Introduction to Deep Learning
Assignment A1

Chloé Tap, Evan Meltz, Giulia Rivetti (Group 36)

November 2023

Contributions

Chloé Tap: Task 1 code and Task 2a, 2c code & report
Evan Meltz: Task 1 code and Task 3 code & report
Giulia Rivetti: Task 1 code & report and Task 2b code & report

Introduction

For this assignment, using Keras/Tensorflow, we intend to experiment with the practical applications of deep neural networks and build various models for classification, regression as well as generative tasks on image datasets. In doing this, we hope to gain proficiency in using Keras and TensorFlow APIs, refine our manual hyperparameter tuning skills and then use this experience to develop a CNN to tackle the "tell-the-time problem". Lastly we will train generative models of an image dataset of cat faces and experiment to see how well we can generate new ones.

1 Learn the basics of Keras API for TensorFlow

In this section we developed MLP and CNN networks and trained them on two different datasets: *Fashion MNIST* containing grayscale images of clothing articles and *CIFAR-10* containing RGB coloured images representing animals and objects. The aim was to classify images in these datasets in a particular category. In particular, the purpose of this part is to experiment with the sample models provided on github by changing the different parameters characterizing the network's architecture and observe how these changes impact the final performance of our model. More specifically we used different:

- initializations: Glorot Uniform, Random Normal, Random Uniform, Zero;
- activations: ReLU, sigmoid, softmax, tanh;
- optimizers: SGD, Adam, AdamW, Adamax;
- optimizers' learning rate values: 0.1, 0.01, 0.001;
- regularizers: L1, L2, dropout, no dropout;
- adding and removing hidden layers;
- adding and removing number of convolutional filters and changing their size.

We have experimented with these options in the given order and for each of them we have selected the one giving the best performance.

1.1 Training with Fashion MNIST data

For both the MLP and CNN models, we started with the code provided in the github repository and we adapted it to our task. For the MLP model, after loading the dataset and splitting it into training and test set, we reshaped and scaled both train and test set for the images. Then we defined six different architectures for the model, in order to find the architecture leading to the best performance. In particular, we added or removed dense and dropout layers, as well as changed the number of nodes (n) of the dense layers.
Regarding the CNN, the model takes as input images of 28x28 pixels; we normalized these images in the range [0, 1]. In this case we experimented with five different architectures. We experimented with maxpooling and dropout layers, adding and modifying the number of filters (f) of the convolutional layers and adding and changing the number of neurons for the dense layers (n).
Table 1 reports the best parameters that we have found for the MLP and the CNN on the Fashion MNIST dataset, whereas Table 2 shows the architectures for both models that have yielded to the best accuracy: 0.895 for the MLP and 0.930 for the CNN.

	MLP	CNN
Initialization	Glorot Uniform	Glorot Uniform
Activation	Sigmoid	Tanh
Optimizer	Adamax (lr = 0.01)	AdamW (lr = 0.001)
Regularization	Dropout (r = 0.1)	Dropout(r = 0.1)

Table 1: Parameters yielding to the best performance for MLP and CNN architectures

Layer type	Parameters
Dense	n = 512
Dropout	r = 0.2
Dense	n = 512
Dropout	r = 0.2
Dense	n = 10

(a) Best architecture for the MLP on Fashion MNIST

Layer type	Parameters
Convolutional	f = 32, k = (3, 3)
Convolutional	f = 64, k = (3, 3)
MaxPooling	ps = (2, 2)
Dropout	r = 0.1
Flatten	
Dense	n = 128
Dropout	r = 0.1
Dense	n = 64
Dropout	r = 0.1
Dense	n = 10

(b) Best architecture for the CNN on Fashion MNIST

Table 2: Best architecture for MLP and CNN on the Fashion MNIST dataset

1.2 Applying to CIFAR10 data

After trying different parameters for both the MLP and CNN models on the Fashion MNIST dataset, we have considered the best three configurations and used them to train an MLP and a CNN this time on the CIFAR-10 dataset.
For the MLP the best three configurations that we have obtained are:

1. Glorot uniform initialization, sigmoid activation function, Adamax optimizer with a learning rate of 0.01, two dropout layers with rate of 0.1.

2. Glorot Uniform initialization, sigmoid activation function, Adam with learning rate of 0.001 as optimizer and the addition of two dropout layers with a rate of 0.2.

3. Glorot uniform initialization, ReLu activation function, AdamW optimizer with learning rate equal to 0.001, one dropout layer with rate 0f 0.2

The three configurations yielding to the best performance in the case of the CNN are the following:

1. Glorot uniform initialization, ReLu activation function, Adam optimizer with leraning rate equal to 0.001 and dropout layers with rate 0.2.

2. Glorot uniform initialization, ReLu activation function, AdamW optimizer with a learning rate of 0.001 and dropout layers with rate of 0.2.

3. Glorot uniform initialization, tanh activation function, AdamW optimizer with a learning rate of 0.001 and dropout layers with rate of 0.1.

As for the architectures, we have used the ones illustrated in Table 2, the one on the left for the MLP and the one on the right for the CNN.
The results that we have obtained for the MLP are summarized in Table 3. From the table we can see that we have obtained a much lower accuracy with respect to the one that we got for the *Fashion MNIST* dataaset for all the three different configurations. This big difference in the performance is not surprising. Indeed, the *CIFAR-10* dataset has coloured images, each one of size 32x32. Instead, the *Fashion MNIST dataset* has grayscale images of dimension 28x28 and it is therefore less complex and detailed with respect to the other dataset. Moreover, the images of *CIFAR-10* are more varied, since they represent different objects and animals, making the dataset more complex with respect to *Fashion MNIST*, whose images represent clothing items.

Configuration	Accuracy
1	0.505
2	0.502
3	0.422

(a) Accuracy for MLP model

Configuration	Accuracy
1	0.7008
2	0.7007
3	0.7001

(b) Accuracy for CNN model

Table 3: Accuracy obtained on the CIFAR-10 dataset with the three configurations specified before

From Table 3 we can also see the results for the CNN model on *CIFAR-10*. We can notice a higher accuracy with respect to the MLP model, due to the fact that CNNs are more suitable for image classification tasks. This is due to the fact that CNNs take into account the dimensional information of an image while MLPs do not. However, comparing these results with the ones obtained for the *Fashion MNIST* dataset, we can also see that the accuracy is lower in this case due to the higher complexity of the dataset, as explained before.

2 Develop a "Tell-the-time" network.

In this task, our goal was to develop several CNN models to predict the times from grayscale clock images in various orientations. The labels corresponding to each image in the data provided were automatically given in ascending order from 0:00 to 11:59. Before feeding the data into the networks, we shuffled the data indices to randomize the order of labels for a more appropriate model to be trained. We considered three different methods for performing this task: classification, regression and multi-head models. In the following sections, we summarize our procedure and results for each method used and compare them using appropriate metrics as well as evaluating the "common sense accuracy" for each method.

2.1 Classification model

Here we apply a N-class classification problem in which we aim is to classify images by dividing them into subclasses corresponding to a given interval Δt. For example, we initially started with 30 minute intervals resulting in 24 classes: class 0 from 0:00 to 0:29, class 1 from 0:30 to 0:59 and so on. Using a dataframe, we stored the hour and minute labels of the images and their corresponding class label (0-23). These new labels were provided in the CNN network.
We tested various architectures of the CNN classification model, starting from that provided in Task 1. Initially this architecture gave a test accuracy of only 1-2% for interval classes of 30 seconds. We reorganised the convolutional layers and added batch normalization and max pooling layers to improve the model accuracy. The final

Layer type	**Parameters**
Convolutional	f = 32, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Convolutional	f = 64, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Flatten	
Dense	n = 128
Dense	n = 64
Dense	n = number of classes

Table 4: Classification model architecture

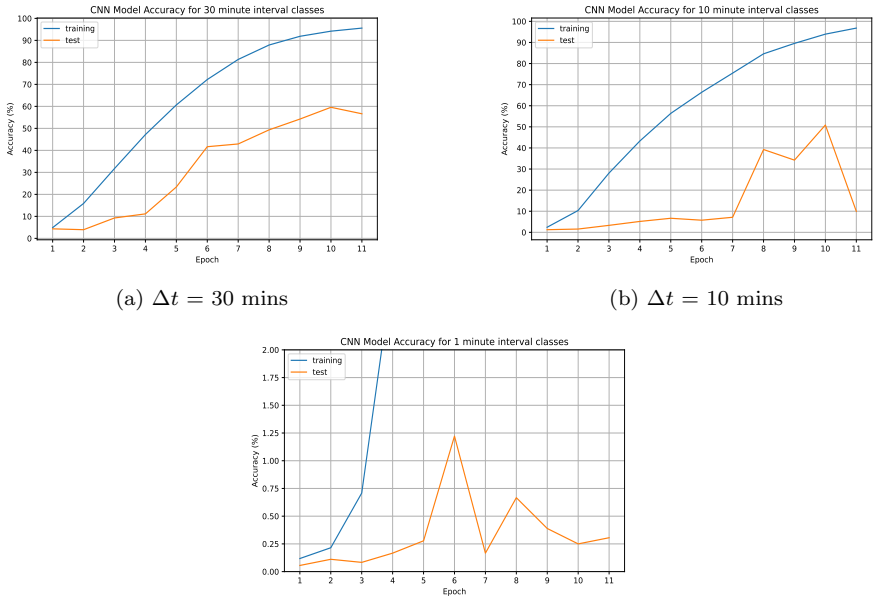
CNN model used is shown in Figure 4.
The loss function used here was categorical cross-entropy which is commonly used for classification problems. This is also referred to as "softmax loss" as it applies the softmax function as well as an additional cross-entropy loss function. This is particularly useful for multi-label classification as it evaluates the probability of an image corresponding to a particular class and propagates this through the network to reduce the difference in the true and predicted labels. Naturally the metric used for this model is the accuracy, i.e. percentage of correctly classified images. This was measured for each epoch but we wished to also evaluate the common sense accuracy as mentioned previously. The common sense accuracy CSA was determined by calculating the time difference between the true and predicted classes (c_{true} and $c_{predicted}$ respectively). This can be expressed mathematically as:

$$CSA = |c_{true} - c_{predicted}|\Delta t \tag{1}$$

We evaluated the mean CSA for all test data as our final CSA measure for the model. Note we did not compute this within our model as a metric, we attempted to do so but given the expression for CSA as given above it is not a great measure to introduce. We instead focused on the classification accuracy as our metric. Once we achieved a reasonable accuracy with this architecture, we proceeded to increase the time interval and thus the number of classes. We tested this for time intervals (in minutes) of 30, 10 and 1. Table 5 summarizes the train and test accuracy evaluated for each of the various class intervals (1, 10 and 30 minutes), as well as the common sense accuracy.

Interval Δt (mins)	**Test accuracy (%)**	**Common sense accuracy (mins)**
30	56.6	41.5
10	9.94	119.9
1	0.31	161.4

Table 5: Results of CNN classification model for various time interval classes Δt



(a) $\Delta t = 30$ mins

(b) $\Delta t = 10$ mins

(c) $\Delta t = 1$ min

Figure 1: Percentage training and test accuracy of classification CNN for time intervals Δt = (a) 30, (b) 10 and (c) 1 minutes as a function of training epoch. For $\Delta t = 1$ minute, the y-axis was scaled down to display the test accuracy variation clearly

As shown in Figure 1, the test accuracy varies significantly across different time intervals chosen. For $\Delta t = 30$ in Figure 1a, test accuracy increases relatively steadily with a small drop at the end of training at epoch 11. On the other hand, in Figures 1b and 1c, test accuracy tends to converge to a constant low accuracy at early epochs and then randomly oscillate to higher values this at later epochs before dropping to this steady value once again. The main reason for this is that the model cannot tell whether it is proceeding in the

correct "direction" to minimise the loss function, and thus determine the true labels accurately. This causes some epochs to appear to reach a "better" accuracy but then drop significantly soon after. Given the various rotations and orientations of the clock images, it is difficult for such a model to discern the attributes for each class effectively. In addition, the common sense accuracy is not a great measure in this problem as for large time intervals, the time difference is only in multiples of such an interval (not accurate to the nearest minute unless all 720 classes are considered). Thus it is evident that a classification regime alone is not suitable for such a task.

2.2 Regression model

In this part, we built a CNN that predicts the time using a regression model. Regression models predict a continuous value based on the input variables. Their main goal is to estimate a mapping function based on the input and output variables. For this case, the categorical labels of hours and minutes are transformed to a single continuous value; for example the corresponding output of time"3:00" should be 3.0; for time "4:30" the corresponding output should be 4.5, and so on. For this model, we have used the architecture shown in Table 6. At first we have tried to used a simpler architecture, with only two sets of convolutional and maxpooling layer, without adding batch normalization. However, in this way we were only able to achieve a CSA of about 80 minutes, so we decided to add more layers.

Layer type	Parameters
Convolutional	f = 32, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Convolutional	f = 64, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Convolutional	f = 128, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Convolutional	f = 256, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Flatten	
Dense	n = 128
Dense	n = 1

Table 6: Regression model architecture

Loss functions often used for a regression model are MSE (Mean Squared Error) and MAE (Mean Absolute Error). In our case, we decided to use the MAE because we believe it's better suited for our task with respect to the MSE. Indeed, images can sometimes contain outliers, such as noisy pixels or extreme values. MAE is less sensitive to outliers compared to MSE because it takes the absolute differences between predicted and actual values. This makes MAE a more robust choice when dealing with image data, where individual pixel values can vary significantly.
However, using the MAE as loss function also comes with some problems. With the MAE, large errors coming from the outliers end up being weighted the exact same as lower errors. This might results in the model being great most of the time, but making a few very poor predictions every so-often.

Run	**MAE**	**Common sense accuracy (mins)**
1	0.151	9.14
2	0.170	10.19
3	0.129	7.75

Table 7: MAE and Common sense accuracy for the CNN regression model on the train set

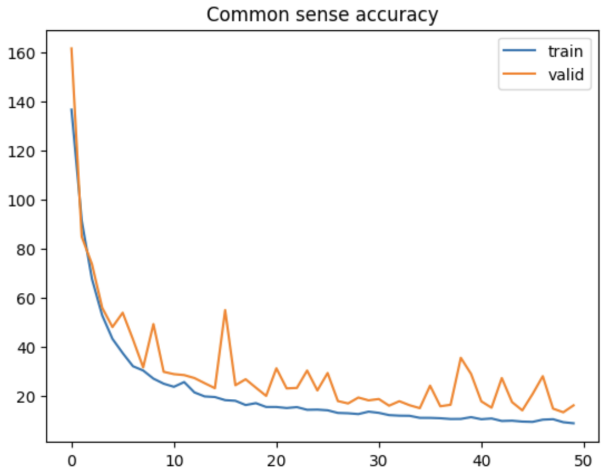


Figure 2: Plot of CSA versus number of epochs for train and validation sets

As for the metrics, we have employed the common sense accuracy, that we have computed by taking the absolute value of the difference between the continuous values of the predicted and target times. The we

multiplied this value by 60, in order to visualize at each epoch the common sense accuracy expressed in minutes. The results that we have obtained are summarized in Table 7. We have run the model three times, each one using 50 epochs and obtained consistent values for both the MAE and the common sense accuracy, which on average was of about 9 minutes on the train set resulting in the model being pretty accurate in telling the time. We have obtained slightly higher CSA values on the test and validation set, which were around 15 minutes. In particular, Figure 2 shows the behaviour of the CSA for 50 epochs on the train and validation set. We can observe that the graph for the validation set is more unstable with respect to the train graph, which was expected since the validation set is used to evaluate the model on new data. In any case, as we pointed out before, also for the validation set the overall CSA is around 15 minutes, so on average it is still accurate.

2.3 Multi-head model

A multi-head model consists of determining multiple outputs for a single dataset. In our case, we use a classification head for predicting the hours and a regression head to determine the minutes. In this way, we hope to achieve a better prediction accuracy by dividing our network into completing two individual tasks for which the model is best suited. The loss functions and metrics used for this output heads are the same as described for each of the classification and regression models we evaluated in Sections 3.1 & 3.2 respectively. We also determined the common sense accuracy by extracting the predicted hours from the categorical array representations, combined this with the predicted minutes then evaluated the absolute time difference (smallest value relative to 720 minutes or 12 hours) between true and predicted times.

Layer type	**Parameters**
Input: Convolutional	f = 32, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Convolutional	f = 64, k = (3, 3)
Max Pooling	ps = (2, 2)
Batch Normalization	
Flatten	
Dense	n = 128
Output (hours): Dense	n = 12
Output (mins): Dense	n = 1

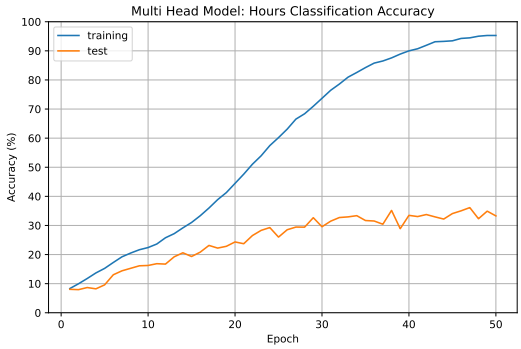
Table 8: Multi-head model architecture

Unlike in Section 2.2, we did not manage to implement the CSA as a metric for our multi-head model. We attempted to do so but the process was very time consuming and we were unsure how to implement such a metric using keras when there are two output heads. In the end, we decided to focus on ensuring that our model could run efficiently and correctly instead. We experimented with various model configurations, in particular adjusting the number of convolutional, max pooling and batch normalization layer sets as used in the previous models in this task. However we noticed that the inclusion of more than two such sets causes the model to undergo drastic overfitting, with highly variable results after only a few training epochs. This replicates the high variability shown in Figure 1 for our classification model. We tried to introduce dropout layers to rectify this but this was not able to significantly reduce this overfitting. So we decided to limit our model to only two sets of these layers as shown in Table 8. Although we did not achieve the greatest accuracy, we were able to produce consistent results over a period of 50 epochs. The results of three such runs of our multi-head model on the test set are given in Table 9.

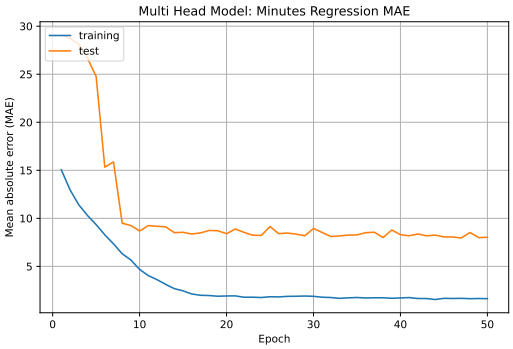
Run	**Hour accuracy (%)**	**Minutes MAE**	**Common sense accuracy (mins)**
1	33.28	8.028	81.53
2	34.08	7.535	86.15
3	38.28	8.178	72.45

Table 9: Accuracy, MAE and common sense accuracy results for various runs of our multi-head model on test data

Shown in Figure 3 is the comparison of accuracy and MAE across of training and test data for all 50 epochs for one such run of our model. Our models for three runs displayed in Table 9 converge to a test hour accuracy between 30-40% and mean absolute error (MAE) for minutes between 7.5-8.5. In addition, we found an average final CSA between 70-90 minutes. If we had been able to implement a CSA metric as in the regression model, we likely would have achieved a value of under 10 mins for CSA. Additionally we could have added additional layers to our model, however as discussed this may lead to extreme overfitting. Despite these low accuracies, we were able to successfully replicate our results across multiple runs for similar final hour accuracies and MAEs for our test set.



(a) Accuracy for hour classification as a function of epoch



(b) Mean absolute error (MAE) for minutes regression as a function of epoch

Figure 3: Metrics for each model head across 50 epochs: (a) Hour classification accuracy (b) Minutes regression MAE

2.4 Models comparison

After having built the three models (classification, regression and multi-head) for our "tell-the-time" network, we can compare them and draw some conclusion on their behaviors and performance. With these three models, we have experimented different ways of representing the labels as well as different architectures.
In the classification model, we divided the labels into a certain number of classes, each corresponding to a fixed time. In this case, we have used a fully connected output layer with a number of neurons equal to the number of classes. As for the regression model, we have converted the categorical labels of hours and minutes into continuous values and in this case we have used an output layer with only one neuron. In the multi-head model we have used a hybrid approach: a classification head for predicting the hours and a regression head for predicting the minutes. Therefore, our architecture's output layer has only one neuron for the minutes, whereas it has 12 neurons for the hours, one for each class.
We have computed the CSA for all the three models, but used it as a metric only in the regression case, due the higher complexity of the other two models. So, for the classification and multi-head models, we have computed the final value for the CSA, which has given us a good idea of the model's performance at the end of training. We have tried to minimize the CSA for the three models and by comparing the results that we have discussed before, we can observe that the best result was reached by the regression model, which yielded to a CSA of only 9 minutes on average for the train set, and 15 minutes for both test and validation sets. As already mentioned the regression model outputs a continuous value, which is more ideal for our task of predicting the time, allowing us to achieve more accurate results. A 40 minutes common sense accuracy was obtained with the classification model using time intervals of 30 minutes and therefore with 24 classes. As already mentioned, in this case we have discretized time into classes, which have resulted in a loss of precision, and hence a higher CSA. The lowest accuracy and therefore highest value for the CSA was found in the case of the multi-head model. We attributed this to the higher complexity of this model, requiring a more careful parameter tuning: our multi-head model has indeed different loss functions as well as different architectures for hours and minutes, resulting in a more complex structure.

3 Generative Models

In this task, we have trained three networks, Convolutional Autoencoder (CAE), Variational Autoencoder (VAE) and Generative Adversarial Network (GAN) on a dataset of our choice. The goal of this part was to train these networks and use the models to generate new images similar to the ones fed as input.

3.1 Dataset

Our first attempt was with a Pokemon image dataset which contained 819 images of various Pokemon of different shapes and sizes. We extracted and pre-processed the images from a zip file and downsized the images to 64x64. However, after running it with the VAE and GAN we could not make anything out. We then normalized it and, after tuning the hyperparameters, noticed improvements but could only make out shapes. Having attempted with various configurations and getting no improvements, we realised the problem

was the high complexity of the images and the vast differences in shape size and colour between them. For this reason, we opted for another dataset with more uniformity, this time an image dataset of cats with shape 512x512. Once again after extracting, downsizing, and tweaking hyper parameters, we saw no good results being produced, even though the uniformity did result in more noticeable images than the Pokemon dataset. However in this case it was due to the sample size being too small. For this reason, we finally settled on a bigger cat dataset containing around 15,000 cat images of shape 64x64 [1]. We still downsized the images further to 32x32, as per the assignment instructions, and this time noticed the best results seen thus far. This is likely due to the fact that there were a lot more images to train the model on. Although trading the quality of the images for the quantity did have a noticeable impact on the quality of the images generated, it was a worthwhile trade-off resulting in the generator creating images that looked similar to the training images. Figure 4 below shows the images from the final dataset chosen after being extracted and pre-processed.

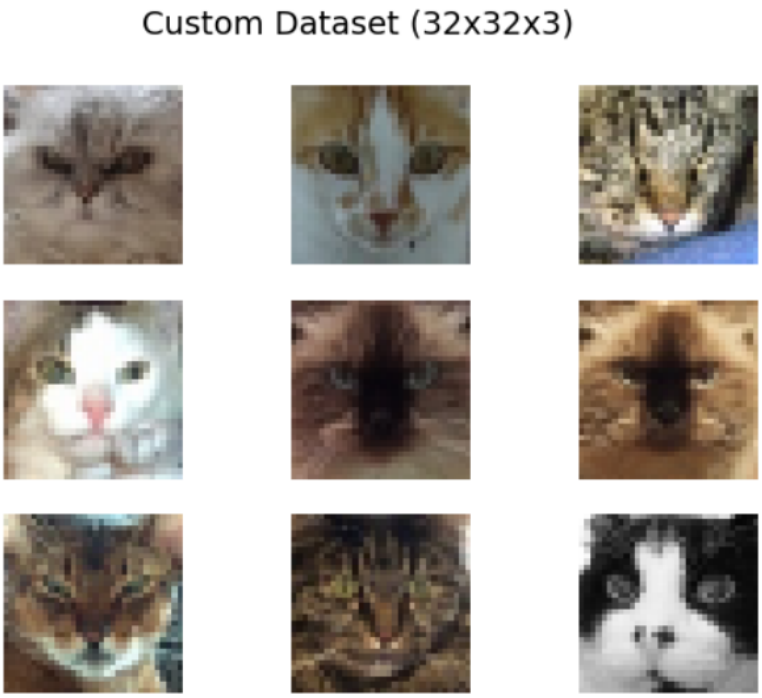


Figure 4: Custom dataset reduced to (32x32x3)

3.2 Convolutional Autoencoder (CAE)

A Convolutional Autoencoder (CAE) works by processing and generating data with a grid-like structure, such as images. It utilizes convolutional layers, pooling layers, and transposed convolutions (deconvolutions) in its architecture.
The CAE includes an encoder for downsampling the input data to extract essential features and a decoder for upsampling using a latent vector to reconstruct data. While its primary purpose is data compression and reconstruction, CAEs are also generative in nature. They show how convolutional and deconvolutional networks can be used together to reconstruct images to a high accuracy which we can see in Figure 5.

3.3 Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) also incorporates an encoder and decoder, like CAE, but its focus is on learning a probabilistic latent representation of the input data. VAE introduces stochasticity by sampling from the learned distribution to generate a latent representation. This stochastic element encourages the generation of diverse and continuous data.
Compared to CAE, which primarily focuses on data structure preservation, VAE's main goal is probabilistic generative modeling. It aims to generate data points similar to, but not identical to, the input, making it a powerful tool for various generative tasks. As we can see in Figure 5 below, while the results are not as

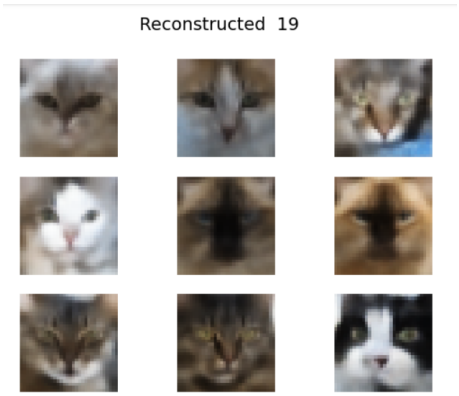


Figure 5: CAE reconstructed images after 20 epochs

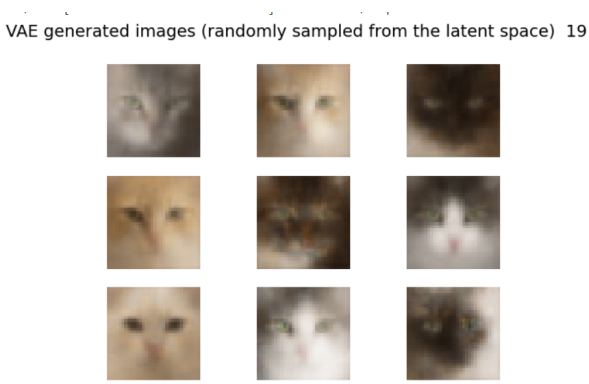


Figure 6: VAE generated images after 20 epochs

clear and don't replicate the original images as well as CAE, this is expected as mentioned the idea is to reconstruct the original images but to sample from the latent space to generate new ones.

3.4 Generational Adversarial Network (GAN)

Instead of using an encoder and decoder as in the previous two models, Generative adversarial networks (GAN) use a generative model and a discriminative model. The discriminator model works as a classifier and must determine if a given image is real (i.e. comes from the dataset) or if it was artificially created by the generator model. On the other hand, the generator model begins with vectors of random numbers (noise) as input values and creates images through a deconvolutional neural network.
Throughout training, whilst the weights and biases in the discriminator and the generator are trained, the discriminator will attempt to classify the real images from the generated ones and will give the generator feedback to produce images that are more similar to the real ones in an attempt to make the discriminator unable to identify which images are real and which are generated. They are essentially in competition with one another and this adversarial training process is what makes GANs unique and powerful. As seen below in Figure 7, the images are more diverse and unique when compared to the other two models. However this also results in some strange and abstract images which do not accurately reflect the initial images that the network was trained on.

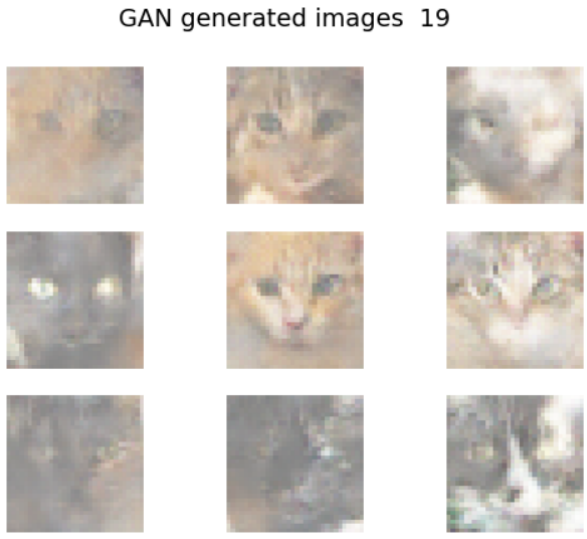


Figure 7: GAN generated images after 20 epochs

Concluding remarks

In summary, the main purpose of this assignment was to learn how to use Keras and TensorFlow APIs, which was done by completing three different tasks. First, we have used MLP and CNN architectures on two different datasets (*Fashion MNIST* and *CIFAR-10*), to experiment with their architectures and parameters, and understand their effects on the performance of our networks. We have observed how the complexity of a dataset can affect the performance of our architectures and concluded that the CNN are better suited for larger and more complex datasets such as *CIFAR-10*.
Then, with the knowledge gained from the first task, we have developed a CNN network able to "tell the time" from grayscale clock images. This was done by using three different models: classification, regression and multi-head. We then compared these models to have a better understanding of the working and performance of each model in the case of our "tell-the-time" network. We have observed that in our case the minimum CSA was reached with the regression model, which, by outputting a continuous value, was better suited than the other two models.
Finally, we have trained and used three generative models (CAE, VAE and GAN) for image generation. For this purpose we have chosen a dataset containing images of cats' faces and trained those networks to get new images representing cats. Also in this case, we have compared the results to better understand how each network works. With all those three networks we were able to get new images of cats, even with a low resolution since we scaled down the input images to 32x32.

References

[1] Kaggle. Cats faces 64x64 (for generative models). `https://www.kaggle.com/datasets/spandan2/cats-faces-64x64-for-generative-models?rvi=1`, February 2019.