Deep Learning
Assignment: MLPs and CNNs in Keras/TensorFlow

Bella Shao

November 18, 2021

Abstract

In this report, we perform three tasks related to Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). We use Keras and Tensorflow to familiarise with the different parameters of the MLPs and CNNs and make models using different combinations of the parameters. In the task, we try to develop a CNN that can recognize the time on different clock images.

1 Develop a "Tell-the-time" network

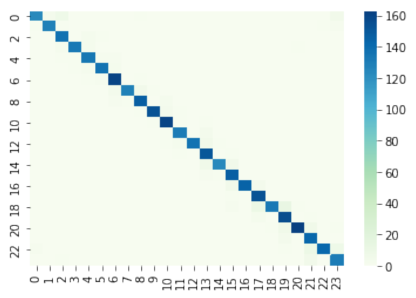
For this task, we experiment with different methods and neural network architectures to "Tell-the-time" on various clock images. The size of clock images is 150×150 with time span from $0 : 0 \rightarrow 11 : 59$. The goal of this task is to recognize time points as accuracy as possible.The methods we will experiment with are as follows: Regression, n-Class classification, Multi-output model as well as re-labeling time points.

1.1 Regression

When training with regression CNN, we ignore activation function in the final output layer. At the first attempt, we constructed a simple architecture with only 3 CNN layers. The first Conv2D layer has filters of size $64 \times 5 \times 5$, followed by the second Conv2D layer with filters of size $128 \times 3 \times 3$, then the last Conv2D layers with filter size $256 \times 3 \times 3$. After each Conv2D layer, we add a MaxPooling2D layer of size 2. Subsequently, in order to mitigate overfitting and improve prediction accuracy, one layer of BatchNormalization and another layer of Dropout are both added to CNN layers. Except for final output layer, Relu is applied as the activation function in all hidden layers. Target labels are transformed into float values. The formula is $hour + minute/59.0$. But this 3-layer model performs really poor, where the mean absolute errors can be as big as 7.98. After some trails and errors testing, a better performing model is build by adding another CNN layer before the last CNN layer and the size is the same as the second layer, which has $128 \times 3 \times 3$ filters. Moreover, we also modify the first Conv2D layer filter size to 7×7

Training this model with epoch equals to 60, we observed this model converges fast and can easily get over-fitting. Since it's a regression model, we choose mean squared errors as loss function, mean absolute errors as model metric. The optimizer is "Adam" with learning rate of 0.001. The average loss is around 0.7775. Rounding the predicted time to one decimal points, the mean absolute error is around 0.51, which is approximately 30 minutes (Every 30-minute interval is 0.5 on the transformed scale).

Problems of this regression model might be inherited in the label transformations. We transfer two-digit label to one float value, as given for instance, $3 : 30 \rightarrow 3.5$. We keep hours as discrete digit class, but transform minutes to float values, which adds extra biases and variances into labels. To this end, the regression model is easily getting over-fitted and could not preserve the discontinuity in the


(a) 24-class Accuracy 94.42% in testset

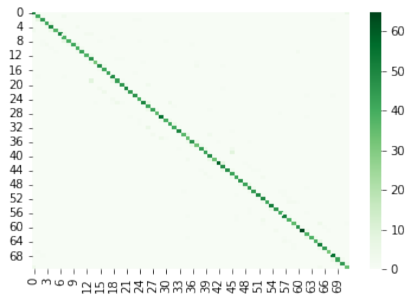

(b) 72-class Accuracy 88.75% in testset

Figure 1: 24 and 72 Classes

original labels. In addition, by rounding time floats to one decimal point, different time points will become the same. For instance, 2 : 29 and 2 : 30 both become 2.5. In this way, different time points can be easily predicted into one value, which makes it even harder to distinguish. Moreover, this kind of label transformation treats time as linear, but in fact time points are circular, a straight line could not be a suitable representation.

1.2 Classification

In order to mitigate the problem of overlapping time labels, in this section we experiment with an n-Class classification method, which treats time labels as discrete categories. Following the instructions in the assignment, we group time labels into categories based on predefined time intervals and regard each category as a class. Subsequently, we can make the time intervals smaller and smaller, hence the number of classes increases. Our goal of this experiment is to check the effects on prediction accuracy of different time interval categorization strategies.

Firstly, we categorize all the time labels into 24 classes, and the interval for each class is 30 minutes (750 labels for each class). Based on the CNN architecture we constructed in regression method, after a few attempts and adjustments, this n-Class classification CNN is displayed in table 1. The output layer is with Dense 24 classes. Running 45 epochs with optimizer "Adam" and learning rate of 0.001, the resulting accuracy in the test set is 94.42%, and telling-time error in minutes is minimized to 6.09 minutes, which is below 10. The prediction is actually really good. However, the test accuracy is highly dependent on running epochs as well as the number of classes.

Subsequently, we repeat the experiment with 72 classes, where we group each 250 labels into one category. The architecture of this model is the same as before. The only difference is we change the output layer to Dense 72 classes. The model converges around 35 epochs and prediction accuracy for test data decrease to 88.75%. The error between true and predicted time points increase to 10.72 minutes, which is slightly higher than 10. Figure 1 shows the heatmaps for 24 and 72 classes respectively.

Continuing with the experiment, we categorize time labels into 180 classes, where we group every 100 labels as one class. Following the same CNN architecture, the resulting accuracy in the test data decreases to 60.06% with 35 running epochs. And the "telling-time" error increase to 44.33 minutes, which is much higher than 10. Furthermore, we also tried running this model with all 720 classes, where each time point is one class as saved in the original labels. The result is even

Table 1: n-Class

CNN Architecture	
Conv2D	64×7 × 7 *Relu*
Maxpool	2×2
BatchNormal	
Dropout	0.25
Conv2D	128×3 × 3 *Relu*
Maxpool	2×2
BatchNormal	
Dropout	0.25
Conv2D	256×3 × 3 *Relu*
Maxpool	2×2
BatchNormal	
Dropout	0.25
Conv2D	512×3 × 3 *Relu*
Maxpool	2×2
BatchNormal	
Dropout	0.25
Dense	240 Relu
Dense	240 Relu
BatchNormal	
Dropout	0.5

worse with only 43.14% accuracy after 40 epochs in the test data.

There arise several problems with increasing number of classes. Training the model becomes harder and harder because it takes longer time and more RAM. For instance, training with 720 classes makes my laptop crash. We only got the chance to train the model once on Google Colab with enough GPU. Additionally, with more classes, the model seems to be difficult to reach convergence. For instance, with 180 classes, after 35 epochs, the loss from training is already started to jumping back and forth, but the training accuracy is only at 80.69% and the model does not converge. Furthermore, the accuracy is even lower in test set. This situation is even worse with 720 classes, where the standard computing power cannot hold anymore. Running after 40 epochs, the training set can only hold accuracy at 58.57%, but the RAM is already used up.

1.3 Multi-output model

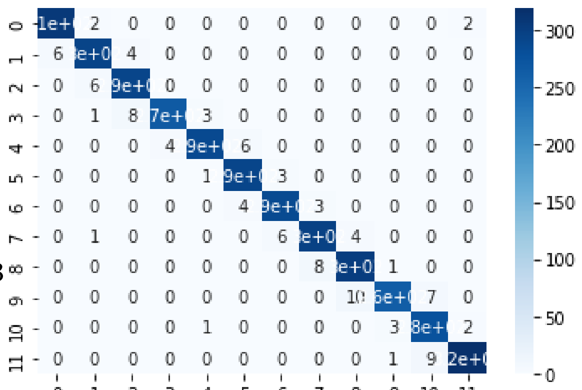
Accuracy of n-Class classification seems to depend heavily on how to categorize time classes, in this section we experiment with a multi-output model, where we split the output to two labels, one is for hour, and the other is for minute. In this way, we can avoid training with all 720 classes. Specifically, since the model produces two outputs at the same time, we can set hours output layers as classification, and minutes output layer as regression. Build of the model architecture is the same as n-Class classification in the CNN layers. Nevertheless, we construct different fully connected layers to incorporate with multiple outputs. Table 2 displays the fully connected layers of the multi-output model. All the other layers are the same as in Table 1.

Table 2: Multi-output

Multi-output Fully Connected layers	
Flatten	
Dense	144 Relu
BatchNormal	
Dense	144 Relu
BatchNormal	
Dropout	0.5
hour_outout	12 Softmax
Flatten	
Dense	200 Relu
BatchNormal	
Dropout	0.5
minute_output	Dense 1

We set Softmax activation function for hours output layer and none for minutes output layer. Consequently, the loss functions for both layers will also be different. For hours class, categorical cross entropy loss function is applied, whereas mean absolute errors loss function is used for minutes regression. Here arising an issue about how to assign loss weights for each output layer. At first attempt, we tried 1 : 1 loss weights for hours and minutes. It turned out that because these two layers are not in the same scale and format, they interference with each other while training. Hence the model was easily got stuck in local minimum and could not find its way out. To solve this problem, we assign 80% loss to hours class and 20% loss to minutes regression. After running 80 epochs with learning rate of 0.001, the model reaches convergence with really good results. In the test set, the accuracy for predicting hours classes approaches as high as 97.06%. The loss of mean absolute errors for minutes regression is as low as 1.762 minutes. And the total MAE between true and predicted time point is 5.062 minutes, which is below 10.

When first building this multi-output model, we also treated minutes as classes and performed a classification for minute output layer as well. The results are actually quite good. We reached accuracy for hour class as high as 96.89% and 85.53% for minute classes. Here only comes with a small issue that we have to make hours classes into



one-hot encoded matrix, while keeping minutes classes as discrete digits from $0 \rightarrow 59$.

Comparing the multi-output model with previous n-Class classification, the difference is that multi-output model is trained with only 2 labels (hour and minute), but n-Class have to use all 720 labels, which cannot be hold by average computing power. Moreover, the accuracy of multi-output model is much higher and more stable, whereas the accuracy of n-Class classification has to depend on the number of categorization of time labels.

1.4 Relabeling time points

Relabeling time points could also be a promising method to recognize time on the clock images. In this section, we experiment with re-labeling time with sine and cosine functions and then check how this kind of relabeling can impact the predicting accuracy.

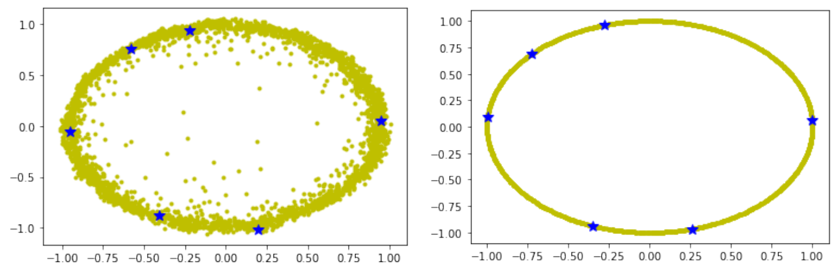
We treat clock face as a unit circle, which has radian of 2π. A clock face has 12 hours, thus 1 hour has radian of $2\pi/12$. Since each hour point on the clock circle has two axes, one is represented by cosine function on the x-axis, and the other is represented by sine function on the y-axis, we can calculate the position of hour points on the clock. Following the same method, each minute can also be expressed by sine and cosine functions, where the only difference is that one unit circle contains 60 minutes. The formula to calculate hour and minute label transformations are as follows:

$$hour_x = \cos(2\pi \cdot hour/12.0) \qquad hour_y = \sin(2\pi \cdot hour/12.0)$$
$$minute_x = \cos(2\pi \cdot minute/60.0) \qquad minute_y = \sin(2\pi \cdot minute/60.0)$$

In this way, we can perform a regression CNN for the reason that transforming time points to sine and cosine values can preserve their inherent locations on the clock circle. We can think about this in a way that we cut the clock circle and stretch it to a straight line, and all the points on this line are time points that are represented by sine and cosine values. To this end, regression is a suitable method for this data structure.

For the ease of training and saving time, we first combine hours and minutes together into time floats for every time points, the formula to calculate time floats is $timefloat = hours \times 60.0 + minutes$. Then we can apply sine and cosine functions on time floats to locate them in the proper position on the clock surface. But here time floats are divided by $12 \times 60 = 720$ minutes because we incorporate hours into minutes. Hereto for every time float, we have two coordinates. Based on previous testing results, we keep the structure of the CNN model as in Table 2 with another fully connected layer added with 100 neurons, and the output layers are replaced by Dense of 1 with linear activation. The loss function is mean squared errors for both coordinates. After running 55 epochs with learning rate of 0.001, Figure 3 shows the predicted time points and true time points on the clock surface. The predicted time points are well positioned as a circle, which is in the same shape of the true time points. We plot some random time points on both of the circles, which are represented by blue stars. We did not check the time beforehand, after plotting them on the graph, we can easily recognize the time point on the clock surface. Furthermore, the randomly chosen time points(blue stars) are on the similar positions on both of the plots. Our model and predictions seem to perform quite well.

When transferring all time points back to radian, we use $\arctan 2$ function from *Numpy*. This arc tangent function can preserve time points' quadrant correctly, and its outputs are radians of time



(a) Predicted time points (b) True time points

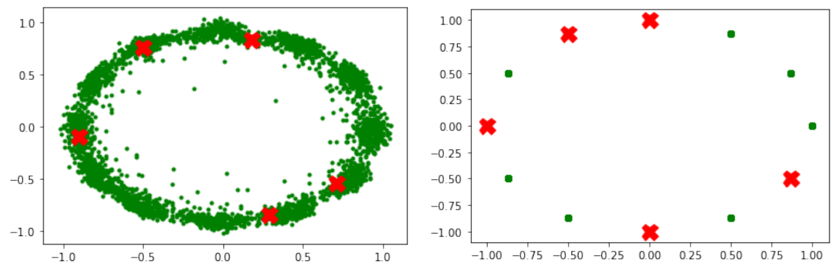
Figure 3: Time predictions

points. The formula to transfer predicted values back to degrees in counter clockwise are as follows:

$$\begin{aligned} hour_{degrees} &= \arctan 2(hour_y, hour_x) \cdot 180/\pi \\ minute_{degrees} &= \arctan 2(minute_y, minute_x) \cdot 180/\pi \\ timefloat_{degrees} &= \arctan 2(sin_y, cosine_x) \cdot 180/\pi \end{aligned}$$

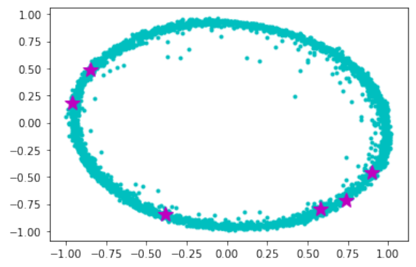
The final step is to transfer degrees back to time points. Since there are 60 minutes and clock circle has 360 degrees, each degree shares $60/360 = 0.16666$ minutes. Applying this formula, the error in minutes between true and predicted time is 1.7273 minutes.

One tiny problem of this kind relabeling is that we can only recognize minutes on the clock surface. If we want both hours and minutes, we could calculate sine and cosine values for both hours and minutes, in this way, we could expect 4 outputs from the CNN model, keeping all other parameters unchanged. According to Figure 4 and Figure 5, we can easily recognize hours or minutes from the graphs.



(a) Predicted hours (b) True hours

Figure 4: Hours predictions


(a) Predicted minutes

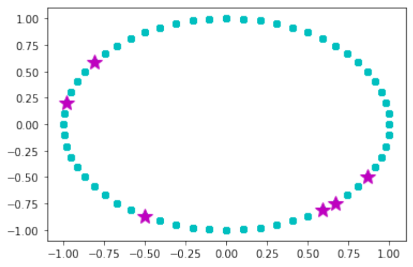

(b) True minutes

Figure 5: Minutes predictions

1.5 Discussions and Conclusions

1.5.1 Task 1 and 2

For the convolutional neural network, removing the dropout layers improved the accuracy by a small margin. This was also observed when we increased the kernel size. However, with the latter, the computation time increased significantly. Coming to hyperparameters, due to the large number of model combinations, we have not listed them in this report. However, our main findings are:

1. Setting the learning rate differently influences the model by a significant amount. Therefore, it is essential to pay attention to the way the learning rate is determined. We follow (Géron, 2017) to do so, where we start with a large value that makes the training algorithm diverge, then divide that value by 3, and repeat the process until the training algorithm stops diverging. Another option, apart from using *learning schedules* [1] would be to use Adam optimization technique because it is an adaptive learning rate algorithm (like AdaGrad and RMSProp). Using an early stopping routine would have more advantages over changing training iterations as well.
2. During batch normalization, a very small value of the batch size would give less precise values of the gradients. For both, the CNN and MLP models, reducing the batch size also made the test accuracy worse.
3. Since we do not observe the significantly different results with obfuscation of data for CNNs, but MLPs perform worse, given the accuracies we obtain, it could be influenced by the parameters we have chosen in our models, or the choice of hyperparameters as well. For the latter, we can use Keras Hyperparameter Tuner to sift through the different hyperparameter combinations in a more random yet accurate manner, thus reducing computation time and improving efficiency.
4. Based on our observations, f-MNIST is harder to classify compared to MNIST data, and CNNs perform much better than MLPs. However the latter is tricky, because CNNs take a much longer time to train the models, so an optimum choice of parameters with sufficient hyperparameter tuning could be a good trade-off choice.

1.5.2 Task 3

Going through this task and experimenting with various methods of training with convolution neural networks, we found out that relabeling method generates the best results with the lowest mean absolute errors of 1.7273 minutes. During training, in order to improve the accuracy, we tried a lot different parameters and their combinations. For instance, He uniform initialization works better than He normal initialization for Relu activation function. In addition, data augmentations and random rotations did not seem to help improving the prediction accuracy, but with enormous amount of training time. Furthermore, adding one hidden layer or leaving one hidden layer out could have massive impacts on the testing accuracy. The same situation also applies to epochs. Most of the time, running one more epoch could really jeopardize the results because of over-fitting. Nevertheless, due to limited

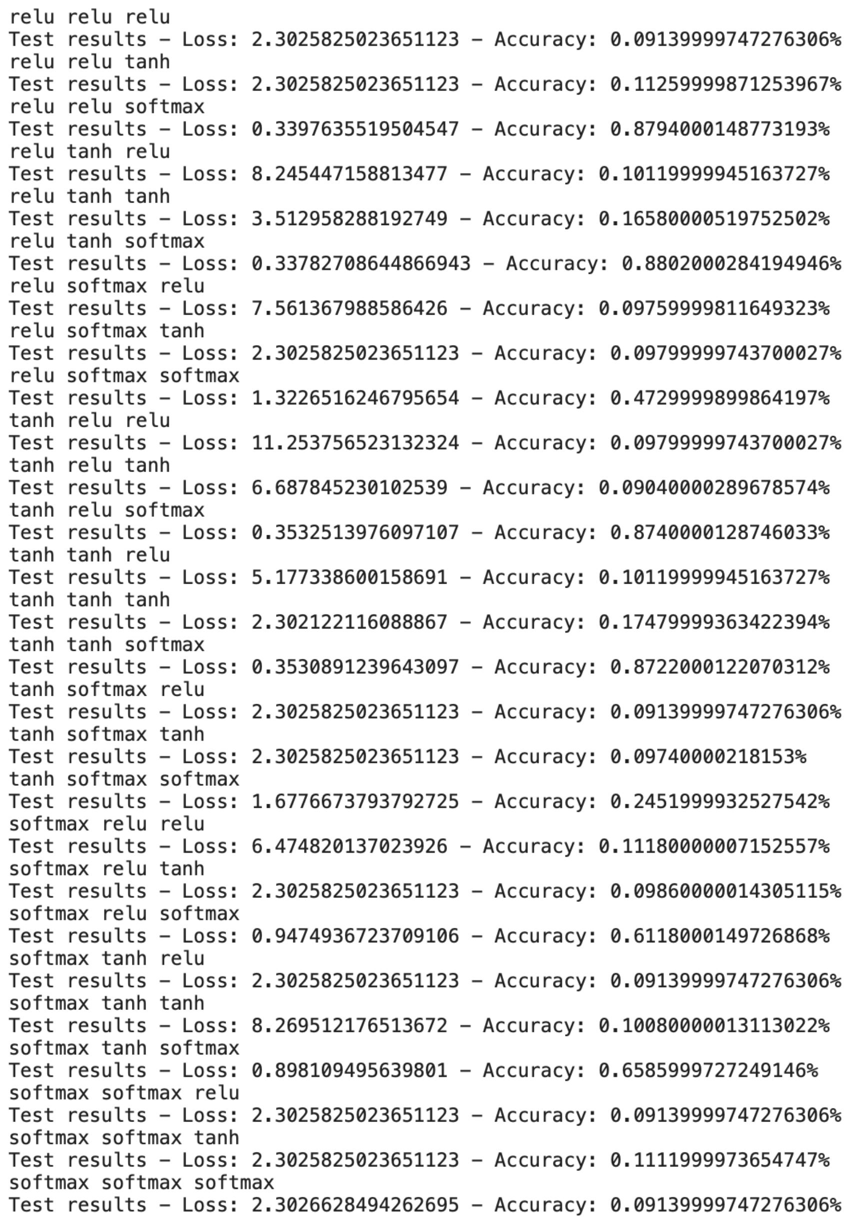
[1](See Pg. 353, Géron (2017) for the different types)

time and RAM, we couldn't try more possibilities with different types of activation functions and optimizer. Adam is the most used optimizer recently, and its advantage is fast converging, but it may also suffer from over-fitting issues. To this end, we would explore further in our future experiments and researches. In summary, multi-output model(97.06% accuracy) and relabeling time points(1.7273 MAE) are the best two performing models for "Telling-the-correct-time".

References

A. Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, CA, 2017. ISBN 978-1491962299.

2 Appendix



```
relu relu relu
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09139999747276306%
relu relu tanh
Test results - Loss: 2.3025825023651123 - Accuracy: 0.11259999871253967%
relu relu softmax
Test results - Loss: 0.3397635519504547 - Accuracy: 0.8794000148773193%
relu tanh relu
Test results - Loss: 8.245447158813477 - Accuracy: 0.10119999945163727%
relu tanh tanh
Test results - Loss: 3.512958288192749 - Accuracy: 0.16580000519752502%
relu tanh softmax
Test results - Loss: 0.33782708644866943 - Accuracy: 0.8802000284194946%
relu softmax relu
Test results - Loss: 7.561367988586426 - Accuracy: 0.09759999811649323%
relu softmax tanh
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09799999743700027%
relu softmax softmax
Test results - Loss: 1.3226516246795654 - Accuracy: 0.4729999899864197%
tanh relu relu
Test results - Loss: 11.253756523132324 - Accuracy: 0.09799999743700027%
tanh relu tanh
Test results - Loss: 6.687845230102539 - Accuracy: 0.09040000289678574%
tanh relu softmax
Test results - Loss: 0.3532513976097107 - Accuracy: 0.8740000128746033%
tanh tanh relu
Test results - Loss: 5.177338600158691 - Accuracy: 0.10119999945163727%
tanh tanh tanh
Test results - Loss: 2.302122116088867 - Accuracy: 0.17479999363422394%
tanh tanh softmax
Test results - Loss: 0.3530891239643097 - Accuracy: 0.8722000122070312%
tanh softmax relu
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09139999747276306%
tanh softmax tanh
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09740000218153%
tanh softmax softmax
Test results - Loss: 1.6776673793792725 - Accuracy: 0.2451999932527542%
softmax relu relu
Test results - Loss: 6.474820137023926 - Accuracy: 0.11180000007152557%
softmax relu tanh
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09860000014305115%
softmax relu softmax
Test results - Loss: 0.9474936723709106 - Accuracy: 0.6118000149726868%
softmax tanh relu
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09139999747276306%
softmax tanh tanh
Test results - Loss: 8.269512176513672 - Accuracy: 0.10080000013113022%
softmax tanh softmax
Test results - Loss: 0.898109495639801 - Accuracy: 0.6585999727249146%
softmax softmax relu
Test results - Loss: 2.3025825023651123 - Accuracy: 0.09139999747276306%
softmax softmax tanh
Test results - Loss: 2.3025825023651123 - Accuracy: 0.1111999973654747%
softmax softmax softmax
Test results - Loss: 2.3026628494262695 - Accuracy: 0.09139999747276306%
```

Figure 6: Results of the combination of different activation functions across the three layers of an MLP. We chose the combination of ReLu-ReLu-softmax or ReLu-tanh-softmax as the top choices on the basis of validation accuracy as shown in this image. We then chose softmax over tanh because the former outperforms the latter in most common scenarios (Géron, 2017).