Deep learning and Neural Networks

Assignment 2

Sjoerd Hermes
Anish Kisoentewari
Thomas Lieber

Universiteit Leiden
Dr. W.J. Kowalczyk
22-04-2020

Contents

1 Learn basics of Keras and TensorFlow

1.1 Architecture

MLP MNIST and Fashion MNIST

The MLP from Geron (2019)[1] on pp. 296 formed the basis for the MLP used for the first assignment on the MNIST data-set. This model consists of 1 `Flatten` layer followed by 3 `Dense` layers, where the first `Dense` layer has 300 neurons, the second 100 neurons, and the final layer has 10 neurons, one for each class. Using the most promising activation, initialization and optimization functions (see below), this model had a test accuracy of about 97%. The first change to the model, in order to improve the accuracy, was in the form of changing the amount of neurons used in each layer, except the last one since the amount of categories remained the same. This did not results in a change in the test accuracy. The following change was increasing the amount of layers: in this step more `Dense` layers got added which also did not result in any change. Inspired by the Keras team[2], the model from Geron was adapted for a model in the style of the `mnist_mlp.py` as found on their Github[2]. This model is arguably more sophisticated and makes use of several `Dropout` layers, alternated by `Dense` layers with a consistent amount of 512 neurons. This model could get to 98% test accuracy. Trying to fine-tune the model both more layers (`Dense` and `Dropout`) were added, and the amount of neurons were changed, but the model could not exceed the threshold of 98% test accuracy. Fine-tuning the dropout rate did seem to have an impact on the model accuracy, but the initial setting proved to be sufficient, as going above a rate of 0.5 decreased the model accuracy.

MLP Fashion MNIST

Aside from different data preparation, input values and amount of neurons in each layer, the MLP model for the Fashion MNIST is identical to the model for the MNIST data set. While trying to adjust the model for the different data set, there was no noticeable improvement in adding more or different layers (which was also rather improbable, given the high accuracy the model had on the MNIST data set). This might very well be because the model has to preform the same task: classify images into a limited set of categories (both data sets have 10 distinct categories).

CNN MNIST

The CNN MNIST model we used was an adaptation of the `mnist_cnn.py` of the Keras team found on their Github[2]. While initially using the Geron (2019)[1] CNN model on pp. 461, trying to fine-tune this by reducing the amount of convolution layers, the model quickly resembled the Keras team network. However, the main reason for the quick model switch was based on the amount of time it took for the Geron network to train. Due to this being a large hindrance when trying to fine-tune the model, a switch was made. The first layer of the model was a `Conv2D` layer with 28 filters and kernel size of (3,3). After that a `MaxPooling2D` layer was used with a poolsize of (2,2). We than used a `flatten` layer. This layer was followed by a `Dense` layer with 128 neurons, which was followed up by a `Dropout` layer having a dropout with a rate of 0.2. Finally a `Dense` layer with 10 neurons was used (again for the 10 different categories). The default activation function for these layers was a `relu` function, except for the last layer which used a `softmax` function. The default initializor was set to create tensors with a random uniform distribution. The default optimizer used as an `RMSprop` with `learning rate` of 0.001 and `rho` equal to 0.9. Similarly to the MLP, the dropout rate was kept at 0.2 as setting this rate too high worsened results. The kernel sizes were kept at 3 by 3 and 2 by 2 respectively, as increasing the size had a negative impact on the accuracy. Instead of a max pooling layer, a mean pooling layer was implemented, but this also proved to be unsuccessful. It seems that for the MNIST data set (as well as the Fashion MNIST) max pooling is more accurate, as it can more easily identify the bright pixels in contrast to a dark background.

CNN Fashion MNIST

The CNN model for fashion was build up in a similar way as the CNN MNIST was. For reasons mentioned above, this model also took the Keras team CNN as a basis, and therefore also started with a `Conv2D` layer with 28 filters with a kernel size of 3. After that a `MaxPooling2D` layer was used, with pool size equal to 2. Then a `Dropout` layer with rate equal to 0.2. We then again used a `Flatten` layer. After this layer came a `Dense` layer with 28 neurons and lastly another `Dense` layer with 10 neurons. This model used the same default activation, initializers and optimizer as the CNN MNIST model. In the end, the CNN model for the Fashion MNIST data set ended up the same as the CNN for the MNIST data set, even after parameter tuning, assumably for the same reasons the MLP models are the same for both data sets.

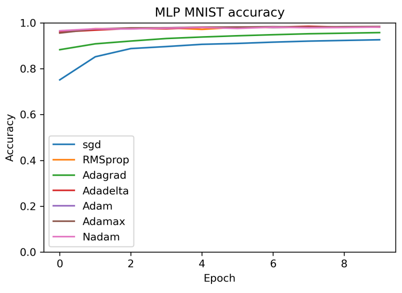
All of the model comparisons made were done so under early stopping to prevent overfitting. Time considerations were also regarded. Post experimenting with layers, neurons, kernels, batchsize etc it seemed that with more parameters, overfitting occurred sooner than when using the simpler models. It also took considerably more computational power/time to train these more complex models. The only upside of these more complex models was the slight increase in model accuracy. However, the downsides to these models far outweigh this very minor accuracy increase.

The experiments
To optimize the models, we experimented using different optimizer, activation and initialization functions. To conduct these experiments, we wrote a function that takes a list of different parameters we wanted to test (for example different activation functions) and replaces the activation functions of the default model. We then evaluated the model by inspecting both the accuracy of the model and the loss function. Based on these measures we judged the performance of the model. The results of these experiments can be found in the graphs below.

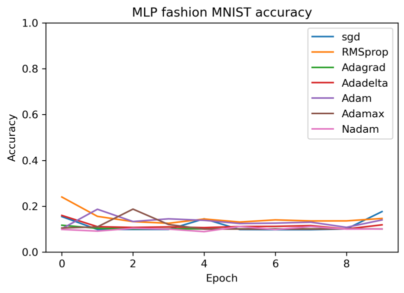
Because early stopping was not implemented in the experiments for the optimizer, activation and initializer selection, validation accuracy was used as a more unbiased metric to determine the "optimal" function to be used in task 2. No more than 10 epochs were used for these experiments, taking time and computational considerations in regard.

1.2 Results

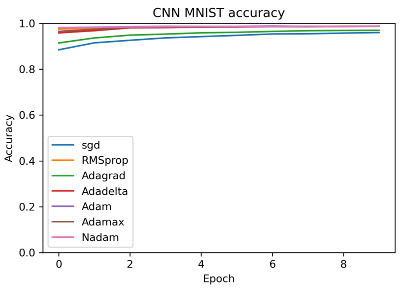
Optimizers
The optimizers we tested where: `sgd, RMSprop, Adagrad, Adadelta, Adam, Adamax and Nadam` functions (figure 1). The optimizers for the MLP models tended to perform similair to each other with a slight exception for `sgd` for the MLP MNIST model. For the CNN models the optimizers also tended to perform similar. Both CNN models had `sgd` as worst performer and `RMSprop` as best.



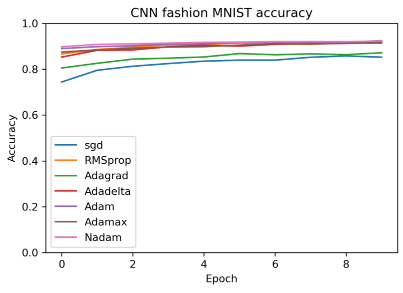
(a) MLP MNIST



(b) MLP Fashion



(c) CNN MNIST

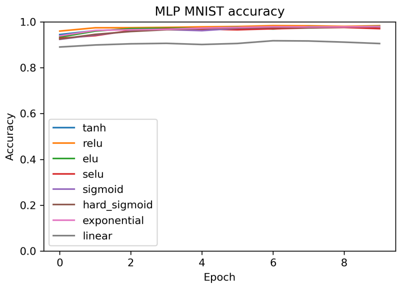


(d) CNN Fashion

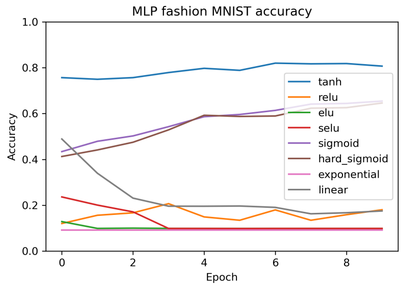
Figure 1: Optimizers

Activations

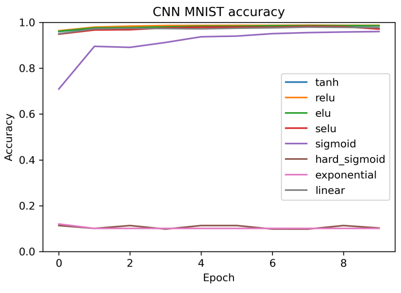
Activation functions we tested where:`tanh, relu, elu, sigmoid, hard sigmoid, exponential and linear` functions (figure 2). In the MLP MNIST model these activation functions tended to perform similar except for the `linear` activation. In the MLP Fashion model however, only `tanh` performed well (figure 2b). In both CNN models activation functions where similar however, in both models `hard sigmoid` and `exponential` functions under performed greatly compared to the other functions (figure 2c and 2d).



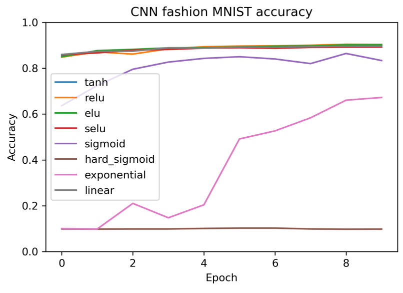
(a) MLP MNIST



(b) MLP Fashion



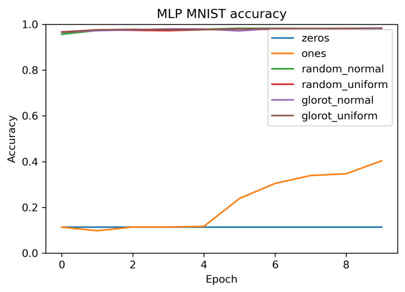
(c) CNN MNIST



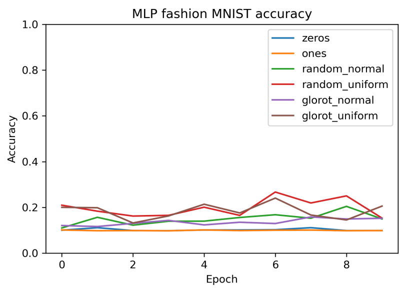
(d) CNN Fashion

Figure 2: Activations

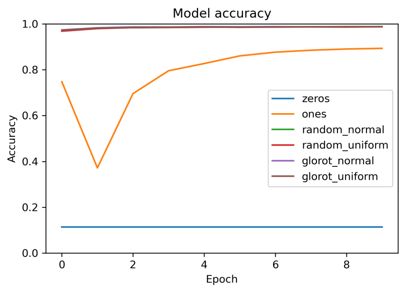
Initializer weights
The initializer weights we used where `zeros, ones, random normal, random uniform, glorot normal and glorot uniform` weights (figure 3). The pattern that popped out immediately was that `zeros` and `ones` as weights always under performed compared to the other options. The other options where close together in performance in all models and mattered only slightly.



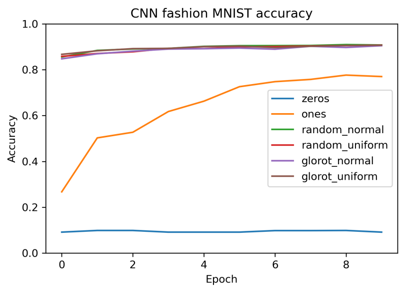
(a) MLP MNIST



(b) MLP Fashion



(c) CNN MNIST



(d) CNN Fashion

Figure 3: Initializers

1.3 Conclusions

The choice of `optimizers` did not impact the accuracy of these models greatly. choosing an `optimizer` seems to be mostly guided by experimenting which one is the best for the model given the data set. The same, although they differed more, can be said about the `activation` functions. The `initializer weights` except for `zeros` and `ones` mattered little for the accuracy of the models.

2 Testing the impact of obfuscating data by randomly permutating all pixels

After finding the optimal hyperparameters based on a 10 epoch scan of validation accuracy, all images from the MNIST and fashion MNIST datasets were permuted based on a single permutation. The same MLP's and CNN's were trained on the normal non-permuted data, and the permuted data, using the optimal hyperparameter settings found in task 1. The training accuracy of these models are displayed in the graphs below. The test accuracy metric of these models are discussed afterwards.

2.1 MNIST permutation

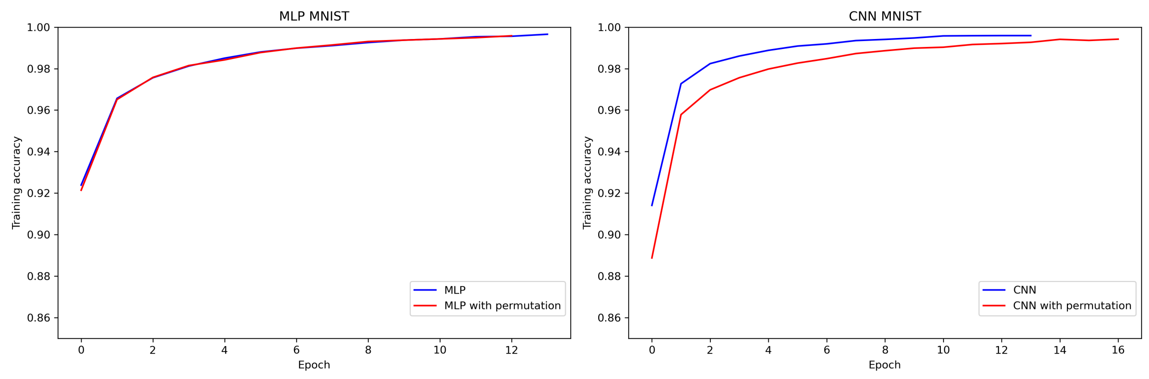


Figure 4: Training accuracy on permuted and non-permuted MNIST data

Comparing the red and blue line in each graph of figure 4, one may observe that the MLP models perform very similar indifferent of data permutation. This is to be expected as only a single permutation is used for all images, meaning there are still the same amount of features to be learned from the images, just in different places on the image. This does not pose a problem for the MLP. Zooming in on the CNN on the MNIST dataset (2nd graph of figure 4), we can see that early stopping to prevent overfitting caused the CNN model on the non-permuted data to have run through a lower number of epochs than the CNN based on the permuted data. In fact, all of the models stopped early to prevent overfitting, thus making comparisons on the test accuracy more reliable.

One can also observe more of a difference in training accuracy between the CNN models compared to the difference between the MLP models. This can be explained due to the way CNN's work. Post convolving of the image done in the convolutional layer of the CNN through the filter, the network is left with a new representation of the input, which is more ”smoothed” to the pixels around it. For regular non-permuted images this actually allows the network to detect patterns better, but for permuted images the operations done in the convolutional layer backfire a bit due to the permutation throughout the entire grid of the image, and smoothing around those randomized pixels.

The same differences, although very minor, can be detected when comparing the test accuracy's of the networks. The test accuracy's of the MLP's are 0.9831 and 0.9836 respectively, netting a difference of approximately 0.0005 in testing accuracy between the MLP's of permuted and non-permuted MNIST images. The difference in test accuracy for the CNN's is approximately 0.005, which is an entire order size or magnitude larger. Even though it is still small, this is still quite a difference compared to the testing accuracy difference found for the MLP's.

2.2 Fashion MNIST permutation

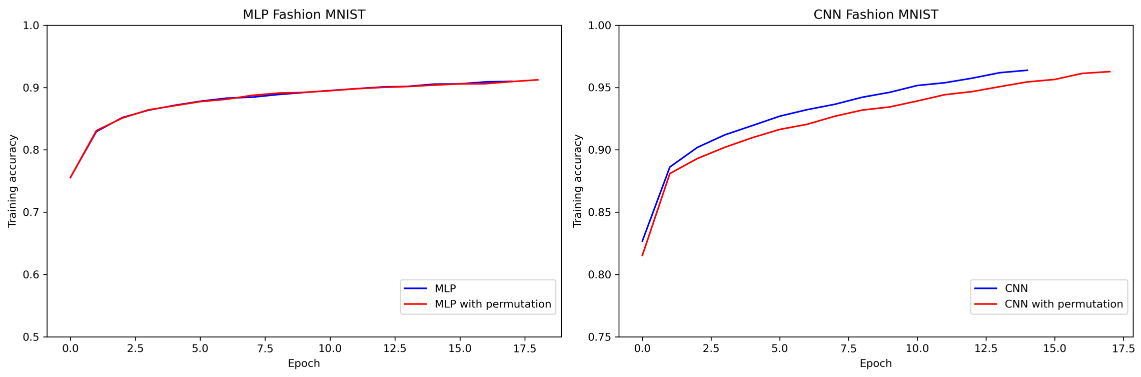


Figure 5: Training accuracy on permuted and non-permuted Fashion MNIST data

Turning our attention to the fashion MNIST model metrics displayed in figure 5, we can detect the same differences in training accuracy between MLP's and CNN's, and the same differences mutually between MLP's and CNN's. The same explanations hold for this dataset as well. Due to the nature of the clothing images (harder to classify than number MNIST), the training - and testing accuracy will consistently be lower compared to the regular MNIST model metrics discussed in section 2.1.

The test accuracy's of the fashion MNIST MLP's are 0.8824 and 0.8819, netting a difference of approximately 0.0005 in test accuracy between the MLP's of permuted and non-permuted MNIST images. This (not very surprising) difference approximately is the same difference found for the MLP's of the number MNIST dataset. The difference in test accuracy for the CNN's for the fashion MNIST data is approximately 0.002, which is even less than an order size larger compared to the MLP difference.

Thus one can conclude that CNN's perform consistently worse on permuted data, whilst MLP models are indifferent to permutation (for these two specific MNIST datasets). Adding more convolutional layers would probably make this conclusion more apparent than it is now as reflected by the testing accuracy differences and training accuracy graphs.

3 Develop a “Tell-the-time” network

3.1 Architecture

The time telling problem was interpreted as a two-fold problem. For one, the amount of hours needed to be correctly classified as one of twelve categories and the amount of minutes was classified using linear regression. The thought process behind this is rather straightforward: as there are only 12 different options for the hour values, multiclass classification seems to be the obvious choice, whereas there are 60 possible values for minutes. Because of this, a regression is likely to preform better than a classification, as it is hard to differentiate between the classes for the minute values. This led to a CNN model containing a separate output for hour and minute. However, considering the neural network preforms best best with values on the interval $[0, 1]$, the minute values had to be normalized, as the current values were on the interval $[0, 60]$. This was easily done by dividing these values by 60. The input for this network was an array with dimensions $150 \times 150 \times 1$. Similar to the CNN’s in task 1 and 2, the `Maxpooling2D` (the data consists of 2D images) layer was used to reduce the dimensionality of the images. Even though `BatchNormalization` layers are not uncommon in neural networks, while building the model, a few of these layers were added as a test. These were quickly discarded however. With these layers, while changing the batch size, amount of neurons and other parameters, the minute accuracy would never go below 10 minutes. However, without these `BatchNormalization` layers, accuracies of below 5 minutes were attained. Furthermore, like in the CNN and MLP of tasks 1 and 2, a `Dropout` layer was included, with the dropout rate set to 0.2. The exact value did not seem to matter, but if the rate was set to 0.5 or over, the model accuracy was reduced significantly. What seemed to matter the most was reducing the batch size: with a higher batch size, the model seemed unstable, even at a size of 50, the results varied a lot, but with the batch size brought down to 32, the results became very consistent. With a larger batch size (256), the minute accuracy turned out to be consistently worse, see figure 6.

So far only minute accuracy has been discussed, but this model has two outputs, the hour accuracy being the other one. However, all the fine tuning of the model that has been discussed above, and improved the minute accuracy, also improved the hour accuracy. Here too, batch size seemed to impact the hour accuracy significantly: by going from a $< 50\%$ hour accuracy to a consistent $> 93\%$ hour accuracy. For the two output layers, hour and minute, their respective last layers consisted of a `Softmax` and a `Linear` activation layer. The choice for the `Softmax` activation layer was pretty straightforward: it preforms very well on classification tasks due to it returning a probability distribution over the target classes in a multi-class classification problem. The same function was used in the networks for task 1 and 2, which also had a classification problem. The linear activation function is the obvious choice for linear regression problems. This then comprised the full time telling model.

3.2 Optimization

Even though the model accuracy was high, we tried to improve this further by using the methods described before: finding the optimal optimizer, initialization and activation functions. As can be seen in figure 7, for the various optimizer functions, the `Adam` and `RMSprop` start to outpreform the others after about 4 epochs. Concering the activation functions, the results are more or less similar in terms of the minute accuracy of the model. Finally, judging from the graph with different initializers, both `zeros` and `random_uniform` preform equally well, though not much better than the other initializers. With this taken into account, the model was run again with an `Adam` optimization function, a `relu` activation function and a `random_uniform` initializer. Even with these seemingly ”best” settings, the minute mae did not go under the 4 minute treshold.

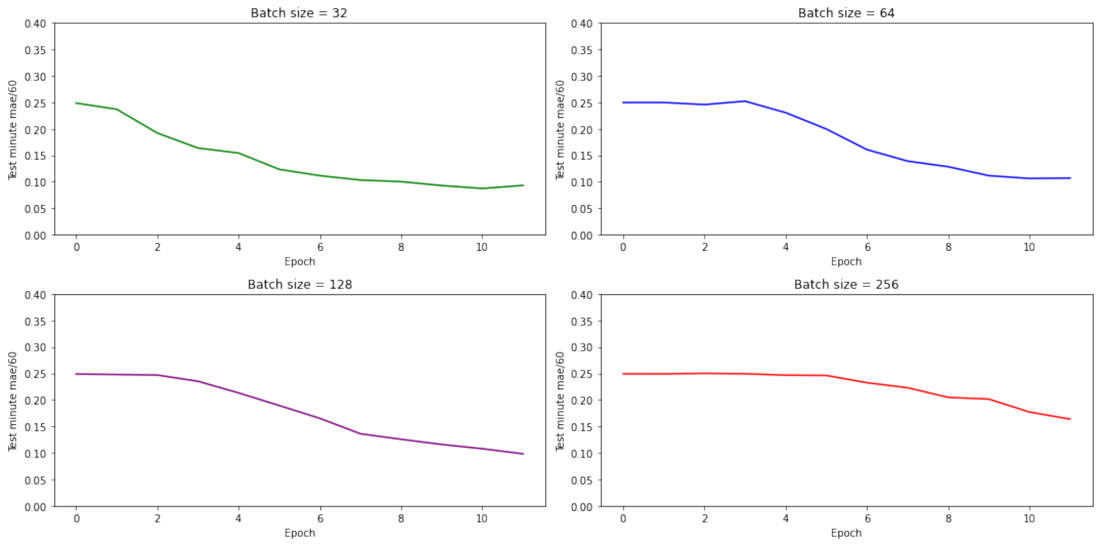


Figure 6: Influence of batch size on minute mean absolute error/60 (test set), where a lower mae is better

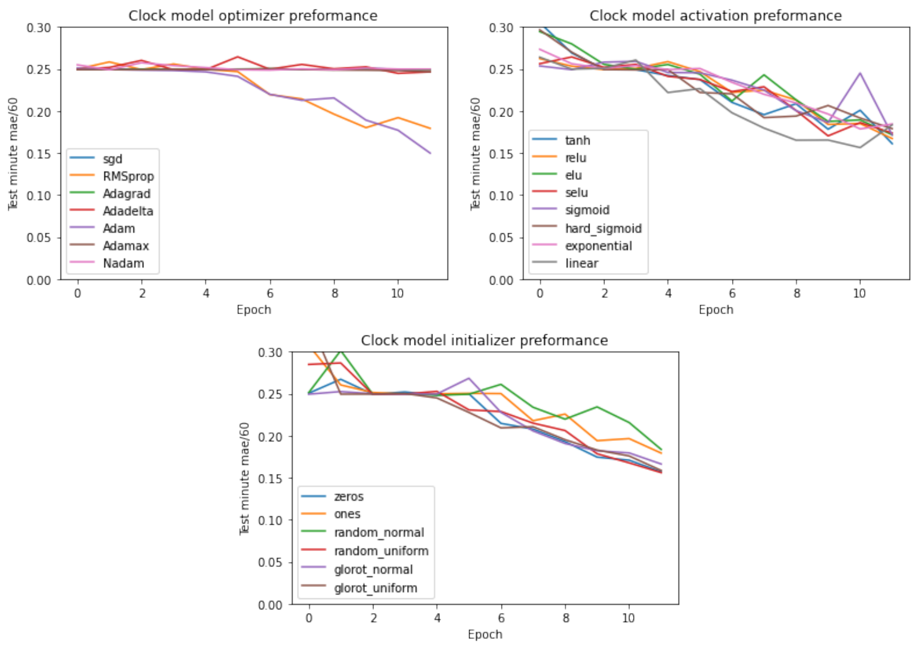


Figure 7: Experimenting with model compile settings, where a lower mae is better

References

[1] A. Geron. Hands-on machine learning with scikit-learn, keras, and tensorflow. 2019.

[2] Keras Team. https://github.com/keras-team/keras. 2020.