Introduction

Neural networks are an incredibly effective technique for image classification and regression. The models can be used to identify various patterns in the data. This can be used for classification tasks, for example, where the network determines which category images belong to such as digits or objects. Secondly, it can be used for regression tasks where it is used to generate continuous values, for example, prognosis of price from features. Convolutional Neural Networks (CNNs) are best used with image data as it can learn spatial pyramids by computing various features (edges, texture, etc.) through convolutional layers.

Task 1

For this assignment we trained neural network models for image classification on the Fashion MNIST dataset, which has 60,000 training samples and 10,000 testing samples, each of which belongs to one of ten clothing categories. The objective was to create both a MLP and CNN model to assign images of clothing to their respective categories. We then tried to increase the model performances by tuning the parameters of these models. We looked at test accuracy and test loss as markers of model performance.

Data Preprocessing
We first defined the dataset loaded from TensorFlow's keras.datasets.fashion_mnist, and scaled the pixel values of the images to have a range of 0-1 by dividing by 255, which is easier for the model to train. The dataset was split into a training, validation, and test set with respectively 54000, 6000 and 10000 images.

Part A: Multi-Layer Perceptron (MLP) Implementation
To start, a model was chosen that was used in Géron (2019) on pages 298 to 302. The basic initial layer applies the Flatten layer for changing the 28 x 28 images to 1D configurations, then two hidden layers which are two dense layers of 300 and 100 neurons respectively. The model uses ReLU as an activation function. The output layer contains 10 neurons that, by using softmax activation function, produces probabilities of the 10 associated clothing categories. The model was run on the training set with a categorical cross-entropy loss function, stochastic gradient descent optimizer and accuracy to check the performance. Due to the computation time, the model was first trained for a fixed number of 30 epochs on the Fashion MNIST data. The accuracy and loss was computed for both the training and validation set respectively during and after each epoch.

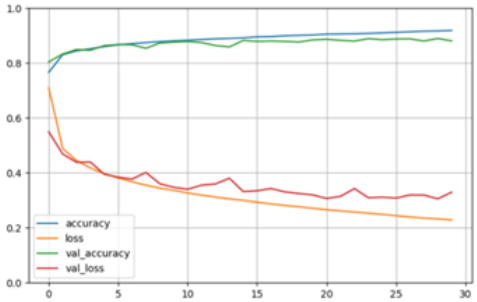

Figure 1: The accuracy and loss of the model depicted against each epoch of the model

The plot shows the behaviour of the accuracy and loss over the course of 30 epochs, which gives a notion about the model's learning curves. The accuracy of the model still seemed to be increasing whereas the loss also seemed to be decreasing. This indicates that a larger number of epochs could still better the models performance. By evaluating the model on the test set we found a test

loss of approximately equal to 0.36 and test accuracy of approximately 87.4% which means we succeed in the Fashion MNIST classification challenge sufficiently.

Parameter tuning and model changes

To try to increase model performance a series of experiments was conducted. For each experiment a single change was made to the original model mentioned before and tested to see if it improved model performance.

Firstly, the complexity of the model was increased by adding a third hidden layer to the initial neural network. The added layer was a dense layer with 200 units, placed in between the already existing hidden layers. The reasoning behind this was to increase the model complexity which provides more detailed representations of the underlying patterns. After training and testing this modified model yielded a test accuracy of 88.2% and a test loss of 0.33. The added layer therefore increased the model performance compared to the initial model.

Secondly, an experiment with Batch Normalisation layers was conducted. This was done to possibly enhance training steadiness and results. This improvement assists to mitigate internal covariate shift that causes faster convergence and potentially better performance. Doing this slightly improved the test accuracy to a figure of about 87.4%. However, the test loss was a little higher with 0.435. This indicated that although the test accuracy had improved by a very slight amount the models confidence or fit might have deteriorated. However, the fact that this brings the overall loss up slightly, coupled with a little improvement to the model performance, indicates that incorporating Batch Normalisation may not have been the best choice. It was beneficial for training stability but did not yield a direct improvement in this scenario.

Next, dropout layers with a rate of 50% were added after the hidden layers. Dropout layers help to minimise overfitting of a model. On our test set, adding the dropout layers resulted in a slight increase in the test accuracy and a small reduction in the test loss, hence a slightly improved generality.

Applying Leaky ReLU with alpha= 0.1 activation as a replacement for ReLU improved the test accuracy and decreased the test loss, therefore fixing a couple of problems which affected the ReLU activation-function, that is, dying ReLUs(dying ReLU is an issue where some neurons become permanently deactivate), and the decremental gradients. This modification improved the model's test accuracy to 88.1% and reduced test loss to 0.337.

As a next experiment, the learning rate of the SGD optimizer was manually set to 0.01. With this, a small improvement in test accuracy and loss was achieved. This was quite peculiar as the default value of the sgd optimizer should already be 0.01. This could be explained by the fact that manually setting a learning rate helps to avoid issues that may arise due to other parameters of the optimizer. This experiment was repeated with a learning rate of 0.001. This, However. performed worse on both accuracy and loss on the test set. This indicates that if the configuration is set up in this manner, a learning rate of 0.01 is the most appropriate because it provides a good compromise between stability and convergence speed.

Experiments on the epoch value were also done. The basic model was trained up to 40 epochs which reduced the loss (0.36). However, the test accuracy remained almost the same (0.88) as what was achieved in 30 epochs, which means that the model might have learnt at its maximum capacity around 30 epochs. That is why it can be assumed that increasing epochs brings incremental results and cannot guarantee significant improvement, and the level of overfitting will increase. To combat this an early stopping argument was included as a callback in the model fit. For the patience parameter a value of 7 was chosen. Lower patience values led to a tendency to stop the model before the optimal test performance was reached. With this the lowest validation loss was achieved at 36 epochs. This increased the test accuracy to 0.88 and decreased test loss to 0.32. This approach validates the techniques commonly used to deal with overfitting, most importantly, the technique of early stopping.

From these experiments the following adjustments to the basic model seemed to improve model performance on the test set: adding an additional hidden layer, adding a dropout layer, changing the activation function to Leaky ReLU, setting the sgd learning rate to 0.01 and setting an early stopping callback.

Creating the final model

As an attempt to create the best model until now, all these modifications were added together to the basic model. This resulted in a small augmentation of the performance on the test set compared to the baseline model. The test accuracy came to 87.8% and the test loss to 0.344 after 50 epochs. The adjustments therefore resulted in a better performing model than the basic model. However, looking at the earlier experiments, better performances were achieved before. The experiments that performed better on their own were the additional hidden layer, the LeakyReLU modification and the epoch tuning. One reason for this could be that the model just had not converged well yet after 50 epochs. The early stopping had not yet stopped the training. Therefore, the model was given some more time to train. After an additional 18 epochs, 68 in total, the model converged. The additional training time led to some enhancements in the results: an accuracy of 88.2% and loss of 0.332. This did not surpass the performance of the manual sgd learning rate and early stopping on its own. This result indicates that the advantages obtained from each alteration are likely not cumulative and some modification may have a progressively reconciling effect when applied together.

Next, just the early stopping and sgd learning rate were applied to the basic model. After converging at 49 epochs, this configuration tested at a lower accuracy (85.5%) and a higher test loss (0.409), than the previous model. This result indicates that, in fact, the creation of additional layers or application of a dropout solution might be required in order to enhance the result of the network on this task. Even when the model simplified, there was no harmony between the corresponding accuracy and loss. As this model performed worse than the other final model, the previous one was used in part C.

Part B: Convolutional Neural Network (CNN) Implementation

Like in part A, a model from Géron (2019) page 461 was chosen as an initial model. This model consists of an input layer, 12 hidden layers and an output layer. The input layer contains 64 filters of shape 7x7. After this comes a MaxPooling layer of 2. This is followed by 2 consecutive convolutional layers, both containing 128 filters of size 3x3. Another MaxPooling layer of size 2 is then used. Next there are two convolutional layers again, this time both with 256 filters of size 3x3. After this a last MaxPooling layer of 2 is applied. Then we come to the fully connected network which starts with a Flatten layer to flatten the data. This is followed two times by a dense layer (respectively of 128 and 64 units) and a dropout layer of 50%. Lastly the output layer is a dense layer with 10 nodes with softmax activation. For all the convolutional layers “relu” is used as an activation function and for padding “same” was used.

To benefit efficiency the train test and validation sets were converted to “float32”. The model was run on the training set with a sparse categorical cross-entropy loss function, SGD optimizer and accuracy to check the performance. For fitting a batch size of 64 was used. After 20 epochs an accuracy of 87.4% and a loss of 0.345 was reached on the test set. According to the source this model could even reach 92% accuracy on the test set. However, due to limitations in hardware this could not timely be reached for this report.

Like before, a series of experiments was conducted to try to improve model performance. For this part, however, a slightly different approach was taken. As mentioned the hardware available for this project made for long computation times. Therefore it was chosen to evaluate experimental models at 3 epochs only and compare them to the basic model performance on the validation set at 3 epochs (accuracy 0.805; loss 0.521). This could maybe have resulted in rejecting more optimal models that took longer to converge. However, it nonetheless gave insights.

The first experiment was to add an extra convolutional layer to the model. The layer was added after the two layers of 256 filters and size 3x3. This added layer had those same specifications. After 3 epochs the validation accuracy was 0.769 with a loss of 0.634. This therefore performed worse than the base model. A possible explanation for this could be that adding more layers increases the complexity of the model, therefore taking longer time to converge. It is possible that assessing the performance after just 3 epochs does not do the model justice.

Secondly, Leaky ReLU was used to replace “relu” as the activation function for both the convolutional and the dense layers. An alpha value of 0.1 was chosen. The alpha value controls the output value for any negative input. In this case the output is 10% of the input value, this helps avoid “dead neurons”, dead neurons refers to a neuron that stops updating and consistently outputs the

same value. After training the validation accuracy was 0.780 with a loss of 0.578. This once again was worse than the basic model. This is possibly due to the way Leaky ReLU handles negative values.
Thirdly, different values for the last dropout layer were tested. The value of the dropout was decreased to 25%. By reducing the dropout, the model maintains a balance between preventing overfitting and preserving enough information for effective learning. A dropout rate of 25% could work better because it introduces less randomness and allows more neurons to remain active during training. After training the validation accuracy was 0.786 with a loss of 0.552. Therefore the considered theory did not seem to hold.
As a change to the compilation function, changing the optimizer to Adadelta was tested.The key difference is that SGD uses the same learning rate for all updates, while Adadelta adjusts the learning rate for each weight based on recent changes, making it more adaptable and removing the need to set a learning rate manually. This resulted in a validation accuracy of 0.152 and a loss of 2.259 which is dramatically worse. This is probably due to the fact that adadelta takes longer to converge. Only training for 3 epochs therefore cannot truly evaluate its performance.
Lastly, the padding of the convolutional layers was changed from “same” to “valid”.with "same" padding, the convolutional layer adds extra rows and columns filled with zeros around the edges of the input data so that the output has the same dimensions as the input. "Valid" padding does not add any extra padding, therefore the output size is smaller. This is because the filter only processes areas that fully fit within the original input. As valid padding changes the shape of the data the second MaxPooling layer also had to be removed. This experiment yielded a validation accuracy of 0.754 and a loss of 0.660. This lesser performance could be due to the fact that without the zero padding of the “same” padding, the convolutional layer cannot capture features at the edges of the input as effectively. "Valid" padding reduces the size of the feature map with each layer. This could have resulted in the model being able to less accurately capture patterns around the edges.
In conclusion, none of the adjustments seemed to improve the performance of the base model. The model where the dropout value was lowered came out as second best. These models were therefore used in part C.

Part C: Performances on the CIFAR-10 dataset
The CIFAR-10 dataset is a collection of 60,000 RGD images of size 32x32x3 split into 10 different categories. These 10 categories are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The goal was to take the three best performing models from before and to apply them to this new dataset to see how the performances translate.
First the new dataset was pre-processed to align with input requirements for the models. The pixel values were normalised to the scale of 0 to 1 by division of the data by 255. The data was split into a train, test and validation set of sizes 45000, 10000 and 5000 respectively and converted to “float32” to aid efficiency.
As for the 3 models, the final MLP model was chosen as well as the best and second best CNN models. They were:

- The best combined MLP model
- The “basic” CNN model
- The CNN model with the 25% dropout value

The MLP model
The model converged after 93 epochs and yielded a test accuracy of 50.9% with a loss of 1.359. The model's performance did not seem to translate to the CIFAR10 dataset. On the Fashion MNIST dataset this model had an accuracy of 88.2% and loss of 0.332. This could be because of the fact that the CIFAR10 dataset is a lot more complex. The MLP model therefore most likely is not complex enough due to insufficient layers. Also the model was made with data with only 1 channel in mind, the model now flattens CIFAR10's images. This results in a loss of information on the arrangement of the pixels. This makes it very hard for the model to recognize patterns.
The “basic” CNN model

The model was evaluated after 24 epochs. By then the test accuracy was 62.5% and the loss 1.064. Compared to the MLP model performance this is a big improvement, however, the performance of this model on the fashion MNIST dataset was a test accuracy of 87.4% and a loss of 0.345 after just 20 epochs. A possible explanation is, again, that the model is not complex enough to deal with the multitude of channels.

The CNN model with the 25% dropout value

After 24 epochs this model scored an accuracy of 65.4% and a loss of 0.994 on the test set, making it the best performing model out of the three. A possible reason why this model performed better than the other CNN model could be because the dropout helps to avoid overfitting by randomly setting a percentage of neurons to zero during training, which forces the model to learn more robust and generalised patterns in the data.

Overall, the CNNs seemed to translate performance better than the MLP on the CIFAR-10 dataset. This is because they are built to pick up on image details like shapes and textures, keeping the layout of pixels intact, which helps with complex images. The MLP, on the other hand, flattens everything out, losing the pixel structure, so it struggles to spot the patterns that make CIFAR-10 images recognizable.

Task 2

Creating a CNN that can accurately predict the time displayed on pictures of analog clocks is the goal of the "Tell-the-time" assignment. Experimenting with various label representations, including regression, multi-class classification, multi-head models, and periodic functions, is the main goal. The aim is to determine which method minimizes the "common sense" error, which is the absolute time difference between the predicted and actual times.

Data preprocessing

After loading, random shuffling was applied to the images and labels to ensure balanced distribution across training, validation, and test sets. The data was split into a 80/20% split for respectively training and testing. From the training set, a small subset of 500 samples was used for validation. The images were standardised by dividing the pixel intensity by 255.0 to scale them into the range of 0 to 1. This was done to enable the model to achieve quick convergence. Every image was preprocessed to contain one grayscale channel, agreeing with the CNN input format specification. Finally, the time values were transformed to multi-class targets using either 24-class or 1-minute labelling depending on the task. Then these labels were encoded with LabelEncoder and transformed into one-hot format as required for categorical cross-entropy loss in multi-class classification.

Classification model

The classification task involves categorising each clock image into discrete time intervals, initially with broader ranges and then progressing to finer divisions. This approach aims to simplify the time-telling problem by treating it as a multi-class problem rather than a continuous prediction. First, the Convolutional Neural Network (CNN) was designed for the classification task. The input shape was set to match the images (150x150 pixels with a single grayscale channel). The model architecture was structured with several convolutional and pooling layers, followed by fully connected (dense) layers to complete the structure.

The model started with convolutional layers with 32 filters and kernel sizes of both 5 x 5 and 3 x 3. Every convolutional layer was then preceded by ReLU or Rectified Linear Unit activation function that brought non linearity into play to enable learning of intricacy. To understand them better the activity maps were furthermore reduced by dimensionality via MaxPooling layers, thus lowering the overall computational cost and thereby discouraging overfitting.

After flattening of the output obtained from the convolutional layers multiple fully connected dense layers were incorporated. These layers helped in enabling the network to learn with the help of features extracted by the convolutional layers. Different layers of units were used as 1024, 512, 256, 128, 64, and 32 layers were utilised for formulating multiple layers to reduce the dimensions while enabling feature extraction and decision processes.

The CNN model had a number of Convolutional layers for spatial learning and fully connected layers for decision making. fully connected part there are several dense layers to factor the data before getting to the final layer of 24 neurons and softmax activation to enable classifying through 24 classes. The model was trained using the Adam optimizer with learning rate set to 0.0001. Categorical cross entropy was used as the loss function with accuracy used as the main metric. An original 'common sense error' time-related function was designed to reduce cyclic temporal errors, together with employing the early stopping technique to stop training when validation loss failed to decrease further. This sort of setup made it easy to learn the model and at the same time assess its performance on other new data.

24 v/s 720 labels:

We first used only 24 classes, at a half-hour granularity. All the times between $hour{:}\,00$ and $hour{:}\,29$ were grouped into the class $2 * hour$, and the rest were grouped into class $2 * (hour + 0.5)$. The model performed well, since it was easy to generalise for intervals that contained 30 different time labels.

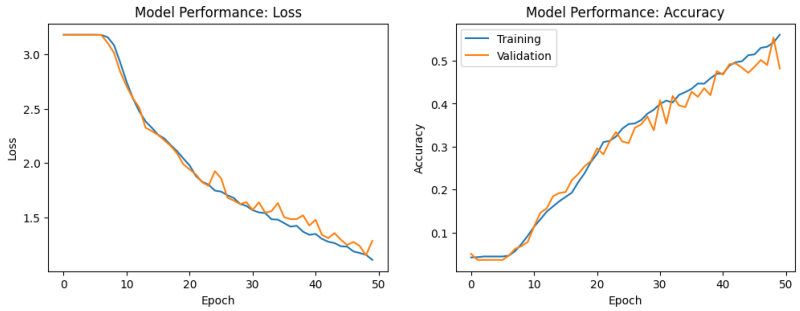

Figure 2(a) : Training and Validation Loss of the Classification Model (24 Labels) (75x75)

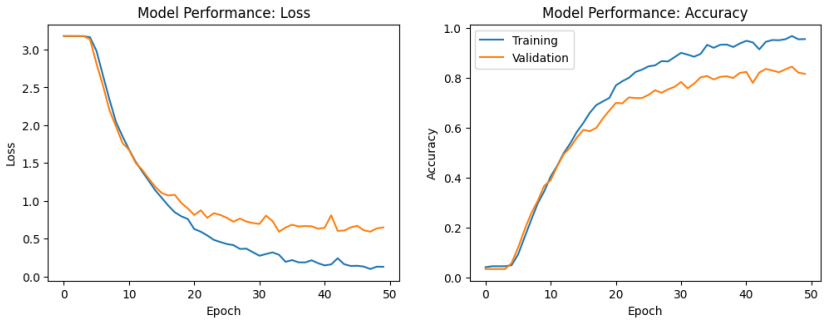

Figure 2(b) : Training and Validation Loss of the Classification Model (24 Labels) (150x150)

However, once we switched to 720 labels (i.e.) minute-level granularity, the model performance dropped completely. We expected this, since there is very little difference between two distinct classes, and the model cannot generalise for each class properly due to the variation in the images (glares and rotations). Additionally, simple Dense CNNs will struggle with such a high number of classes, unless we provide a greater number of layers.

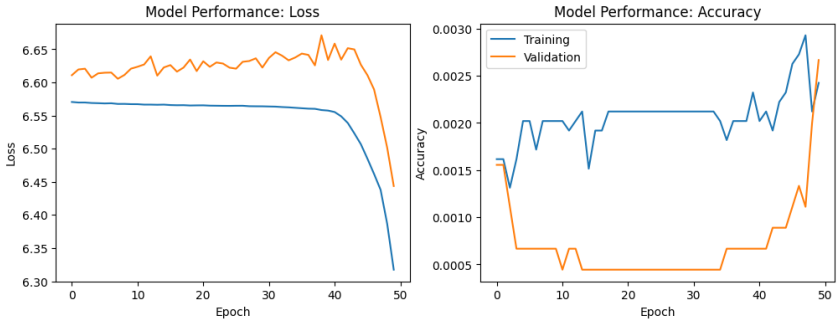

Figure 3: Training and Validation Loss of the Classification Model (720 Labels)

Strangely, once we changed the dataset splits so that the validation set was 15% of the training set (overlap), the model training accuracy skyrocketed from 0.3% to 60%. We believe this is simple overfitting of the data, however what we cannot explain is why the testing data accuracy also showed a similar increase- 0.2% to ~51%.

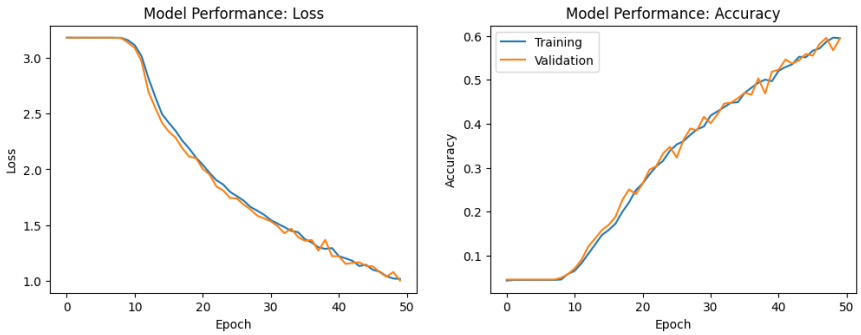

Figure 4: Training and Validation Loss of the Classification Model (720 Labels) after dataset change (75x75)

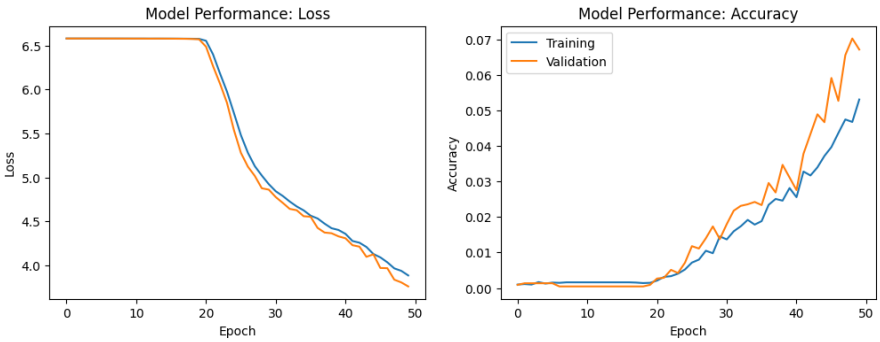

Figure 5: Training and Validation Loss of the Classification Model (720 Labels) after dataset change (150x150)

The gap between training and validation showed that the model was slightly overfitted to the training data as the epochs progressed. Although the training metrics continue to improve, the validation metrics stop improving and, in the case of validation loss, fluctuate, suggesting instability. We used regularisation, dropout layers and also early stopping of the model fitting. The model showed less overfitting however, no change was observed in the results of the model.

The model was then used to predict the classes on the test set, and the highest confidence label was picked and compared to the actual label. We defined accuracy as the number of instances which have been correctly classified by comparing the predicted labels and the actual labels. We also calculated the average common sense time difference (CSTD) between the true label and the

predicted label. The common sense time difference is calculated by the following formula: $min(pred - true, pred + num_{labels} - true)$

	75x75 images		150x150 images	
No. of Labels	**Accuracy**	**Avg. CSTD**	**Accuracy**	**Avg. CSTD**
24	51.47%	Between 2.5 & 3h	83.08%	Between 0.5 & 1h
720	51.75 %	2.17 minutes	1.72%	162.2 minutes

Table 1: Comparison of Classification Model performance with 24 and 720 labels

Regression model
The purpose of the regression model was to forecast time as a continuous quantity. In order to express time in fractional hours, the time labels were first combined into a single continuous variable by dividing the minute values by 60 and adding it to the hour value. As a result of this the model was able to handle time as a continuous target instead of a multi-class classification problem.

Model Architecture:
With a single neuron output layer and a "softplus" activation function to predict the continuous time value, a Convolutional Neural Network (CNN) was constructed utilising layers resembling those in the classification model. Instead of using fully connected (dense) layers for deeper feature learning, the CNN used several convolutional layers with ReLU activation and pooling layers to capture spatial data.

Mean Absolute Error (MAE) and a "common sense error" function were two customised metrics used to track the model after it was improved using the Adam optimizer and compiled using the Mean Squared Error (MSE) loss function. The common sense function compares the true time with the predicted time in both 12 and 24 hour formats and returns the smaller difference. In order to account for the cyclical structure of time, the custom common sense error metric computes the smaller difference between predicted and true times. In order to avoid overfitting, training was stopped early if validation loss did not improve after ten epochs.

Following training, the test set was predicted, and the average "common sense error," which showed the model's accuracy in time prediction, was computed. The prediction accuracy was visually represented by a histogram plot of the common sense error distribution, and the learning curves demonstrated steady drops in training and validation loss throughout epochs, indicating successful model learning.

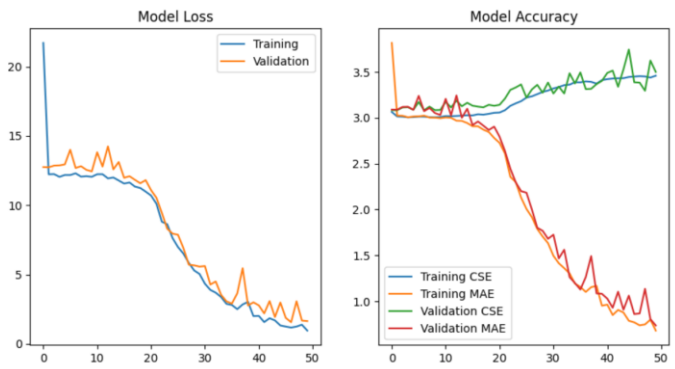

Figure 5: Regression Model Training Performance

The regression model's training and validation performance over 50 epochs is shown to be effective in learning and avoiding overfitting. The model's accuracy metrics, including common sense error and mean absolute error, show a gradual decrease in errors, indicating accurate estimation of time on unseen data. According to the final average common sense error of 0 hours and roughly 44.7 minutes. This suggests that the model was able to accurately simulate the real times seen in the pictures.

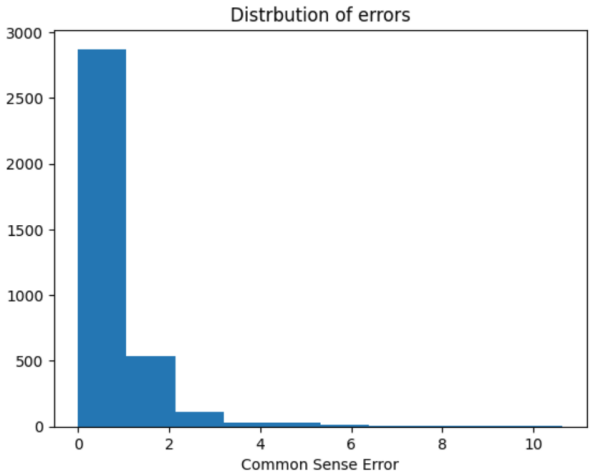

Figure 6: Distribution of CSE on predicted times via the regression model

The histogram shows the model's common sense error distribution, with most predictions close to zero, indicating accuracy. A large concentration around lower errors reflects the model's effectiveness, with an average common sense error of 44.7 minutes.

Two-Headed model

We use the Two-Headed model to divide our problem into two subproblems- classification for the hour value and regression for the minute value, which is essentially a conjoining of the two models above. First, the labels, hours and minutes have been split and minutes are divided by 60 in order to make continuous regression. The "common sense" error functions (common_sense_hour and common_sense_minute) were defined to calculate the minimum error, accounting for the circular nature of hours and minutes.

The model takes in grayscale clock images and starts with several convolutional and max-pooling layers to extract features from the images. This is followed by a flattening layer to prepare the features for the fully connected (dense) layers. After feature extraction, the flattened output passes through a series of dense layers, shared by both heads of the model. Two output heads were created from the final dense layers: one head (hour) is a classification model with a softmax layer that outputs 12 heads, each representing a number from 1 to 12 on the clock the second head, minute was a regression model which only had one neuron in the output using a softplus function with the output being the number of minutes as a regularised value between 0 and 1.

The model was compiled with two losses: sparse categorical cross-entropy losses for the hourly predictions and MSE for the minute regression. There were two types of measures: accuracy, and mean average error (MAE). Validation loss was tracked to decide when to stop earlier in order to avoid overfitting, and the model is trained with both split data sets again for 30 iterations. The training accuracy and loss of each epoch are recorded for evaluating the convergence and training of the model for both the sigmoid and softmax outputs.

Our accuracy metric was calculating the common sense difference between the predicted and the actual time. After a 50-epoch train, the model was able to bring the average common sense error down to under 15 minutes (~14.95-14.99 mins). In fact, the hour head of the model hit and

plateaued at 85% accuracy. The minute-head, just like the regression model, struggled to hit an accuracy above 60%.

Figure 7: Training performance of Two-Headed model (75x75)

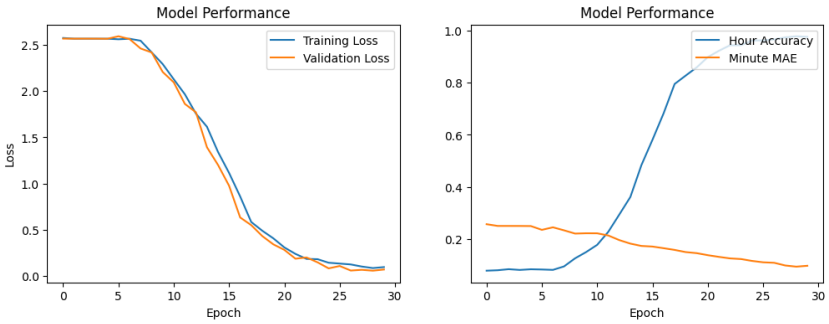

Figure 7: Training performance of Two-Headed model (150x150)

However, for 150x150 images, minute loss did see a significant drop during training, however the average testing CSE was 15.1 minutes, suggesting overfitting of the model to training data. For this, we believe that adding noise to the images will be a better solution than modifying the model architecture, since we believe the current setup is well optimized for the general task.

Sources

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Unsupervised learning techniques*.

Contributions

Saghar: write report
Rajat: Task 2
Hannah: Task 1, report task 1