Neural Networks 2019 - 2020 Assignment - 2

Deadline April 26th 2020

1 Introduction

Recent technological advances within the computer industry has made computation power readily available and motivated the scientific community to experiment more and more with computationally expensive models, which are able to "self" learn functions from the data itself. Under conditions of a diverse and large data, such models are able to extract abstract features and outperform humans on the same actions, e.g. image recognition. This paper examines the performance of different neural networks used for image recognition and classification problems within the field of computer vision, which can be regarded as a collection of such models. Specific focus will be on Multi-layer perceptrons (MLPs) and Convolutional Neural Networks (CNNs) applied on the MNIST, fashion MNIST and a clock image data-set.

First, we compare the results of different MLPs and CNNs on different architectures, parameters and settings in section 2. The key objectives of these experiments is to develop a basic understanding of Keras and TensorFlow, two popular open source libraries for developing machine learning models and neural networks. Subsequently, the impact of image obfuscation is examined in section 3, where the most successful setups of the MLP and CNN networks from section 2 are applied to permuted versions of both data-sets. Performances of both networks are compared on the permuted versions of the data-sets. Finally, in section 4, we inspect the performance of a CNN on a more complex operation, using a clock image data-set in which the CNN has to learn to tell the time as accurately as possible. As such, we examined the following topics in chronological order: Architectural design of MLPs and CNNs, the impact of image obfuscation on MLPs and the potential of CNNs on a more complex "tell the time" image recondition task.

2 Multi-layer Perceptrons and Convolutional Neural Networks

The purpose of this section is to develop a general idea about the architectural elements of MLPs and CNNs by experimenting with the network architecture. When designing a neural network a number of parameters such as 1) the number of hidden layers, 2) number of neurons per hidden layer, 3) regularization, 4) learning rate, 5) batch-size, 6) optimizer, 7) activation function have to be considered. In this part of the assignment, different adjustments to the architectural choices mentioned above are made. Special focus is given on determining the most successful MLP and CNN setup for the MNIST and Fashion MNIST data-set. Therefore, in this section, we experiment with a Multi-Layer Perceptron (MLP) and a Convlutional Neural Network (CNN) to classify digits and clothing articles of the MNIST and fashion MNIST dataset respectively.

2.1 Multi Layer Perceptron

The MLP experiments utilize two different Network Architectures , with two different methods of regularization and three different activation functions to achieve the best accuracy on the MNIST and Fashion MNIST data set. With respect to the number of hidden layers, the performance of models with two hidden layers and one hidden layer are compared. The number of neurons per hidden layer follow the practice of forming them in a pyramid like fashion with fewer and fewer neurons at each layer. A simple representation of the architecture used can be examined in Table 1. The network takes as input an image represented as a 784 (28×28) dimensional-vector of pixels, representing normalized values of gray-scale intensity, that is, values in the interval $[0, 1]$. The output layer contains 10 nodes, one for each of the 10 classes and utilises a Softmax activation function. A Softmax activation function converts the output of the network into probabilities by squashing the parameter space using a sigmoid function. As a result, the output values also sum up to 1 across the 10 classes. Given a vector of input values $\boldsymbol{x}$, the MLP will output a vector $\boldsymbol{y}$ with 10 probability values that sum up to one. The index $i \in \{0, 1, 2, ..., 9\}$ of the output vector $\boldsymbol{y}$ represents each classes category. As such the value with the highest probability within the output vector $\boldsymbol{y}$ then corresponds to the index i, which is the class the input will be classified to. Therefore, the index i that corresponds to the highest probability within the output vector $\boldsymbol{y}$ is the predicted class for input $\boldsymbol{x}$. Batch-size was set to 16 since the optimal batch-size is smaller than 32. A smaller batch-size results in greater computational efficiency during training iterations and because a smaller batch-size has been shown to be closer to the optimal batch-size. The benefit of more accurate gradient estimation when using larger batch-sizes (> 32) are negligible, since more accurate gradients do not guarantee to point towards the direction of the optimum due to the complexity of the optimisation landscape. As such a batch-size of 16, is sufficient. The Network Architectures are as follows:

Table 1: Comparison of Model Architectures

Layer type	Nodes
Dense	300
Dense	100
Output	10

a) Architecture 1

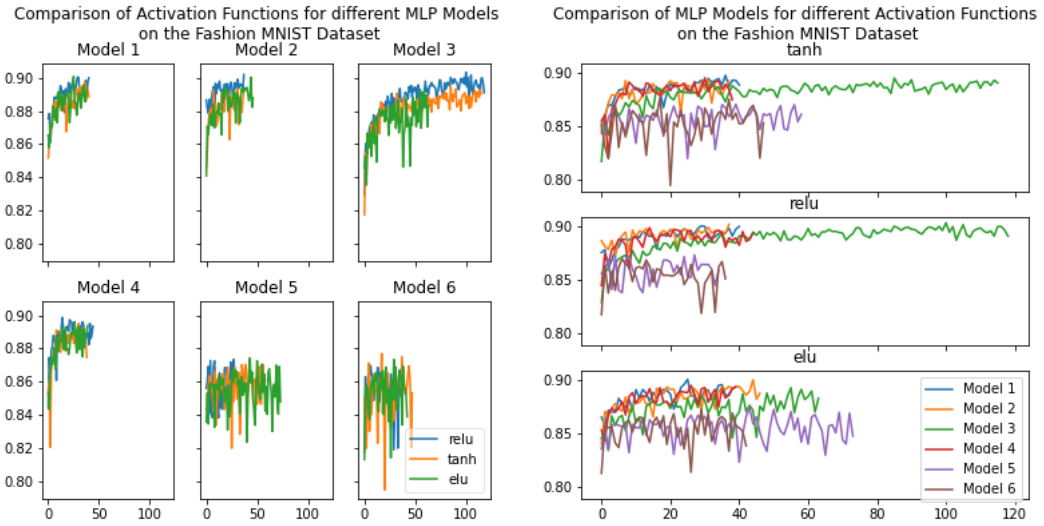
Layer type	Nodes
Dense	300
Output	10

b) Architecture 2

Three forms of regularization are applied to both models architectures: (1) No regularization; (2) Dropout: For the net with two hidden layers the dropout was between the layers with $p = 0.5$. For the net with one hidden layer it was before the output layer with $p = 0.2$. (3) l_2 regularization: $\lambda = 0.001$ on all hidden layers.

As for the choice of activation function, three different activation functions are used: tanh, relu and elu with a he, glorot and he initialization respectively. Theory tells us that elu should be superior to relu, which in turn should be superior over tanh [1]. Our comparison can therefore shed practical insight on theory. For all models early stopping is used, meaning that as soon as the validation accuracy does not improve significantly for a certain number of epochs, the algorithm is treated as if it converged. The number of epochs relating to the early stopping mechanism were set to 30 epochs. A total number of $2 \times 3 \times 3 = 18$ networks are therefore trained on both data-sets. In our graphs, Models 1, 3 and 5 use Architecture 1 and Models 2, 4 and 6 use Architecture 2. Models 1 and 2 use no regularization. Model 3 and 4 use Dropout in the hidden layers with a probability of 0.5. This means a node has a probability of $p = 50\%$ to be temporarily dropped out and does not contribute to the forward backward pass of information within the network. As such, dropout avoids over-fitting the model, because the network samples a new architecture each time a new input vector is presented. Models 5 and 6 use $l2$-regularization, with a

regularization factor of 0.001. The results can be seen in Figure 1 and 2.



(a) (b)

Figure 1: Comparison of Validation Accuracies while Fitting for Different Activation Functions and Model Accuracies.

In Figure 1 (a) we compare the convergence speed of different activation functions for different architectures on the Fashion MNIST dataset. While elu and relu are generally considered superior to other models it is hard to pick a single winner from these graphs. A general pattern seems to be that elu converges relatively quickly and that relu achieves a high accuracy relatively quickly. Furthermore we can see that $l2$-regularization, which prevents single weights of getting too high, can lead to non-convergence. Experimentation with lower regularization factors than the used 0.001 might have shown better results but were outside the scope of this experiment.

In Figure 1 (b) we compare the different architectures for different activation functions. Here we see that the models with too much regularization fail to achieve the same accuracy than models with less. Furthermore we see that Model 3, which uses dropout, continues to show small improvements for a relatively long time. The choice of the number of hidden layers (1 or 2) seem to play a relatively minor role as no architecture clearly dominates the other.

In Figure 2 we show the results of all models on both data-sets. We will first examine results for the MLP then for the CNN. We can see that the results on both the Fashion as well as the Digits

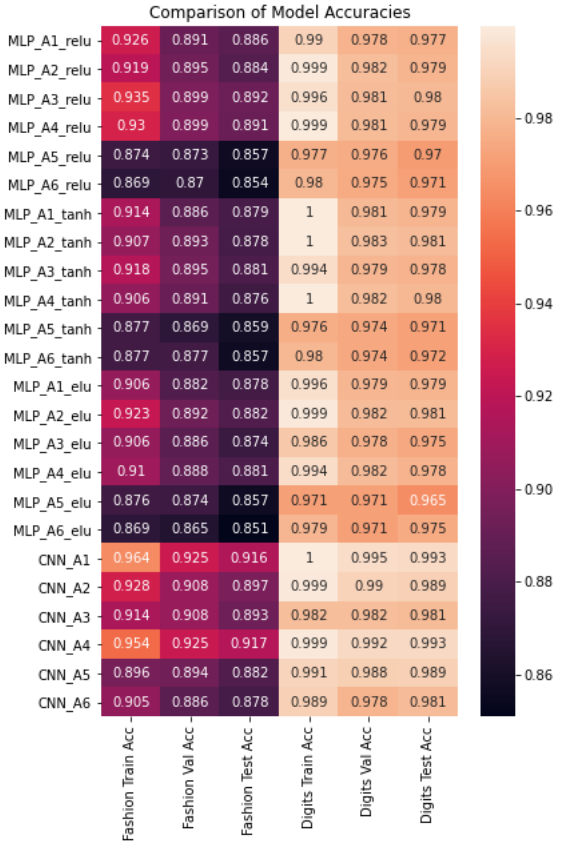


Figure 2: Heatmap of Model Accuracies of CNNs and MLPs.

data-set are relatively close together, with test accuracies in the range $[0.851, 0.892]$ and $[0.965, 0.981]$ respectively. Because of fluctuations in the accuracies, we are careful about strong claims of superiority. However, patterns emerge nevertheless. For one, Architecture 1 (2 hidden layers) almost always outperforms Architecture 2 for all forms of regularization and all activation functions in the Fashion data-set, but vice versa in the digits data-set. Deeper neural networks can learn more abstract features. An ability, which proves useful in the Fashion MNIST but seems to be unnecessary for the Digits data-set. Furthermore, relu seems to be better than elu, which seems to be better than tanh. However, tanh does very well on the Digits data-set. Our best performing MLP is therefore MLP_A3_relu, which is Architecture 1, with dropout and relu activation. This Model is chosen as it achieves competitive scores on both data-sets. However, note that, as one can see in Figure 1, it took significantly more epochs to converge than other models. This should be kept in mind when computational considerations play a bigger role.

2.2 Convolutional Neural Networks

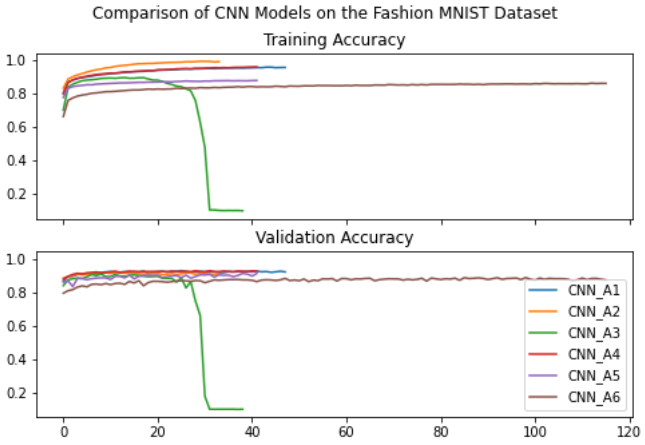


Figure 3: Comparison of training for different CNN Architectures

Next, we used Convolutional Neural Networks (CNNs) to classify digits and fashion articles of the MNIST and Fashion MNIST data-set. The values of each image are normalized in the interval [0, 1] as before. The input are n single images of 28×28 pixels. In other words, they are not flatten as we did in our MLP-models. We tried six different Architectures, which can be seen in Table 7. The hyper parameters we experimented with were Number of Filters (F) and kernel-size (K) for the Convolutional Layers, where K is the number of pixels on one side of a quadratic kernel and number of nodes of a dense layer (N). Furthermore, we experimented with different configurations of stacking convolutional, dense and pooling layers. Lastly, we experimented with Dropout and $l2$-regularization. A comparison of training processes can be seen in Figure 4.

When looking at the training processes in Figure 4 the first model one notices is Model 3, whose accuracy suddenly drops after about 25 epochs. While the exact reason for this is unclear, the model is likely to be too complex or that it is caused by the Dropout in the last layers. Furthermore, we see that model 1 (blue) and 4 (red) are very close together. The main difference between the two models is the kernel sized used, as we suspected 7 to be rather high for images that small. However, kernel size did not seem to affect accuracies too much. While the relatively small model 2 achieves higher training accuracy than the other models, its validation accuracy is comparable. This is because of the lack of regularization. We observe the opposite phenomenon for model 5 and 6. They lack behind during training but achieve competitive validation accuracies. This is because they use $l2$ regularization, with a regularization factor of 0.001.

In Figure 2 we see the final results of the models. Again we see that the results are relatively close together. The best performing models are 1 and 4, where Model 4 slightly

outperforms Model 1 on the Fashion dataset. We therefore choose the smaller Kernel size as superior and Model 4 as our winner.

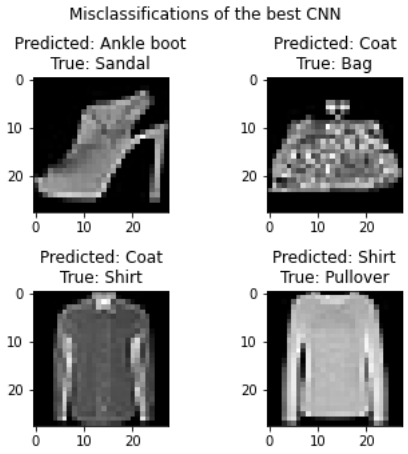


Figure 4: Comparison of training for different CNN Architectures

Lastly, we should compare our MLPs to our CNNs. In the heatmap of model accuracies (Figure 3) we can see that our top CNNs outperformed our top MLPs. A test accuracy of 99.3% on our Digits Test set is a remarkable achievement. Similarly, a test accuracy of 91.7% on our fashion data is quite impressive. Our MLPs are not able to achieve such a performance on either dataset. However, we should expect the benefits of CNNs to be much bigger on more complex images and images with higher resolutions, as they can learn more complex structures with less weights.

3 Impact of data obfuscation using fixed random permutation

In this section we implemented a simple permutation function, in which each input image of the MNIST and fashion MNIST data-set follows the same random permutation of pixels. That is, each input image of 784 pixels (28×28 matrix) is randomly shuffled, using one randomly generated index of size 28×28 for all images. An example of the resulting permutation can be seen in Figure 5.

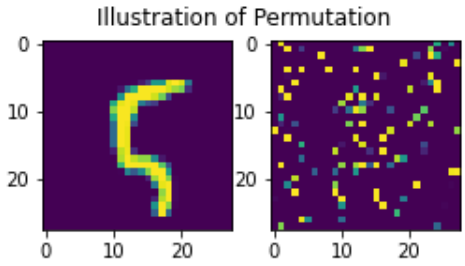


Figure 5: Example of a Permutation on the MNIST data.

Using the most successful MLP and CNN setups from section 2, the networks are trained on the permuted data-sets and performance is compared. As previously mentioned, the third specified MLP model "MLP Arch 3 Relu" (see Figure 2) in Section 2, which resulted in an accuracy of 0.98 and 0.892 for the MNIST and fashion MNIST data-set respectively, was chosen as the most successful MLP. Results of this MLP on the permuted data-set can be examined in Table 2. Visibly, on the permuted data this MLP resulted in accuracy of 0.98 and 0.88 for the MNIST and fashion MNIST data. Notable, there is a difference of 0.012 between the accuracy of the original and permuted fashion data, which however is negligible. Based on the results, the MLP is invariant to random permutation of pixels and order of the input vector does not matter. This is due to the fact that, a fixed random permutation can be regarded as a change of coordinates in the 784 dimensional space, in which permuted images of the same class category will result in similar images, i.e. permuted digits representing a 5 will have a similar dispersion of grey-scaled pixels on the 28×28 dimensional space.

As for the CNN, the fourth specified CNN model "CNN Arch 4" (see Figure 2) was chosen as the most successful CNN. The CNN resulted in an accuracy of 0.993 and 0.917 on the original MNIST and fashion MNIST data-set. Results the permuted data-set can be examined in Table 2 as well. On the permuted data-set we observe that CNN reaches an accuracy of 0.97 on the MNIST data and an accuracy of 0.88 on the fashion data. There is a drop in accuracy of 0.023 and 0.037 of both the MNIST and fashion MNIST on the permuted image data.

Overall, we observe that our MLPs achieved comparable results on the permuted and

	MLP on Fashion	CNN on Fashion	MLP on Digits	CNN on Digits
Train	0.92	0.93	0.99	0.99
Val	0.89	0.89	0.98	0.98
Test	0.88	0.88	0.98	0.97

Table 2: Comparison of Models on Permuted Data

unpermuted data, while the performance of our CNNs dropped to the performance of our MLPs. Since our CNN has more layers than our MLPs, it is still able to learn a similar function. However, the convolutional layers become useless as proximity between pixels becomes meaningless after the permutation, as it is not indicative of the target class anymore. The problem of this is aggravated when using higher pooling sizes. When using the same Architecture (Model 4 in Table 7) but with a pooling size of 4 we only achieve test accuracies of 87% and 95% on the Fashion MNIST and the Digits MNIST respectively. We thus conclude that our theory is correct, that is MLPs perform almost as well as on the unpermuted Data, while CNNs perform worse. However, CNN performance in our case does not drop below the performance of the MLPs, as our relatively deep CNN is still able to learn the same function.

4 Tell-the-Time Network

In this final section, we combined our previously gained knowledge to develop a neural network that, when applied to images from the test set would tell the time as correctly as possible. The data-set consists of 18000 images (150×150 pixel) of an analogue clock and a numeric representation of the time shown on these images. A random collection of these images is shown in Figure 6.

The goal is to reduce the absolute value of the difference between actual and predicted time. This problem can be formulated either as a multi-class classification problem with e.g. 12×60 classes or as a regression problem with e.g. predicting the number of minutes after 12:00 (or, equivalently, 0:00). Initially, we decided to tackle the problem as a regression task. In order to predict the number of minutes after 12:00 we had to transform the representation of the actual time into the total number of minutes after 12:00 by multiplying the hours with 60 and then adding the minutes. The resulting representation of the actual time is in the range $[0, 720]$. If we want to minimize the absolute value of the difference between actual and predicted time, we have to take into account that the absolute difference between e.g. 11:55 and 0:05 is just 10 minutes and not 11 hours and 50 minutes (or between 715 and 5 the absolute value of the difference is just 10 and not 710 on the transformed scale). Therefore, we created a customized loss function as follows:

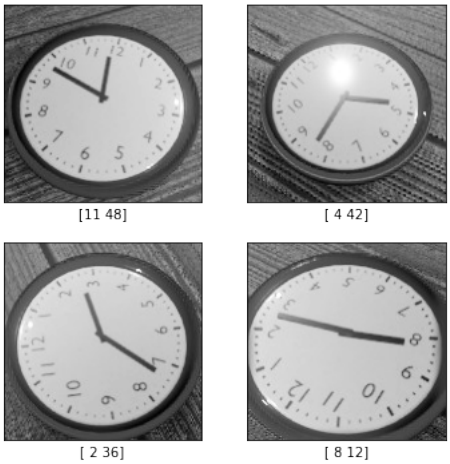


Figure 6: Random collection of images and the numerical representation of the actual time from the clock data-set

```
# Custom Loss Function
def custom_mae(y_true, y_pred):
    y_pred = tf.math.floormod(y_pred, 720)
```

```
4       abs_diff = tf.math.abs(y_true - y_pred)
5       min_diff = tf.minimum(720 - abs_diff, abs_diff)
6       return tf.reduce_mean(min_diff)
```

Line 3 ensures that predictions above 720 are reduced to the remainder of the division by 720. Line 4 takes the absolute difference of predicted and actual time. Line 5 solves the above mentioned problem by selecting always the absolute value of the difference that is smaller than 360.

We started with a relatively simple CNN architecture which is described in Table 3. In the beginning, the network basically consists of three sets of two stacked convolutional layers followed by a max pooling and batch normalization layer. Then, another convolutional layer is added on top followed by three dense layers including the output layer. The customized loss function was used to minimize the loss of the network. Unfortunately, the results showed that the loss was not decreasing during training after all. We concluded that the customized loss function is too complicated in order to derive gradients properly. Because of the modulo-operator used in the loss function, the loss landscape has many local minima and does not converge. We decided to go ahead using the ordinary mean absolute error (MAE) as loss function instead and the customized loss function as a metric of evaluation. This is not an optimal solution, because using the MAE instead of the customized loss not only increases the range of the value of the loss function from $[0, 360]$ to $[0, 720]$, its gradients also encourage to decrease predictions when they should be increased instead (if MAE > 360). Nevertheless, we preferred the ordinary MAE compared to the mean square error loss, because the MAE is more robust to bigger differences because it does not make use of the square.

Layer type	Specifications
Convolutional	F:64; K:7; S=3
Convolutional	F:64; K:7; S=3
MaxPooling	PS=2
BatchNormalization	
Convolutional	F:128; K:5
Convolutional	F:128; K:5
MaxPooling	PS=2
BatchNormalization	
Convolutional	F:256; K:3
Convolutional	F:256; K:3
MaxPooling	PS=2
BatchNormalization	
Convolutional	F:512; K:3
Flatten	
Dense	300
Dense	100
Output:Dense	1

Table 3: Architecture of the regression network

However, using the same network architecture as before, this time applying the ordinary MAE loss function and evaluating with the customized MAE leads to the results shown in Table 4. By using the ordinary MAE loss, the network is able to predict the time with an accuracy of around 15 minutes on the test set.

	Customized MAE
Training	5.1064
Validation	14.3453
Test	15.2369

Table 4: Evaluation of the regression network on train, validation and test set.

In order to get a better overview of the network's performance we plotted the residuals against the time. The results can be seen in Figure 7. Apparently, the network's performance seems to be considerably worse in the boundary regions, compared to the slight variation of the performance in between. A reason for this behaviour is probably related to the above mentioned problem of the common sense accuracy. It follows, that the biggest differences in actual and predicted time occur if one of the values is slightly before 12 o'clock and the other is slightly after 12 o'clock.

Since we were not able to implement a loss function that is able to correct for this problem, we decided to implement another approach in which we divide the problem of telling the time into two separate tasks by using a combined

approach of classification and regression techniques. We developed a neural network with two output values for the hours and minutes respectively, where predicting the hour is considered a classification task with 12 classes and predicting the minutes is considered a regression task, thereby reducing the size of the regression problem substantially to the range $[0, 59]$.

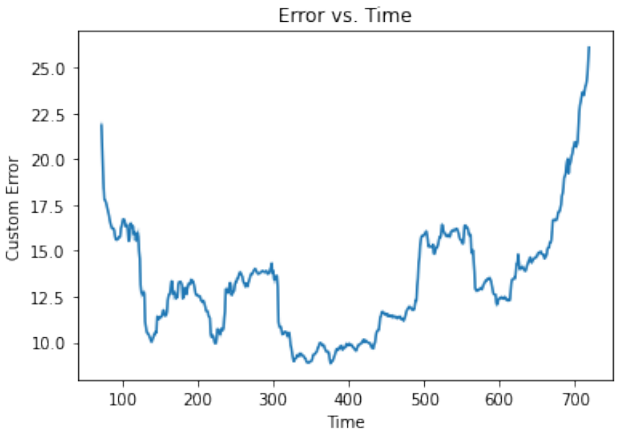


Figure 7: Plot of smoothed error against time from the regression network.

The network's architecture consists of 3 convolutional layers in the beginning, each followed by a max-pooling and batch normalization layer. It follows another convolutional layer before the network splits into two branches of fully-connected layers for the hours and minutes respectively. An overview of the architecture can be seen in Table 5. Since predicting the hour is a classification task we applied the softmax activation on top of the corresponding output nodes, whereas predicting the minutes is a regression task and thus a linear activation is applied at the corresponding output node.

We used the sparse categorical cross entropy loss for the classification task and the mean squared error loss for the regression task and evaluated the model using the accuracy and the customized MAE for hours and minutes respectively. Note that we used the mean squared error loss instead of the MAE loss now, because in this reduced problem the heavy penalization of big differences is actually beneficial. After the training process, we evaluated the model on all three parts of the dataset. The results are shown in Table 6. The combined approach predicts the hour with an accuracy of approximately 98% on the test set. Note that we scaled the minutes to the range $[0, 1]$ prior to training in order to provide better performance of the network. Consequently, the minute MAE denotes the mean absolute difference on the transformed minute scale. The third column represents the values of the common sense accuracy. We used the predictions for the hours and minutes and transformed them together with the actual values back to the total minutes scale and then applied our customized MAE function again. The combined approach is able to predict the time

Layer type	Specifications
Convolution	F:64; K:5; S=2
MaxPooling	PS=2
BatchNormalization	
Convolution	F:128; K:3
MaxPooling	PS=2
BatchNormalization	
Convolutional	F:256; K:3
MaxPooling	PS=2
BatchNormalization	
Convolutional	F:512: K:3
Flatten	
Hour:Dense	N:144
Minute:Dense	N:200
Hour:Dense	N:144
Minute:Dense	N:100
Output:Hour:Dense	N:12
Output:Minute:Dense	N:1

Table 5: Architecture of the combined model.

with an accuracy of 3.131 minutes. This is a remarkable improvement compared to the previous regression only approach. In fact, the combined approach is able to predict the time almost five times more accurately.

	Hour Accuracy	Minute MAE	Custom MAE
Training	1.0	0.0186	1.2488
Validation	0.9829	0.0413	3.1455
Test	0.9783	0.0414	3.1311

Table 6: Evaluation of the combined model on train, validation and test set.

Again, we plotted the residuals against the time to get more insights into the network's performance in Figure 8. Obviously, the range of the error is much smaller compared to the regression only approach. Furthermore, the we longer observe higher errors in the boundary regions. Splitting the task seems to solve the problem of differences between values before and after and 12 o'clock. What we now observe are the peaks occurring around every full hour. A possible reason for this behaviour might be wrong predictions of the hours. Imagine the actual time is 8:58, then the hand on the clock is so close to 9 already that the network's prediction could be 9 instead of 8, while the predictions for the minutes are still close to 58. The loss value that is minimized by the network is the sum of the individual losses. During training, we observed that the hour loss most of the time dominates this sum. It follows that wrong predictions for the hour lead to the bigger differences right before every full hour. A further step would be to introduce different weights for the individual losses in order see if this improves the network's performance even further. However, the problem of the common sense accuracy for the minute loss still exists but on a much smaller scale compared to the regression only approach. Another step would thus be to introduce an appropriate loss function which is able to correct for this problem. However, for the scope of this research we are more than content with the achieved results.

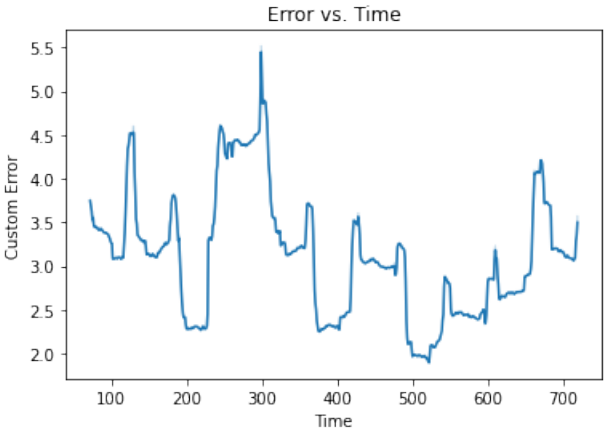


Figure 8: Plot of smoothed error against time from the combined network.

References

[1] Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

Appendix

Layer type	Specifications
Convolution	F: 64; K: 7
MaxPooling	PS = 2
Convolution	F: 256; K: 3
MaxPooling	PS = 2
Flatten	
Dropout	$p = 0.5$
Dense	N: 128
Dropout	$p = 0.2$
Output	N: 10

a) Model 1

Layer type	Specifications
Convolution	F: 32; K: 7
MaxPooling	PS = 2
Flatten	
Dense	N: 128
Output	N: 10

b) Model 2

Layer type	Specifications
Convolution	F: 64; K: 7
MaxPooling	PS = 2
Convolution	F: 128; K: 7
Convolution	F: 128; K: 7
MaxPooling	PS = 2
Convolution	F: 256; K: 7
Convolution	F: 256; K: 7
MaxPooling	PS = 2
Flatten	
Dense	N: 128
Dropout	$p = 0.5$
Dense	N: 64
Dropout	$p = 0.2$
Output	N: 10

c) Model 3

Layer type	Specifications
Convolution	F: 64; K: 3
MaxPooling	PS = 2
Convolution	F: 256; K: 7
MaxPooling	PS = 2
Flatten	
Dropout	$p = 0.5$
Dense	N: 128
Dropout	$p = 0.2$
Output	N: 10

d) Model 4

Layer type	Specifications
Convolution	F: 128; K: 3
MaxPooling	PS = 2
Convolution	F: 256; K: 7
Dropout	$p = 0.5$
MaxPooling	PS = 2
Flatten	
Dropout	$p = 0.5$
Dense	N: 128, $l2$-reg
Dropout	$p = 0.2$
Output	N: 10

e) Model 5

Layer type	Specifications
Convolution	F: 128; K: 3
MaxPooling	PS = 2
Convolution	F: 256; K: 7
Dropout	$p = 0.5$
MaxPooling	PS = 4
Flatten	
Dropout	$p = 0.5$
Dense	N: 128, $l2$-reg
Dropout	$p = 0.2$
Output	N: 10

f) Model 6

Table 7: Comparison of CNN Architectures. F specifies the number of Filters of a convolutional layer and K the kernel size. PS specifies the pooling size of a Pooling Layer. N the number of nodes of a dense layer and p the probability of Dropping on a Dropout Layer.