Assignment 2

Filippo Boni (s3628590) s3628590@vuw.leidenuniv.nl
Joël During (s3558339) s3558339@vuw.leidenuniv.nl
Bram van Kooten (s3738574) s3738574@vuw.leidenuniv.nl

Task 1: Learn the basics of Keras API for TensorFlow

Task Definition

In this task, we will familiarize ourselves with the Keras API for TensorFlow using chapters 10 and 14 of *Geron's textbook* [1] and the documentation on https://keras.io/. We will apply this knowledge to the Fashion MNIST dataset, which is a drop-in replacement for the standard MNIST dataset containing more complicated classes of clothing articles [2]. We will experiment with different MLP and CNN architectures and hyperparameters on this dataset. Then, we will apply our best models to the CIFAR-10 dataset, which contains a more diverse set of classes [3].

Methods

For these experiments, we use the MLP and CNN as presented in chapters 10 and 14 of *Geron's textbook* as a starting point [1].

The default MLP used for this task contains 784 input neurons, and two hidden layers of 512 neurons with the ReLU activation function [4] are used. After each hidden layer, a dropout layer is used with a dropout rate of 20%. The output layer consists of 10 neurons (one for each class) and utilizes the softmax activation function. All layers are fully connected.

For the CNN, padding is never used, and the ReLU activation function [4] is used unless otherwise specified. The default network contains the following layers:

- An input layer of size 28x28.
- A convolutional layer with 64 kernels of size 7x7 and a stride of 1
- A max pooling layer with a 2x2 kernel and a stride of 2.
- Two convolutional layers, each containing 128 kernels of size 3x3 and using a stride of one.
- Another 2x2 max pooling layer with a stride of 2.
- Two more convolutional layers, this time containing 256 kernels of size 3x3, again using a stride of 1.
- A last 2x2 max pooling layer with a stride of 2.
- Two fully-connected layers with 128 and 64 neurons, respectively. A dropout layer is placed after each fully-connected layer with a dropout rate of 20%.
- An output layer with 10 neurons, using the softmax activation function.

Both networks are trained using a sparse categorical cross-entropy loss using Stochastic Gradient Descent with a learning rate of 0.01 and a batch size of 32. The MLP is trained for 50 epochs, and the CNN for 20.

Experimental Results

We set up a series of experiments to investigate how changing parameters, optimizers, or the architectures of these networks influence their performance. In each experiment, different versions of the MLP and CNN will be evaluated on the MNIST data set. The networks are evaluated over five runs to combat randomness, and the results are averaged.

Experiment 1: weight initialization, activation functions, regularization methods, and batch normalization

The results of the first experiment, which tries different weight initialization strategies, activation functions, regularization methods, and batch normalization, can be seen for the MLP in Figure 1. Unless otherwise specified, the default parameters mentioned above are used. For the MNIST classification task, the weight initialization strategy does not influence the MLP network's performance on the training or validation set. On the other hand, the activation function has a large influence. The ReLU activation function performs

best on the training and validation set. The hyperbolic tangent performs similarly but slightly worse, and the Sigmoid activation function performs the worst of these three.

Figure 1 also shows that of all the regularization methods, only dropout might lead to a small increase in performance on the validation set. L1 and L2 regularization only decreases performance. Batch normalization reduces the difference between the performance on the training and validation sets but does not influence the performance on the validation set.

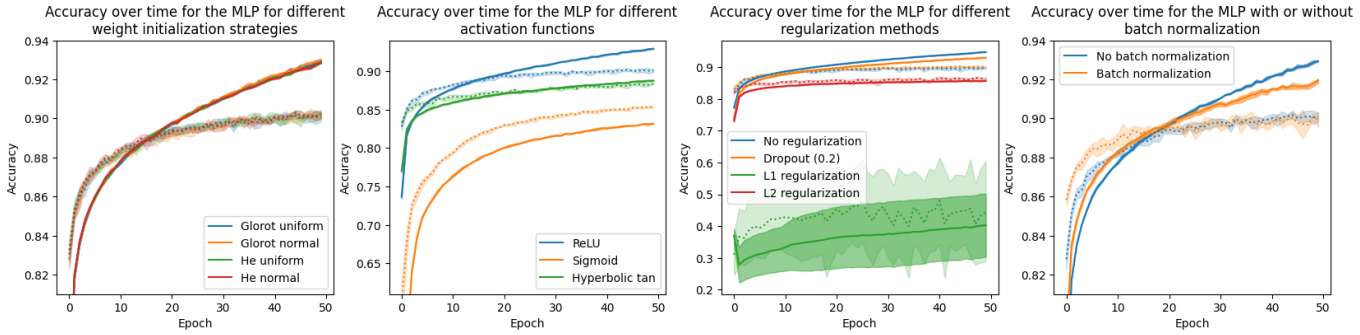


Figure 1: Accuracy over time for different versions of the MLP network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

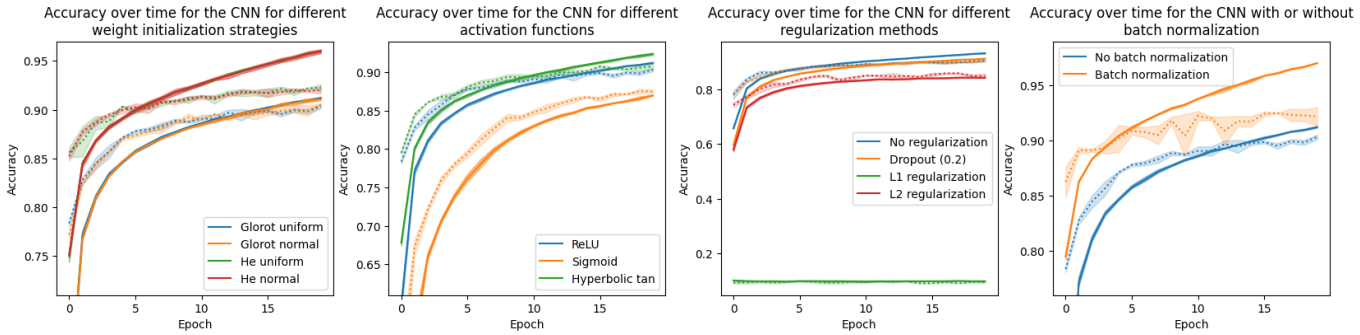


Figure 2: Accuracy over time for different versions of the CNN network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

Figure 2 shows the results of the first experiment for the CNN. We can see that for the CNN, the He weight initialization performs better than the Glorot weight initialization on the training and validation set. For the activation function used in the fully connected part of the CNN, we see that the hyperbolic tangent performs slightly better than ReLU. The sigmoid activation function performs the worst. The regularization methods perform similarly to the MLP, with the network with or without dropout performing better than the network with L1 or L2 regularization. We can also see that, for the CNN, batch normalization after the convolutional layers increase the performance on both the training and validation set. However, batch normalization does increase the variance of the performance on the validation set.

Experiment 2: Network architecture

A second experiment investigates the influence of the network architecture on the network's performance. This experiment's results for the MLP can be seen in Figure 3. The figure shows that decreasing the number of hidden layers of the MLP to one decreases the performance of this network. However, increasing it to three does not increase the performance (on the validation set. On the other hand, increasing the number of neurons per layer of the MLP increases the performance on the training set but does not influence the performance on the validation set.

Figure 4 shows the results of the second experiment for the CNN. We can see that the architecture of the fully-connected part of the network does not have a big influence on the performance of the network on

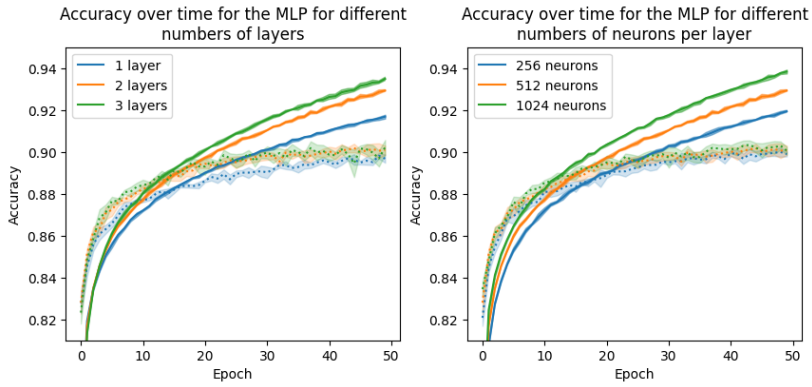


Figure 3: Accuracy over time for different architectures of the MLP network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

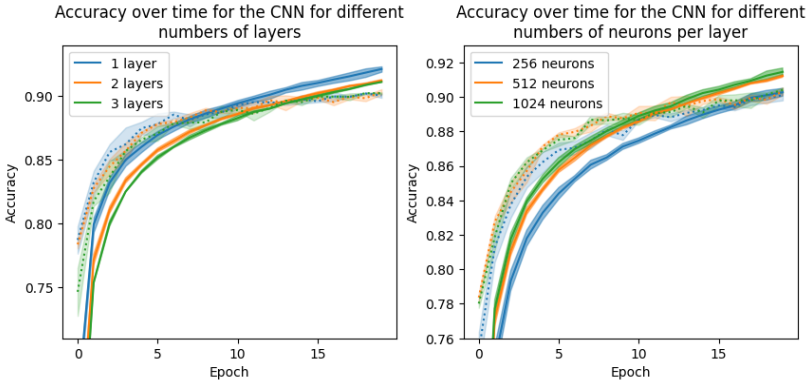


Figure 4: Accuracy over time for different architectures of the CNN network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

the validation set. However, a more complex network with more layers or nodes does seem to learn slightly faster.

Experiment 3: Optimizers and learning rates

A third experiment investigates the performance of different optimizers and different learning rates. This experiment's results for the MLP can be seen in figure 5. The figure shows that for the SGD and Adagrad optimizers, a higher learning rate produces a higher accuracy on the training set. However, for both optimizers, the performance on the validation set does not increase for a learning rate higher than 0.005. The Adam optimizer performs worse than the other two for this network configuration. For this optimizer, the best result is achieved with the lowest learning rate tried. Therefore, the performance might be better for an even lower learning rate.

Figure 6 shows the results of the third experiment for the CNN. We can see that again, for the SGD and Adagrad optimizers, a higher learning rate seems to increase the performance. For the Adam optimizer, a lower learning rate produces better performance, and a higher learning rate can reduce the performance drastically. With a learning rate of 0.001, the Adam optimizer outperforms the other two with their best-found learning rates.

Best models

The best models found on the fashion MNIST classification task are the following:

- *MLP1*: The default MLP
- *MLP2*: The default MLP with the Adagrad optimizer, with a learning rate of 0.01.

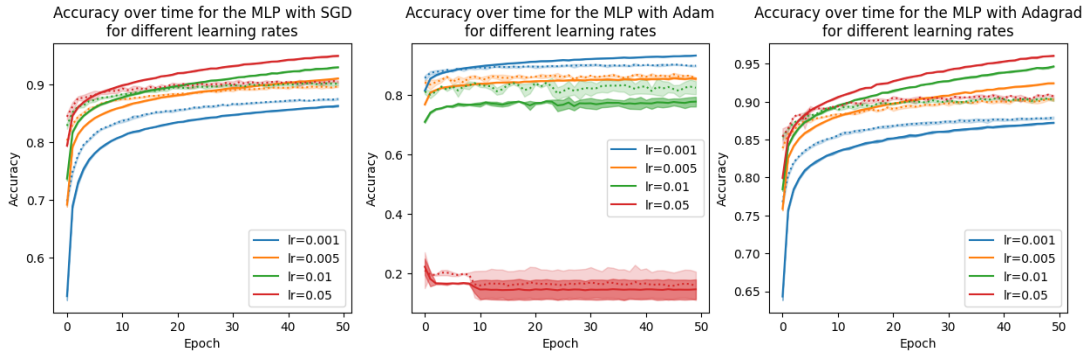


Figure 5: Accuracy over time for different optimizers and learning rates of the MLP network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

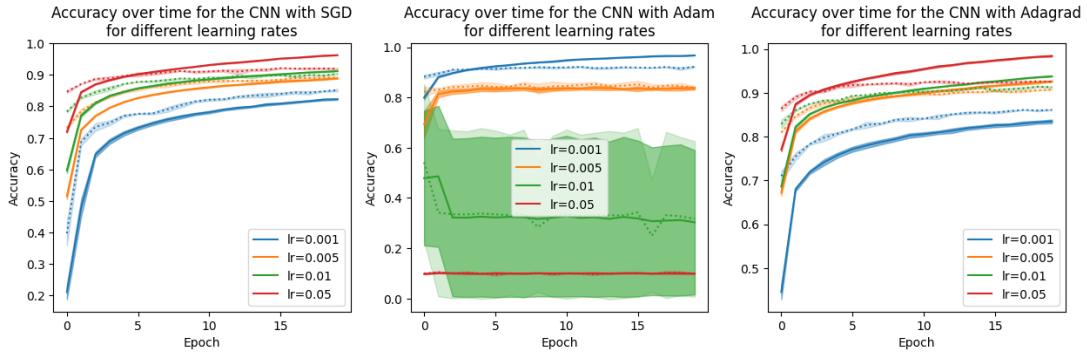
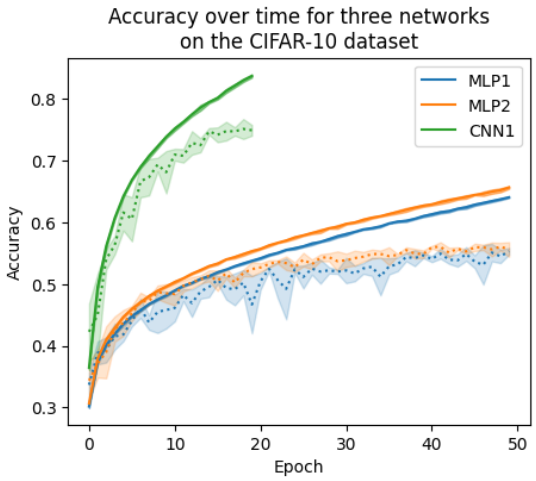


Figure 6: Accuracy over time for different optimizers and learning rates of the CNN network on the MNIST classification task. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the validation set.

- *CNN1*: The default CNN with He uniform weight initialization, the hyperbolic tangent activation function in the fully connected part, batch normalization after convolutional layers, and the Adam optimizer with a learning rate of 0.001.

We evaluate these three models on the test set. The results of this evaluation can be seen in Table 1.



	Mean accuracy	**Stddev**
MLP1	89.2%	0.2%
MLP2	90.0%	0.1%
CNN1	91.4%	0.5%

Figure 7 & Table 1: *Left*: Accuracy over time for the three best-found networks on the CIFAR-10 dataset. Solid lines show average performance on the training set over five runs with its standard deviation, and dotted lines show the same on the test set. *Right*: Accuracy on the fashion MNIST classification task test set for the three best-found networks.

Performance on the CIFAR-10 dataset.

We apply our three best-found models to the CIFAR-10 dataset to evaluate whether our performance gains translate to a new task. Figure 7 shows the results of this comparison. We can see that the two MLP networks compare comparably. However, where they reached 90% accuracy on the fashion MNIST classification task, they now only reach 50%. We can see that the CNN performs much better, reaching an accuracy on the validation set of almost 75 %. Even though this is less than the accuracy achieved on the fashion MNIST classification task, the improved performance over the MLP networks shows that the performance of the CNN translates better to more complicated tasks.

Conclusion and Discussion

We have performed three experiments to compare the performance of various MLP and CNN networks on the fashion MNIST classification task. We have found an MLP architecture that performs well, reaching almost 90% accuracy on the test set. We have also found a CNN which performs slightly better, reaching an accuracy of over 90%. We have applied our best-found network to a new dataset, with more variance among its classes. We have found that all of our networks perform worse on this dataset. However, the CNN performs much better than the MLP networks, reaching an accuracy of almost 75% on the test set. From this result, we suspect that CNNs are more suitable for more complicated image classification tasks than MLP networks. This result is not surprising since image classification is one of the main applications of CNNs.

Task 2: Develop a "Tell-the-time" network

Task Definition

We create a Convolutional Neural Network (CNN) for this task to solve the "Tell-the-time" problem. In this problem, images of analog clocks are provided in different orientations. The goal of the network is to correctly establish the time shown on the clock. For this assignment, we look at four different models for the CNN. These are regression, classification, multi-head, and finally, label transformation.

Methods

This section will discuss our approach for the different network models to solve the "Tell-the-time" problem. The four different models have the network we created for task one as a basis since this seemed to work well. The main difference between the network of task one is that the input size is different. Because of this, we have also decided to increase the stride in the early layers of the network. Different changes that were made for the specific models will be discussed below. In order to account for randomness in the data, the plots shown are taken from an average of three runs.

Regression

The dataset labels are converted to a single floating-point number to model the problem as a regression task. For instance, 3:00 will be converted to $y = 3.0$, and 5:45 will be converted to $y = 5.75$. The number after the decimal place represents the fraction of the full hour that has passed. With this representation, our CNN will have to output a single digit. Because of this, our output layer will only consist of 1 neuron, which uses a linear activation function instead of the softmax function that was used in the network from task 1.

When working with this type of model, the problem arises that the same amount of movement of the clock in its hour hand changes the final output of the network a lot more than the movement of the minute hand. Because of this, the performance of the regression model is often off by a full hour. This is also reflected in the plots shown in Figure 8, as both the training and validation set seem to converge to around the 60-minute mark.

Classification

The different possible times are split into sections of a given interval to model the problem as a classification task. For instance, if an interval of 30 minutes is chosen, 3:00 to 3:30 would be put in the same category.

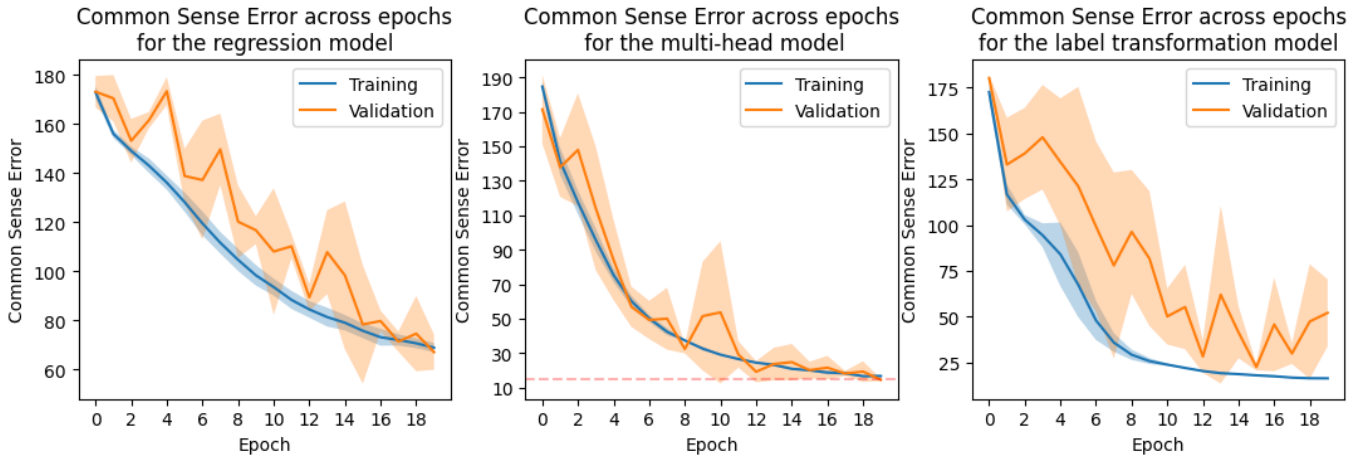


Figure 8: Results on the training and validation set from running the regression, multi-head, and label transformation models.

The main obstacle when dealing with this task as a classification problem is that within the given interval, there is still an inherent error in the given time. When we look at the interval of 30 minutes, for instance, we expect that, on average, the time would be off by half of the interval, so 15 minutes. This inherent error can only be mitigated by making the interval as small as possible. However, this takes a big hit in the performance of how well our algorithm actually learns the problem. This can clearly be seen in figure 9, where the common sense error remains very high even after many epochs for both the training and the validation set. When looking at the training set (figure 9), we can also see that for the 1-minute interval, the standard deviation starts to grow over time while the other intervals have converged.

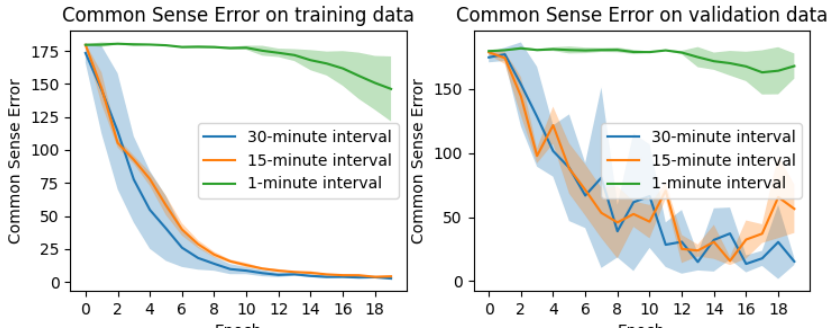


Figure 9: Results of the classification model on the training and validation sets.

Multi-head models

In order to try and mitigate the issues created by the two previous methods, we use a multi-head model to split up the result of the hours and minutes. Here the hours are represented as a classification problem, while the minutes are represented as a regression problem. Because of this, the network needs two different outputs. This is reflected in our network by branching off right before, where we would switch to the fully-connected layers leading to the output layer. Instead, the two branches from this point reflect what that part of the network would be in the regression and classification models.

The results from the multi-head algorithm can be seen in figure 8. Here we can clearly see that both the training and validation set converge around a common sense error of 15 minutes (a dotted line was added at 15 minutes for clarity). The validation set also seems to better follow the direction of the training set, unlike the two previous models, which are more frantic. The quality of this model can be understood from the mean test common sense error: 14,195 minutes.

Label transformation

Finally, we transformed the labels to represent the unit circle's angles. This was done by scaling the hour and minute labels to the range $[0, 2\pi]$. These angles were then transformed into sine and cosine values, which had to be learned by the network. In this strategy, the network is not concerned with predicting the actual times but merely with finding the angles of the small and large arms of the clock. Because sine and cosine values always lie in the range $[-1, 1]$, the ideal activation function to use is the hyperbolic tangent (which results in the same range). For our network, we used the same CNN as for the previous tasks but with a hyperbolic tangent activation function in the fully-connected part. The output layer consists of 4 nodes to predict the sine and cosine values of the angles of the small and large arms.

The results of this label transformation network can be seen in figure 8. We can see that the common sense error on the training set converges to around 20 minutes, while the common sense error on the validation set never gets lower than 30 minutes.

Experimental Results

Next, we compare the different CNN models' results to see which performs best. In figure 10, we can see that on the training set, the classification model reaches the lowest common sense error. However, when looking at the validation set (figure 10) we see that the classification model performs way worse, signifying that the model is over-fitted. Regarding common sense error, the multi-head and label transformation model performs similarly on the training set, just slightly worse than the classification model. The multi-head model performs the best on the validation set, meaning that this model is the best suited for dealing with the Tell-the-Time problem. The label transformation model does not perform better than the multi-head model. Perhaps its performance could be improved by only applying the label transformation to the minutes and using a multi-head classification for the hours.

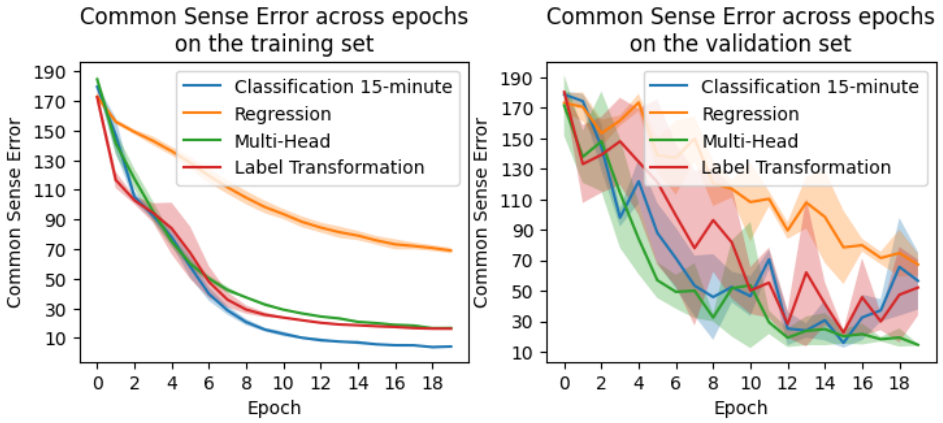


Figure 10: Comparison of the different CNN models on the training and validation sets.

Conclusion and Discussion

We have evaluated four different CNN models on the "tell-the-time" task. We have seen that a classification network can reach a reasonable accuracy at classifying images to be within an interval of 15 or 30 minutes. However, this approach does not extend to intervals as small as 1 minute. We also saw that a regression network struggles to predict the hours, resulting in a high common sense error. We set out to combine the best of these networks in a multi-head approach that uses classification for the hours and regression for the minutes. This approach reached the lowest common sense accuracy at classifying down to the minute. Lastly, we attempted to improve on this performance further by using a label transformation. This result was not as successful as the multi-head approach. Further experimentation, for example, in combining label transformation and multi-head, is required.

Task 3: Generative Models

Task Definition

One of the main deep learning tasks is to build models that capture the input data's probability distribution and then generate new data samples. The models that solve this task are called generative models, and the two main ones are Generative Adversarial Networks (GANs) and Variational Autoencoders (VAE). We will analyze their performance in generating images and compare the results by highlighting the main differences. Specifically, we aim to learn from a dataset (15746 samples) of 64x64x3 cat images to generate new cat faces.

Methods

GANs

A GAN has two main components: the **discriminator** and the **generator**. Its goal is to generate as realistic images as possible.

The generator takes as input an N-dimensional noise vector and using a series of deconvolutional layers, it generates an image. The discriminator is a binary classification that takes as input the images generated by the generator (fake images) and the real images of the dataset. Its main goal is to tell using a series of convolutional layers whether the input is a real or a fake image. The two models are trained through backpropagation in an alternated way. At each iteration, the discriminator is trained first and the generator second. During the training, the generator learns to generate more realistic images using the feedback of the discriminator while the discriminator learns to recognize fake images that become more realistic during training.

VAE

A VAE has two main components: the **encoder** and the **decoder**. The encoder takes as input an image, and using a series of convolutional layers, it generates a compressed representation of the input in a D-dimensional latent space. It then learns the parameters of the probability distribution that characterize the latent space. The distribution of the latent space is assumed to be a D-dimensional Gaussian, so the parameters learned are the mean and standard deviation. The decoder samples from the probability distribution learned by the encoder and uses the D-dimensional sampled vector as input. Using a series of deconvolutional layers, it generates an image that is compared to the one in the encoder's input.

The VAE is trained through backpropagation and bases the learning on the distance between the generated and the original image. If the training is successful, the encoder learns parameters of the probability distribution of the latent space such that the VAE generates images similar to the inputs.

Architecture of the models

Discriminator	**Encoder**		**Generator/Decoder**
5 × Conv2D(K:3,S:2,F:128)	5 × Conv2D(K:3,S:2,F:128)		Dense(O:1024)
Dense (O:1)	Dense (O:64)		5 × ConvTran2D(K:3,S:2,F:128)
	z_m: Dense (O:32) and z_s:Dense (O:32)		Conv2D(K:3,F:3)

The default architecture of the two models is very similar. Both models are trained for 50 epochs. The batch size used for GAN is 32, while for VAE, it is 8. In addition, instead of $ReLU$, the $LeakyReLU(\alpha = 0.3)$ is used as an activation function of the layers of the discriminator. After every layer (except for the input one) of the discriminator, a $Gaussian\ Noise(\sigma = 0.01)$ is also added. This is used to stabilize the training of the GAN. In addition we use MSE as loss function. Using it the model gives a signal to the generator about the fake samples that are far from the discriminator's decision boundary for classifying them as real or fake.

Experimental Results

GANs

(i) GAN generated images epoch 50

(ii) GAN generated image

(iii) image from the dataset

Figure 11: *Left* : Images generated from the GAN,*Right* : Comparison between generated and real image

After the hyperparameter tuning, we analyze the final results of the training in 11. Although the quality of the generated images is not the same of real images, it is possible to see that the GAN can produce realistic images of cats, as the comparison of images two and three of fig 11. From the first image of fig 11 we can see that sometimes GAN produces not-so-high-quality images, missing some important features of a cat face. Even if the introduction of the LeakyRelu activation function and the Gaussian noise in the discriminator contributed to the generator performances, the quality of the generated images improved greatly when we decreased the batch size from 64 to 32. We can hypothesize that a bigger number of update steps (because of the batch size reduction) allowed the GAN to stabilize the training and avoid local minima.

VAEs

(i) VAE generated image

(ii) VAE generated images epoch 50

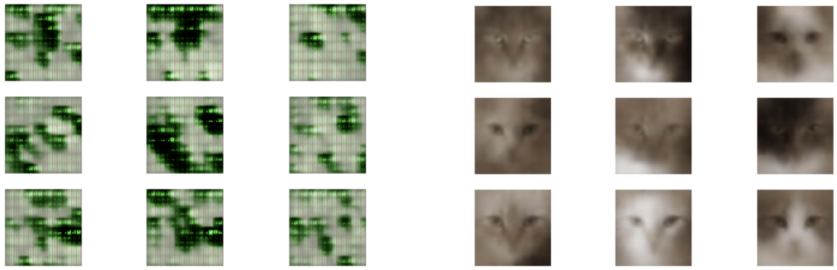
(iii) image from the dataset

Figure 12: *Left* : Images generated from the VAE,*Right* : Comparison between generated and real image

We analyze the final results of the training in 12. From fig 12, we see that the images generated by the VAE are more blurred than the ones generated by the GAN. But still, the generated images can represent cat faces quite well, as we can see by comparing the second and the third image of 12.

VAEs vs GAN

Now, let us compare the results of the two methods in fig 13. The first thing to notice is that the VAE takes less time to generate good images than GAN. This is probably due to the fact that the input of the decoder is a vector sampled from the distribution of the compressed representation of the input image. Whereas the input of the generator is a vector of random numbers. But in the end, the GAN generates sharper images and so more realistic images than VAE as it is possible to see from fig 11 and fig 12. More information from Figure 14. It is built by randomly sampling two vectors from the latent space, linearly interpolating



(i) GAN generated images epoch 2

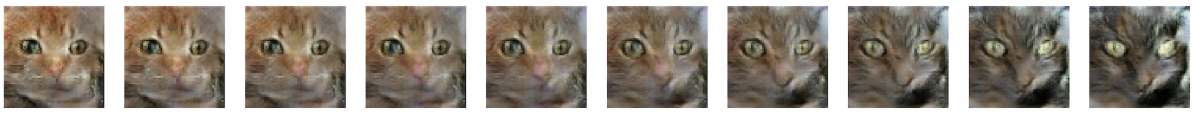
(ii) VAE generated images epoch 2

(iii) GAN generated images epoch 11

(iv) VAE generated images epoch 11

Figure 13: Images generated by VAE and GAN in different epochs

them, and showing the images generated from the latent points resulting from the linear interpolation. This visualization shows the distribution of the images generated from the latent spaces of the two models.



(i) images generated by linearly interpolated latent space points (GAN)

(ii) images generated by linearly interpolated latent space points (VAE)

Figure 14: Images visualizing the interpolation between two random points in the latent spaces of GAN and VAE.

Since the probability distribution of the latent space of a VAE is assumed to be a Gaussian distribution, the latent space tends to generate images with a visible symmetry. For this reason, the changes in the images along the latent space appear smoother. Whereas GAN tends to generate the best image given an input. For this reason, the changes in the images along the latent space appear sharper.

Conclusion and Discussion

From the analysis, it is easy to conclude that the GAN was the model which generated the best and most realistic images. On the other hand, the VAE can understand the main features of the inputs faster than the GAN. Thus the images generated in the first epochs are more realistic.

Dataset

The dataset used for the generative models can be found at the following link: https://www.kaggle.com/datasets/spandan2/cats-faces-64x64-for-generative-models In order to run the code, the dataset needs to be converted into a *.npy* file. If you want us to send the dataset already converted, please send us an email.

Contributions

For this assignment, we always worked together on the code and report. This means that the contributions can not be clearly divided. Rather, this assignment should be seen as a group effort.

References

[1] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[2] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[3] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[4] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.