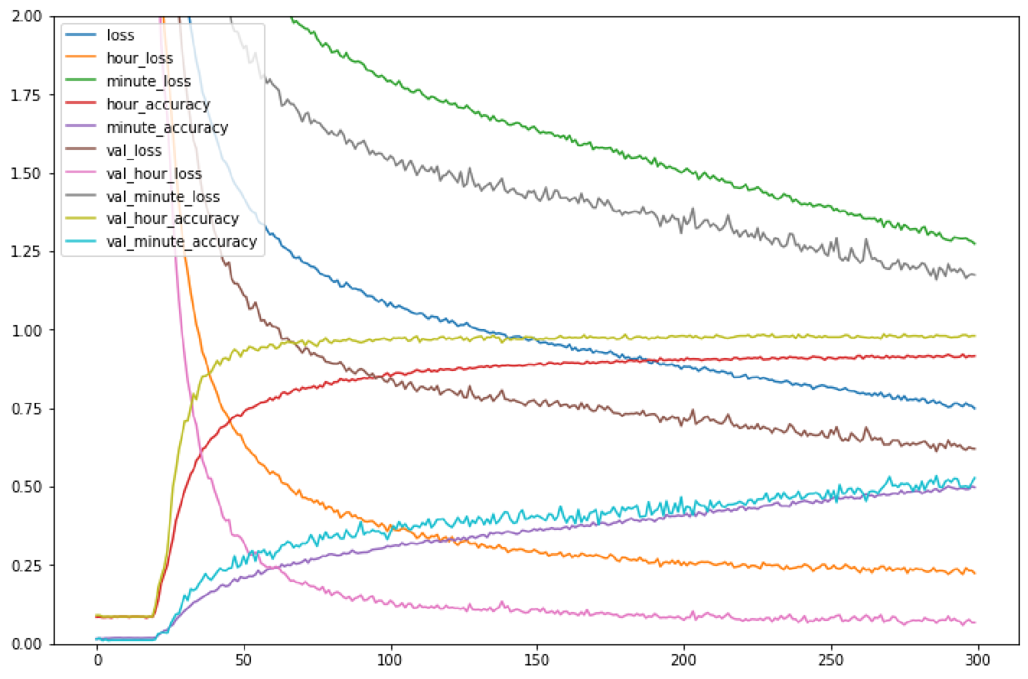


Figure 6: The loss and accuracy on training and validation data of the selected final CNN model

Predicted		TRUE	
Hour	Minute	Hour	Minute
(1	59)	(0	59)
(11	39)	(11	37)
(6	19)	(6	19)
(2	50)	(2	49)
(3	26)	(3	29)
(3	10)	(3	11)
(8	1)	(8	0)
(0	25)	(0	25)
(11	23)	(11	23)
(9	22)	(9	24)

Table 6: A random sample of 10 predictions on the set

Quick Notes on How to choose Optimizer In Keras (2020). URL: `https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/` (visited on 04/20/2020).
Train a simple deep CNN on the CIFAR10 small images dataset. (2020). URL: `https://keras.io/examples/cifar10_cnn/` (visited on 04/20/2020).