Introduction to Deep Learning
Assignment 1

J.D. Meijerhof (s1453491)
Wilson (s3120163)

November 12, 2021

1 Experiment in Keras and Tensorflow

1.1 Background

The main purpose of this task is to carry out various machine learning experiments with different hyper parameters settings over the two data set (MNIST, Fashion MNIST) and neural work (MLP, CNN). In order to make the task manageable and scalable, a script is created — to generate hyper parameter combinations, to execute cases that is traceable, to save record in an reusable way — to facilitate huge amount of execution sets with wide range of hyper parameters variations. The details of the script features are further described in section 1.2.

1.2 Methodology

1.2.1 Features to support multiple hyper parameters combinations

In order to manage a vast amount of cases from parameter variations, several important features are developed to make things easier, including a feature that generates the hyper parameters combinations; a simple way to dynamically execute specific range of cases; a way to store the results for further processing and analysis. Despite the ability to generate thousands of cases, it is not feasible to execute all of them with limited time and computational resource. Thus, we have selected some parameters that are considered to be more meaningful to test with. This is further discussed in next section 1.2.2.

The generate_mlp_dict_content.py file generates all the possible combinations from given parameters list in each hyper parameters defined in the beginning part of the same file. For instance, if L1 list has [0, 0.01] and Dropout list has [0.2, 0.4]. Then 4 cases with different hyper parameters combinations will be generated. Those generated hyper parameters settings with an unique id will then be manually copied to a dict in the "hyper_para_mlp_dict.py" and "hyper_para_cnn_dict.py" file. This identifier can then be used for case execution, and also for back tracing from output results.

Four core .py files (task1a_mlp_mnist.py, task1b_cnn_mnist.py, task1c_mlp_fashion_mnist.py, task1d_cnn_fashion_mnist.py) are created with the combinations of the two data sets and neural networks. It will execute the case with the hyper parameters according to the given id "execute_dict_id_list" in the code.

The results of the cases are being stored in a csv defined by the variable path "result_output_file" The csv contains the hyper parameter id that is used for the case, which makes it easier to back trace what parameters is being used, the csv also serves the purpose to indicate which case is being executed already.

1.2.2 Parameter selection

There are many hyper parameters settings available such as layer, loss functions, optimizer, activation, initializer, etc, the number of cases grows exponentially when we include more parameters and values variations. There are more than 10 hyper parameters available for tuning and even just assigning 3 variations among each of those will already create 10^3 = 3000 cases.
We have the workflow to support the execution of this amount of cases, but we noticed that it is not feasible to execute them all for the given time. Thus, we have to reduce the amount of epochs and hyper parameters values. From a few MLP and CNN runs, we realized CNN has much longer execution time

MLP. For this reason, we set the epoch for MLP and CNN to 20 and 2 respectively. On he other hand, we selected eight hyper parameters for values variations (hyper_para_CNN, MLP_dict.py). In MLP, six of them has two variations except L1 and L2 has 3 variations (0, 0.01, 0.001), where 0 represents no L1 or no L2. To further reduce the number of cases in CNN, 0.001 is removed from L1 because L1 has lower importance due to its lower effectiveness as compared to L1. In the end, there are 576 cases for MLP and 384 cases for CNN, by applying it to the two data set, the total number of cases to run is 1920.

1.3 Result

The 1920 cases are executed which took more than 4 days in total. The raw results data can be found at this link. (https://docs.google.com/spreadsheets/d/1tJDjNMBsyU7-MDNys4DIcHri6tuxgbpFegHJOckUHeA/edit?usp=sharing). The data is being further processed and sorted in descending order by Test Accuracy, Training Accuracy and Validation Accuracy. The top 10 results of each process are shown in the four tables below.
Layer data in table 1 and 3 represents the MLP neural network structure. The first integer and second integer represents the number of neurons at first and second layer respectively. Struct ID in table 2 and 4 represents the Neural Network structure ID which the structure is shown at 1.3. "Ran Nor" and "cat cro" is a short hand of Random normal and categorical crossentropy. Dout stands for Dropout, Opt stands for Optimizer and Init stands for Initializer of the weights.

Table 1: Best performed hyper parameters settings for MNIST with MLP

ID	Init	L1	L2	Act	Dout	Layer	Loss	Opt	Train Acc	Validation Acc	Test Acc
10220	uniform	0.0	0.0	sigmoid	0.2	(256, 128)	cat cro	Adam	0.9856	0.9804	0.9804
10508	Ran Nor	0.0	0.0	sigmoid	0.2	(256, 128)	cat cro	Adam	0.9872	0.9788	0.9788
10072	uniform	0.0	0.0	sigmoid	0.0	(512, 256)	cat cro	Adam	0.9923	0.9773	0.9773
10366	Ran Nor	0.0	0.0	sigmoid	0.2	(512, 256)	mse	Adam	0.9764	0.9771	0.9771
10218	uniform	0.0	0.0	sigmoid	0.0	(256, 128)	mse	Adam	0.9871	0.9769	0.9769
10360	Ran Nor	0.0	0.0	sigmoid	0.0	(512, 256)	cat cro	Adam	0.9946	0.9766	0.9766
10216	uniform	0.0	0.0	sigmoid	0.0	(256, 128)	cat cro	Adam	0.9943	0.9763	0.9763
10078	uniform	0.0	0.0	sigmoid	0.2	(512, 256)	mse	Adam	0.9766	0.9762	0.9762
10076	uniform	0.0	0.0	sigmoid	0.2	(512, 256)	cat cro	Adam	0.9868	0.9761	0.9761
10364	Ran Nor	0.0	0.0	sigmoid	0.2	(512, 256)	cat cro	Adam	0.9825	0.9757	0.9757

Table 2: Best performed hyper parameters settings for MNIST with CNN

ID	Init	L1	L2	Act	Dout	Struct ID	Loss	Opt	Train Acc	Validation Acc	Test Acc
20288	Ran Nor	0.0	0.0	relu	0.0	2	cat cro	Adam	0.9826	0.9851	0.9851
20100	uniform	0.0	0.0	relu	0.2	2	cat cro	Adam	0.9735	0.9851	0.9851
20000	uniform	0.0	0.0	relu	0.0	1	cat cro	Adam	0.9845	0.9841	0.9841
20004	uniform	0.0	0.0	relu	0.2	1	cat cro	Adam	0.9786	0.9836	0.9836
20290	Ran Nor	0.0	0.0	relu	0.0	2	mse	Adam	0.9746	0.9803	0.9803
20304	Ran Nor	0.0	0.001	relu	0.0	2	cat cro	Adam	0.9676	0.9779	0.9779
20198	Ran Nor	0.0	0.0	relu	0.2	1	mse	Adam	0.9582	0.9765	0.9765
20294	Ran Nor	0.0	0.0	relu	0.2	2	mse	Adam	0.9563	0.9765	0.9765
20308	Ran Nor	0.0	0.001	relu	0.2	2	cat cro	Adam	0.9499	0.9757	0.9757
20292	Ran Nor	0.0	0.0	relu	0.2	2	cat cro	Adam	0.9481	0.9746	0.9746

Table 3: Best performed hyper parameters settings for Fashion MNIST with MLP

ID	Init	L1	L2	Act	Dout	Layer	Loss	Opt	Train Acc	Validation Acc	Test Acc
10288	Ran Nor	0.0	0.0	relu	0.0	(512, 256)	cat cro	Adam	0.8959	0.8789	0.8789
10220	uniform	0.0	0.0	sigmoid	0.2	(256, 128)	cat cro	Adam	0.8753	0.8753	0.8753
10432	Ran Nor	0.0	0.0	relu	0.0	(256, 128)	cat cro	Adam	0.8969	0.8721	0.8721
10504	Ran Nor	0.0	0.0	sigmoid	0.0	(256, 128)	cat cro	Adam	0.8951	0.8718	0.8718
10076	uniform	0.0	0.0	sigmoid	0.2	(512, 256)	cat cro	Adam	0.8649	0.8705	0.8705
10000	uniform	0.0	0.0	relu	0.0	(512, 256)	cat cro	Adam	0.8942	0.868	0.868
10508	Ran Nor	0.0	0.0	sigmoid	0.2	(256, 128)	cat cro	Adam	0.876	0.8677	0.8677
10510	Ran Nor	0.0	0.0	sigmoid	0.2	(256, 128)	mse	Adam	0.8666	0.8677	0.8677
10144	uniform	0.0	0.0	relu	0.0	(256, 128)	cat cro	Adam	0.9004	0.8669	0.8669
10222	uniform	0.0	0.0	sigmoid	0.2	(256, 128)	mse	Adam	0.8705	0.8653	0.8653

Table 4: Best performed hyper parameters settings for Fashion MNIST with CNN

ID	Init	L1	L2	Act	Dout	Struct ID	Loss	Opt	Train Acc	Validation Acc	Test Acc
20004	uniform	0.0	0.0	relu	0.2	1	cat cro	Adam	0.887	0.8979	0.8979
20196	Ran Nor	0.0	0.0	relu	0.2	2	cat cro	Adam	0.8756	0.8926	0.8926
20100	uniform	0.0	0.0	relu	0.2	2	cat cro	Adam	0.875	0.8898	0.8898
20102	uniform	0.0	0.0	relu	0.2	2	mse	Adam	0.8633	0.8845	0.8845
20002	uniform	0.0	0.0	relu	0.0	1	mse	Adam	0.8862	0.8814	0.8814
20288	Ran Nor	0.0	0.0	relu	0.0	2	cat cro	Adam	0.8818	0.8814	0.8814
20000	uniform	0.0	0.0	relu	0.0	1	cat cro	Adam	0.8779	0.8798	0.8798
20292	Ran Nor	0.0	0.0	relu	0.2	2	cat cro	Adam	0.8517	0.8757	0.8757
20006	uniform	0.0	0.0	relu	0.2	1	mse	Adam	0.8521	0.8707	0.8707
20198	Ran Nor	0.0	0.0	relu	0.2	2	mse	Adam	0.8539	0.8701	0.8701

```
model = Sequential()
if network_structure_id == 1:
    model.add(Conv2D(32, kernel_size=(3, 3))
    model.add(Conv2D(64, kernel_size=(3, 3))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout))

    model.add(Flatten())

    model.add(Dense(128))
    model.add(Dropout(dropout))

    model.add(Dense(num_classes, activation='softmax'))

elif network_structure_id == 2:
    model.add(Conv2D(64, kernel_size=(3, 3))
    model.add(Conv2D(32, kernel_size=(3, 3))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout))

    model.add(Flatten())

    model.add(Dense(64)
    model.add(Dropout(dropout))

    model.add(Dense(num_classes, activation='softmax'))
```

Code 1: Network structure referring to Network Structure ID (Struct ID)

1.4 Discussion

The four tables from the result section reveals a lot of information. It is observed that both networks have higher performance in MNIST classification over fashion MNIST. The overall performance between MLP and CNN is broadly similar, but it is worth to mention that MLP has 20 epochs execution whereas CNN only has 2. However, the total execution time of MLP is faster than CNN. In Random Initialization settings, there are not much difference regardless of what strategy is being used. L1 and L2 mostly work best when it is 0, this could be because the complexity of the network is just fit to the problem, which here are no overfit adjustment needed. There are a high diversity in activation function. Relu stands out in CNN network where all the top settings has relu. However, settings in MLP are not as consistent. It has Sigmoid dominated the MNIST data set but only half in Fashion MNIST set. Dropout rate seems to not have much affect to the results to all 4 result set as we see an even distribution of it over all top records, which is also the case for which layer or structure being use. This could be because the difference of each of them are not high enough. I have found that some good dropout rates could go up to 0.5, and of course the network could also be much more complicated. However, we decided to only try out 2 combinations due to limited time and the exponential growth of cases for any extra value variation. For loss function, categorical crossentropy seems to have slight advantage in MNIST with MLP, or otherwise it performed quite similar as MSE. On the other hand, the result reveals that Adam (learning rate 0.01) owns all of the highest positions, which rules out SGD from any of the top 10.

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

2.1 Result

Table 5: Performance of each dataset in each neural networks with the best performance found in Section 1

Network	DataSet	Hyper Para ID	Train Accuracy	Validation Accuracy	Test Accuracy
MLP	MNIST	10220	0.6673	0.1956	0.1956
CNN	MNIST	20288	0.1105	0.1134	0.1134
MLP	Fashion MNIST	10288	0.2671	0.2599	0.2599
CNN	Fashion MNIST	20004	0.7346	0.7480	0.7480

2.2 Discussion

Table 5 indicates the performance of the four process over the permutated dataset with the highest rank settings found in section 1. The result reveals that there are high diversity among the test cases. In the MLK MNIST case, it looks like it has an over fitting problem, where the training accuracy is very high compare to the test accuracy. CNN MNIST set has a very low accuracy, which appears to have high bias, which is also similar to what is observed in CNN Fastion MNIST set. Lastly, the CNN Fashion MNIST set looked normal among the four result sets that is actually abnormal. Although it has a reasonably high train and test accuracy, it does not make senese that it performs better than the other three result sets. It is because the other three result set has a lower difficulty. One possible reasons is that the random permutation "unluckily" permutated it in a way that has not much difference since each process has their own permutation set. However, this chance are not high in general.

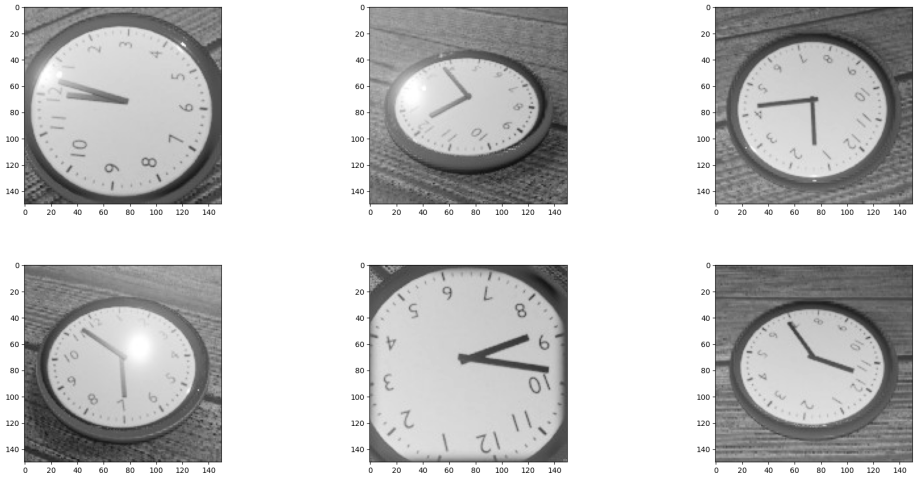

Figure 1: Six examples from the *tell-the-time* data set.

3 The *tell-the-time* problem

The *tell-the-time* problem consists of building a neural network to read an analog clock. A data set has been provided consisting of 1.8×10^5 150x150 pixel grayscale images. Each image has a label consisting of two digits, one indicating the hour and one indicating the number of minutes after the hour.
Six example clocks are shown in Figure 1. These examples illustrate the differences between the images in the data set. Clocks can be rotated with regards to each other. They are photographed from far and near distances, under different angles. Sometimes the reflection of a light source obfuscates part of the clock. Despite these differences each image is built up from the same components. Each clock has an edge, a dial with Arabic numerals, stripes indicating each minute and two pointers. This makes telling the time a problem that is excellently suited to the capabilities of *convolutional neural networks*. In this section we shall describe the implementation and results of two types of networks. One that treats the *tell-the-time* problem as a regression problem and one that treats it as a classification problem.

3.1 Regression

If we wish to use a regression algorithm, we need to express time as a single number instead of two. A simple function to use is:

$$t_{\text{float}} = hh + mm/60 \tag{1}$$

Where hh is the number of hours and mm is the number of minutes in the digit representation of time. The difference between any two times t_1 and t_2 can be calculated by:

$$\Delta t = \min\left(|t_1 - t_2|, 12 - |t_1 - t_2|\right) \tag{2}$$

This way of calculating the minimum ensures we calculate a time difference of 10 minutes between 00:05 and 23:55. If we had naively just used the absolute value of the difference between t_1 and t_2 a difference of 23 hours and 50 minutes would have been obtained. Float representation here has been omitted for readability.
The set of images is split into three sets. A training set consisting of 70% percent of all images, a validation set of 10% and a test set of 20% of all images. As the images are sorted in time they are randomly shuffled before dividing into the set.

3.1.1 Network Design

We implement the simple CNN described in Geron [2019], page 461. However, since our input images are much larger than the images in Fashion MNIST we increase the filter size of the first convolution layer, as well as the stride. The network setup is as follows:

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=64, kernel_size=10, strides=(3,3),
                        padding="same", activation="relu",
                        input_shape=[150, 150, 1]),
    keras.layers.MaxPooling2D(2),
```

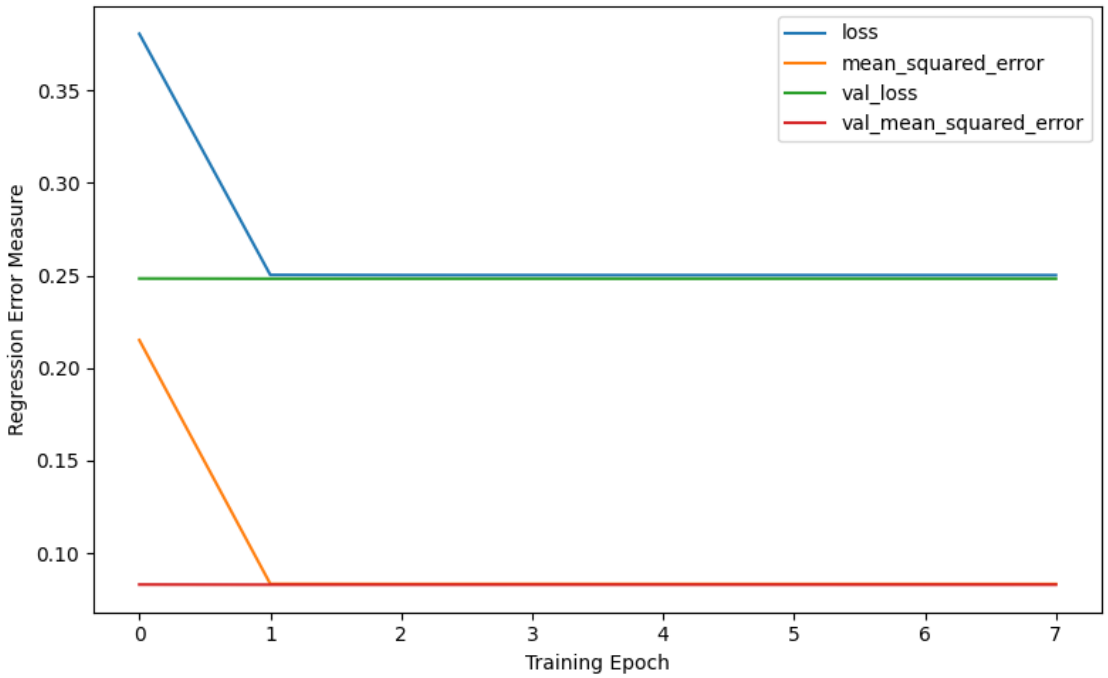

Figure 2: Caption

```
    keras.layers.Conv2D(filters=128, kernel_size=3, strides=(1, 1),
                        padding="same", activation="relu"),
    keras.layers.Conv2D(filters=128, kernel_size=3, strides=(1, 1),
                        padding="same", activation="relu"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(filters=256, kernel_size=2, strides=(1,1),
                        padding="same", activation="relu"),
    keras.layers.Conv2D(filters=256, kernel_size=2, strides=(1,1),
                        padding="same", activation="relu"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation="sigmoid"),
])
```

Note that the model has a single output, which we want to compare to the float representation of the true time discussed before. The final layer uses a Sigmoid activation, causing the output to fall in $[0,1]$. The true time values are therefore normalised from $[0,12)$ to $[0,1]$. The optimisation function used is Stochastic Gradient Descent (SGD) with a learning rate of 0.01; the momentum parameter is not used. The loss function used is the Mean Absolute Error (MAE): `keras.losses.MeanAbsoluteError()`. This error measure was used, because using Equation 2 requires implementing a custom `keras` loss function. Unfortunately we could not get this working.

3.1.2 Training Results

The training progress of this network is shown in Figure 2. The network converges after two training epochs. The achieved error however is not so impressive. On both the train, validation and test set a MAE of 0.25 was achieved. A MAE of 0.25 equates to a mean absolute error of 3 hours. This seems to be a consequence of the error measure used, as discussed in the previous section.

3.1.3 Weaknesses of the Regression approach

One obvious weakness to the regression approach is that the network is trying to squeeze too much information into one parameter. After all, a clock has two pointers. An error made in the number of hours will make the network adjust its weights by a lot, even if it had predicted the number of minutes, corresponding to the large pointer, correctly. So even two outputs would make things a lot easier, as the network could learn to read the small and large pointer separately.

References

Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edition).* O'Reilly, 2019.