Introduction to Deep learning

Assignment 2: building MLPs, CNNs and generative models with TensorFlow

Xiang He (s3627136)
Chentao Liu (s3083853)

Leiden Institute of Advanced Computer Science (LIACS)
Faculty of Science
Universiteit Leiden

November 22th, 2022

1 Introduction

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. Multilayer perceptron (MLP) is a fully connected class of feedforward artificial neural network. Convolutional neural network (CNN) is widely used for image and video recognition, recommender systems, and image segmentation. Variational autoencoder (VAE) and generative adversarial network (GAN) are classical generative models used to sample new data with the same distribution as the training set. In this report, we first learn how to use Keras and TensorFlow by comparing various MLP and CNN architectures. Then, we'll try to apply the knowledge to develop a CNN model for "tell-the-time" problem. Finally, two generative models will be used for anime face generation.

2 Learn the basics of Keras API for Tensorflow

The purpose of this section is to learn the basics of Keras API for TensorFlow. We ran the python scripts of MLP and CNN in GitHub on our device (CPU: AMD Ryzen 7 3700X 8-Core, GPU: NVIDIA GeForce RTX 3080 10GB), with tensorflow-gpu==2.10.0, keras==2.10.0, cuda==11.8.0. Based on the above python scripts, we trained the networks on the Fashion MNIST dataset and experimented with four initializations, activations, optimizers, and regularizations, each by orthogonal experimental design. After these optimal hyperparameters were chosen, we tried two different architectures of the network and three different learning rates. Finally, we decided on the three best ones for each network and trained new models on the CIFAR-10 dataset to see whether performance gains translated to a different dataset. The Fashion MNIST dataset and CIFAR-10 dataset were split into training data (80 %) and test data (20 %).

2.1 Turning hyperparameters on Fashion MNIST data

The First experiment we did was using orthogonal experimental design to find the optimal combination of initializations, activations, optimizers, and regularizations (Table 1). All the models (Basic CNN 9(a) and MLP model 9(b)) were trained on training set of Fashion MNIST for 20 epochs. The classification cross entropy loss was used.

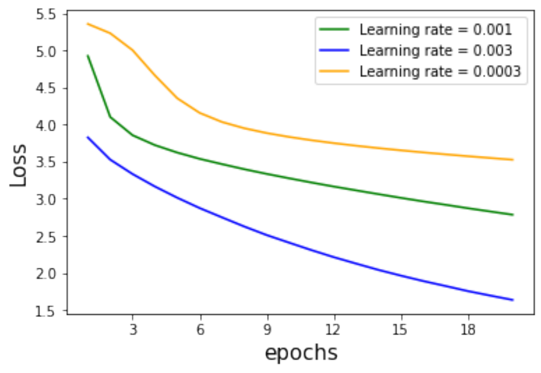
Table 1: Orthogonal experimental design to find optimal combination

Initializations	**Activations**	**Optimizers**	**Regularizations**	**MLP**		**CNN**	
				Train loss	Test loss	Train loss	Test loss
Ones	sigmoid	SGD	l1&dropout	25.110	25.110	3.084	3.084
Ones	relu	Adadelta	l2&dropout	37.261	37.261	6.103	6.120
Ones	tanh	Adam	l1	3.513	3.513	2.534	2.548
Ones	softmax	RMSprop	l2	2.304	2.304	2.306	2.306
Zeros	sigmoid	Adadelta	l1	27.205	27.205	5.517	5.517
Zeros	relu	SGD	l2	27.205	2.428	2.428	2.376
Zeros	tanh	RMSprop	l1&dropout	5.544	5.544	8.229	8.229
Zeros	softmax	Adam	l2&dropout	2.303	2.303	2.303	2.303
$\mathcal{U}(-0.05, 0.05)$	sigmoid	Adam	l2	1.727	1.731	2.306	2.306
$\mathcal{U}(-0.05, 0.05)$	relu	RMSprop	l1	5.592	5.592	6.768	6.768
$\mathcal{U}(-0.05, 0.05)$	**tanh**	**SGD**	**l2&dropout**	**1.061**	**1.084**	**0.700**	**0.726**
$\mathcal{U}(-0.05, 0.05)$	softmax	Adadelta	l1&dropout	25.739	25.739	5.063	5.063
$\mathcal{N}(0, 0.05)$	sigmoid	RMSprop	l2&dropout	1.817	1.827	2.311	2.311
$\mathcal{N}(0, 0.05)$	relu	Adam	l1&dropout	3.118	3.118	3.746	3.746
$\mathcal{N}(0, 0.05)$	tanh	Adadelta	l2	12.860	12.879	3.117	3.138
$\mathcal{N}(0, 0.05)$	softmax	SGD	l1	2.637	2.637	2.903	2.903

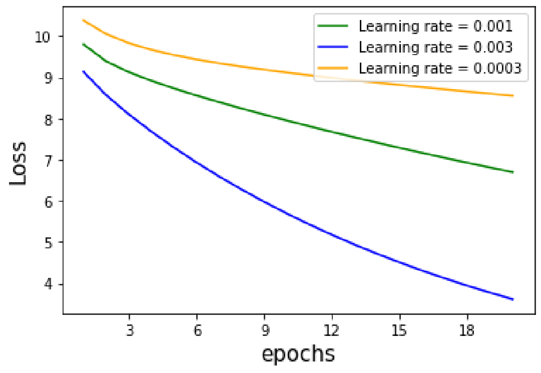
In the Table 1, we found $\mathcal{U}(-0.05, 0.05)$, tanh, SGD, l2 with dropout is the optimal combination to MLP and CNN model. The average loss of each parameter was compared to find the optimal one. For example, we would like to compare the performance of initialization, then we calculated the average of the loss of each initialization. The best one is the one with the lowest average loss.

Based on the optimal combination, we tried 2 different architectures of the network (Custom CNN Model: Add a Conv2 layer on top of the CNN 9(c), Custom MLP Model: Add a Dense layer and a

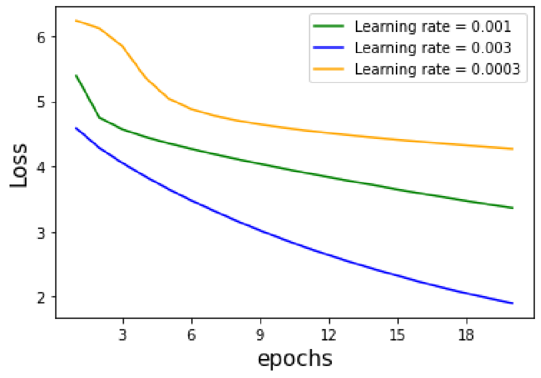
Dropout layer on top of the MLP 9(d)) and 3 different learning rates (0.001, 0.003, 0.0003). After 20 epochs training, we got the following results:



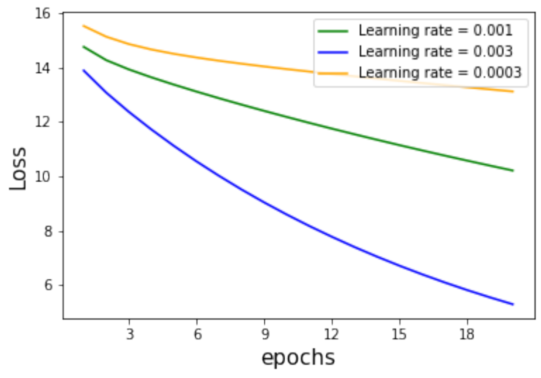
(a) Basic CNN Model



(b) Basic MLP Model



(c) Custom CNN Model



(d) Custom MLP Model

Figure 1: Effects of different architectures and learning rates

2.2 Applied the optimal model to CIFAR-10 data

Based on the performance, the first three best model of each network were chosen to train on CIFAR-10 data.

Table 2: Comparison of same model on different data set

Models		**MINST**		**CIFAR-10**	
architectures	learning rate	**Train loss**	**Test loss**	**Train loss**	**Test loss**
Basic MLP Model	0.001	6.668	**6.668**	15.276	15.280
Basic MLP Model	0.003	3.585	**3.608**	9.026	9.031
Custom MLP Model	0.003	5.273	**5.296**	11.047	11.053
Basic CNN Model	0.001	2.765	**2.789**	4.109	4.105
Basic CNN Model	0.003	1.613	**1.638**	2.910	**2.915**
Custom CNN Model	0.003	1.873	**1.901**	3.178	3.191

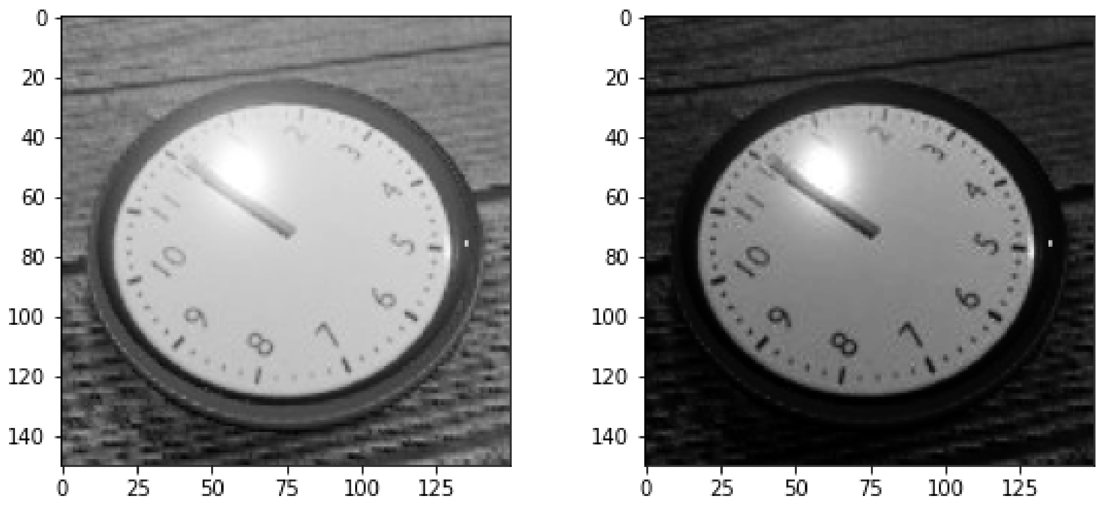
In Table2, we found that Basic CNN Model with a 0.003 learning rate performs best on CIFAR-10 data, and the loss of a model in Fashion-MINST data is usually lower than that in CIFAR-10 data. That means an optimal in one dataset may not work well on another.

3 Task 2

Convolutional Neural Network (CNN) is a kind of deep learning model that is dramatically good at extracting image features. We trained the CNN network on an analogue clock dataset in this task. We compared various Output layers and loss functions to find the best model structure. The experiment on the loss function is to avoid the "common sense" error. When dealing with the clock time prediction, the difference between 11:00 and 1:00 is just 2 hours, not 10 hours. The expected loss functions, such as Mae and Mse, do not consider this. So, we had to redefine a custom loss function to avoid it. In this task, we have designed a custom MSE function for "Transform Label" and a custom MAE function as a standard metric.

3.1 Preliminaries

According to the analogue clock dataset, the clock hands, numbers, and border are the essential features we want the model to focus on, while the wood grain in the background and clock board reflections is negative features. To counteract the effects of these disturbances, we have devised a method by enhancing the images' gamma values, which can effectively reduce their impact.



(a) Gamma= 1.0 (b) Gamma= 3.0

Figure 2: Visualization of Gamma Enhancing

We chose the `skimage` package to achieve the algorithm. As observed in Figure2, after increasing the gamma value from 1 to 3, significant features, including the clock's hands, borders, and numbers, are stranded out of the surrounding area.

Then, as shown in Table3, we constructed a primary CNN network. The network can be divided into two blocks. The first block has a few groups of Convolutional layers to extract image features; each group contains a Convolutional layer followed by a MaxPooling layer, a BatchNormalization layer, and a Dropout layer. The second block started with a Flatten layer and three Dense layers, including the Output layer.

Table 3: Primary CNN Network

Layer type	Output Shape	Param
conv2d (Conv2D)	(None, 148, 148, 32)	320
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
batch_normalization (BatchNormalization)	(None, 74, 74, 32)	128
dropout (Dropout)	(None, 74, 74, 32)	0
conv2d (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2 (MaxPooling2D)	(None, 36, 36, 64)	0
batch_normalization (BatchNormalization)	(None, 36, 36, 64)	256
dropout (Dropout)	(None, 36, 36, 64)	0
conv2d (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d (MaxPooling2D)	(None, 17, 17, 128)	0
batch_normalization (BatchNormalization)	(None, 17, 17, 128)	512
dropout (Dropout)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 256)	9470208
dense (Dense)	(None, 128)	32896
output (Dense)	(None, 1)	129

3.2 Regression Function Label Transformation

To implement the regression model, we need to transform the label format as one dimension, so we change the label from $[hours, minutes]$ to $[hours + minutes/60]$. For example, $[1, 30] \to [1.50]$. After that, we should change the Output layer as a one-unit Dense and choose 'linear' as its activation (Just like Table3). Before we train the regression network, a custom metric function should be designed. Our custom Mae function shows below.

Listing 1: Custom Mae

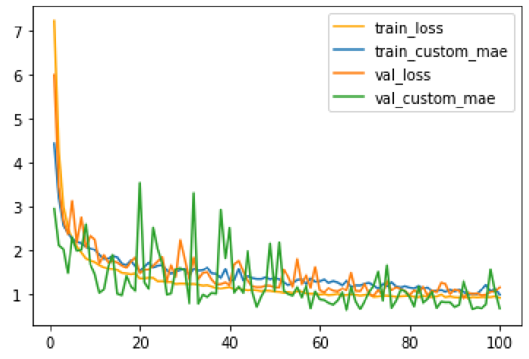
```
def custom_mae(y_true, y_pred):
    tail = 0.001
    penalty = 10.0
    j_0 = tf.minimum(y_pred, 0.0) - tail
    j_1 = tf.maximum(y_pred-12.0, 0.0) + tail
    diff1 = tf.math.abs(y_true - y_pred)
    diff2 = tf.math.abs(y_true - y_pred - 12.0)

    return tf.minimum(diff1, diff2)
           +penalty *(tf.math.log(j_1)+tf.math.log(-j_0)-2.0*tf.math.log(tail))
```

Here we tried to define a MAE that periodic changes. Also, the value smaller than 0, or larger than 12, are penalized. However, the problem is that this MAE is not perfectly continuous, and it would have two local minima in the range of 0 to 12. But the result shows that the loss decreased very slowly during the training process when we used the custom MAE as a loss function. For this reason, we used MAE as the loss function and the custom MAE as the metric.

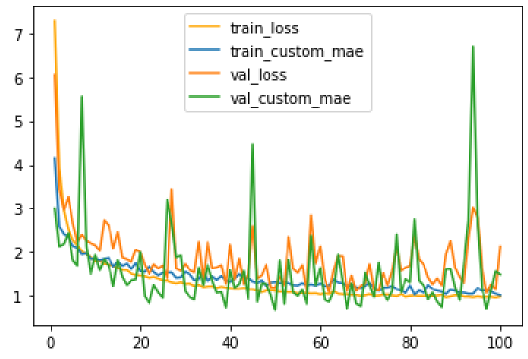
We trained the model both on the gamma-enhanced data and the original data, by comparing the training plot and the result of the test set to verify that the gamma-enhancing technique is valid. The batch size we chose is 32, and the epoch is 100, with 0.001 learning rate .

Table 4: Regression results

	Regression with Gamma Enhance	**Regression without Gamma Enhance**
Training Loss	0.9072	0.9816
Test loss	1.1515	2.1259
Training custom Mae	0.9779	1.0090
Test custom Mae	0.6669	1.4811



(a) Regression with gamma enhance



(b) Regression without gamma enhance

Figure 3: Regression training plot

The results in Table4 show that the gamma-enhancing method performs better on both MAE and custom MAE and with lesser over-fit on the test set. Also, the decline of the loss values is more stable in Figure3. We have validated this approach on all models in this report, so this strategy is used by default in subsequent models. Otherwise, we can notice greater volatility in the custom Mae's loss curve compared to the Mae. This is because the Mae function does not concern the common error problem. The process will produce a wrong prediction when the actual value is near 0 o'clock, which increasing the loss suddenly.

Therefore, a label transformation function is provided. This function represents each label as a group of trigonometric functions, which means all clock time can constitute a periodic function, and no common mistake exists. We define a label transform function and a custom loss function to achieve this algorithm. The details are shown below.

Listing 2: Label Transform function and Custom Mse

```
def label_transform(time_label):
    trans_label = np.zeros(shape = (len(time_label), 4))
    for i in range(len(time_label)):
        hour = time_label[i][0]
        minute = time_label[i][1]
        hour_sincos=(np.sin(hour*(1/6)*np.pi+minute*(1/360)*np.pi),np.cos(hour*(1/6)
                   *np.pi+minute*(1/360)*np.pi))
        minute_sincos = (np.sin(minute * (1/3) * np.pi), np.cos(minute * (1/3)*np.pi))
        time = [hour_sincos[0], hour_sincos[1], minute_sincos[0], minute_sincos[1]]
        trans_label[i] = time
    return trans_label

def custom_mse(y_true, y_pred):
    return tf.sqrt((tf.reduce_sum(tf.square(y_true - y_pred)) * [3600, 3600, 1, 1]) /
        3601)
```

We transformed the hours label into the cos and sin value of the angle between hour's leg and the 12 o'clock. The same thing was being done on minutes label. The MSE here was defined as the weighted average of the distance between the true time label and the prediction time label. The weight is the 60 for hour and 1 for minute, which measured the common sense loss. Then we can train the Label transformation regression model. We used the same model structure as the regression model and the same hyperparameters.

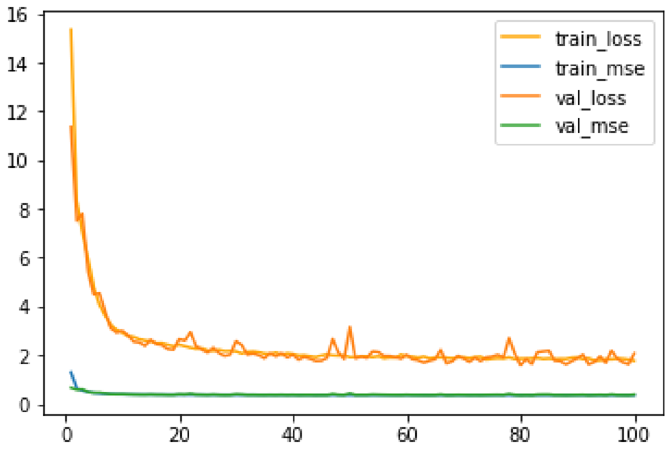


Figure 4: Label Transformation plot

Training Loss	1.7719
Test loss	2.0738
Training Mse	0.3582
Test Mse	0.3779

Table 5: Label Transformation results

As Fig4 shows, the rate of decline of the training and test loss is slow. In the 3.5 sections, we will compare all models using the same criteria.

3.3 Classification Function

Another solution is to consider this a multi-classification problem. In this case, we transform the labels from $[hours, minutes]$ to $[hours * 60 + minutes]$, thus we divided the labels into 720 categories. A 720-class classification problem is almost impossible, so we group a range of categories into one category. For example, if we consider every 30 minutes as a category, we have 24 categories (30/720) in total. The predict of classification model should be a n length tensor (n depend on classes number), and the final result can be got by argmax function. Therefore, we designed the Output layer as a Dense layer with n units and activated by SoftMax. Then we chose categorical_crossentropy as the loss function and categorical_accuracy as the metric (both provided by Keras). Based on experience in regression model, we trained classification model on gamma-enhanced dataset, with batch_size = 128, epochs = 100 and learning_rate = 0.001.

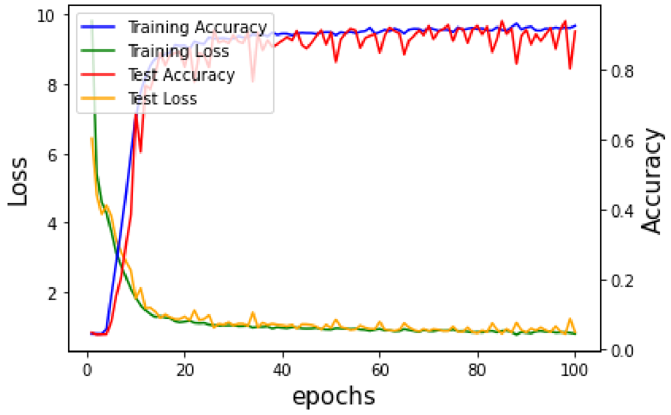


Figure 5: Classification training plot

Training Loss	0.7889
Test loss	0.8344
Training accuracy	0.9253
Test accuracy	0.9088

Table 6: Classification results

As shown in Figure5 and Table6, the model performs well in handling the 24 classification problems, achieving 90% classification accuracy on the test set. Next, we attempted to train the classification model with 720 categories and set epochs=100. From Figure6, it can be seen that the loss is dropping very slowly on both the training and test set. The final test accuracy can only reach 0.00055 after 100 epochs.

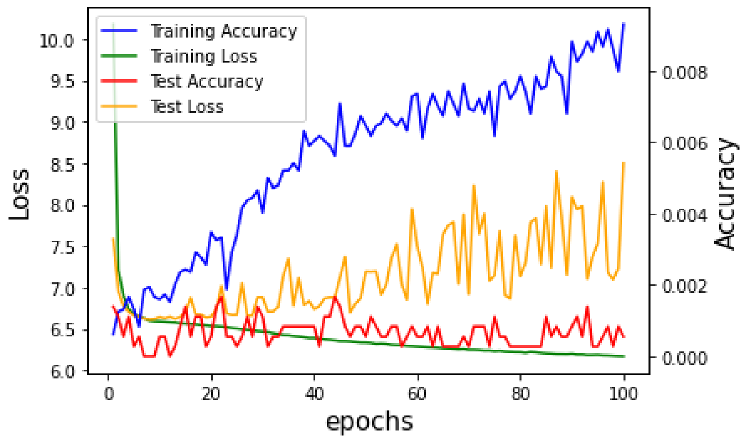
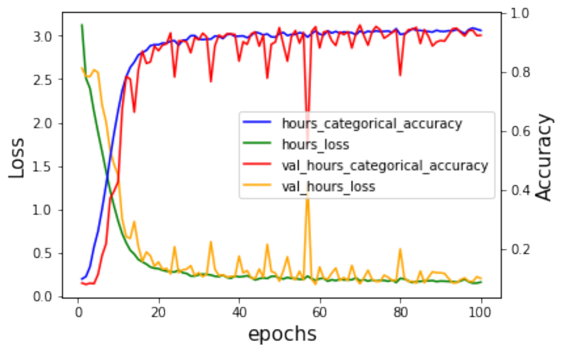


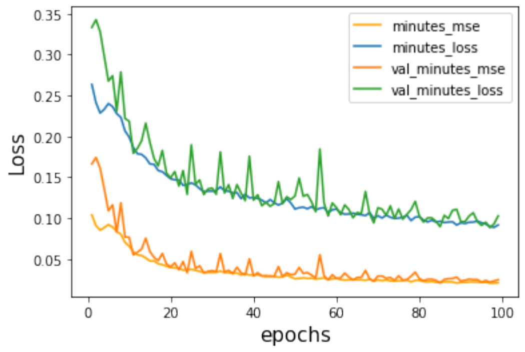
Figure 6: Classification training plot (720 categories)

3.4 Multi-Head Function

Based on the knowledge above, we can design a multi-head neural network that predicts time and minutes separately. Thus, we combine the regression and classification models, reserve both the regression Output layer (output_regress = Dense (1, activation = 'leaner', name='minutes') and classification Output layer (output_class = Dense (12, activation = 'softmax', name='hours') as a multi-head Output 9(e). In addition, we have to divide the labels into hour labels and normalization minute labels, such as (label:[1, 30] is divided to hour:[1] and minute:[30/60]). Finally, we can compile the multi_model = (inputs = input, outputs = [output_class, output_regress]) with loss functions categorical_crossentropy and Mae. Then we trained the Multi-head model with batch_size = 128, epochs=100, learning_rate=0.001.



(a) Hours



(b) Minutes

Figure 7: Multi-head model training plot

Table 7: Multi-head results

Hours training Loss	0.1653
Minutes training Loss	0.0917
Hours test Loss	0.2089
Minutes test Loss	0.1028
Hours training accuracy	0.9406
Hours test accuracy	0.9242
Minutes training Mse	0.0212
Minutes test Mse	0.0248
Training loss	0.7760
Test loss	0.8277

The multi-head model performs quite well. Hours forecasts had an accuracy of 92%, while minutes predictions achieve only 0.02 MSE on test set.

3.5 Final Model

We compared the three models that successfully predicted the complete clock above to find the most effective model for this task. As these models use different loss functions and labelling strategies, we needed to develop a uniform comparison criterion before comparing them.

We chose to convert both the predict of each model and their test labels into 'Label transform' format and to calculate their errors on the test set by the Custom MSE method (Just like we did in Listing2).

Table 8: Comparison of models with the same training setting

Models		**Test Loss**
Architecture	Loss function	Custom MSE
Regression	MAE	0.6075
Label transform Regression	Custom MSE	13.4489
Multi-head	categorical crossentropy and Mae	0.0942

The results (Table8) show that the Multi-head Model performs best, and the significant loss of the Label transformation regression model may be caused by more restriction from MAE in regression than custom MSE defined for common sense loss. When the default MAE is applied, it restricts the observations that close to 12 o'clock. For example, when the true label is 0:05, the prediction can not be 11.55 if the default MAE is applied. However, our custom MSE can not prevent this situation happened.

3.6 Other work

We also tried two other models combining the above knowledge. One is to divide the minute labels into 30 classes and build a multi-head model through two classification output layers. Otherwise, we also tried to design different fully-connected layers for the two output layers of the multi-head model. But results show that the multi-head model we plan on 3.4 still performs best.

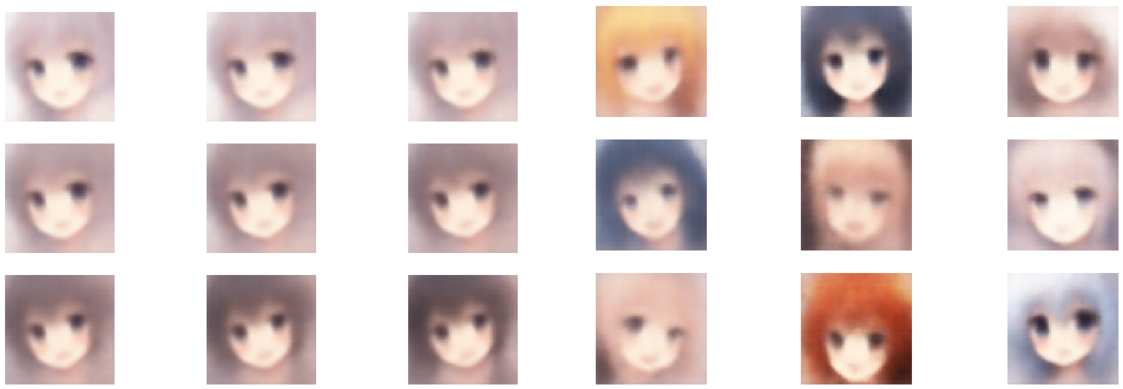
4 Generative Models

The purpose of this section is to develop a general idea about the working principles of generative models. Also, we chose a suitable data set, anime face data set, for VAE and GAN, to compare their performance based on observation of generated plots. The anime face dataset that we used is generated by Crypko and collected by Arvin Liu. Finally, we created one of these visualizations by linearly interpolating between two random points in the latent space and seeing how the output of VAE and GAN changes.

4.1 VAE

A generative model usually two components: a downsampling architecture, which either extract features from the data or model its distribution; and a upsampling architecture, which will use some kind of latent vector to generate new samples that resemble the data that it was trained on. The basic architecture of autoencoder is a good start to illustrate generative model. An autoencoder has an encoder to extract features from the data, the data here is high dimensional and the features are in low dimension space. So, it works like a dimensionality reduction method, e.g. PCA, to gets compress image to a vector. Then, the decoder uses the feature vector to reconstruct a dataset that as close as possiable to the original distribution. So, the loss can be measured by distance between the original data and the reconstructed data.

Compared to the autoencoder, the encoder used by VAE is slightly changed. The feature vector extracted from different plots from a same dataset is added by a noise vector. The noise vector is usually normal distributed. Intuitively, we hope that could let the feature vectors form a plane in low dimensional space, from which a sample could be a combination of original two or more observations. As a result, we could randomly generate a vector and give it to the trained decoder to get 'new' plots.



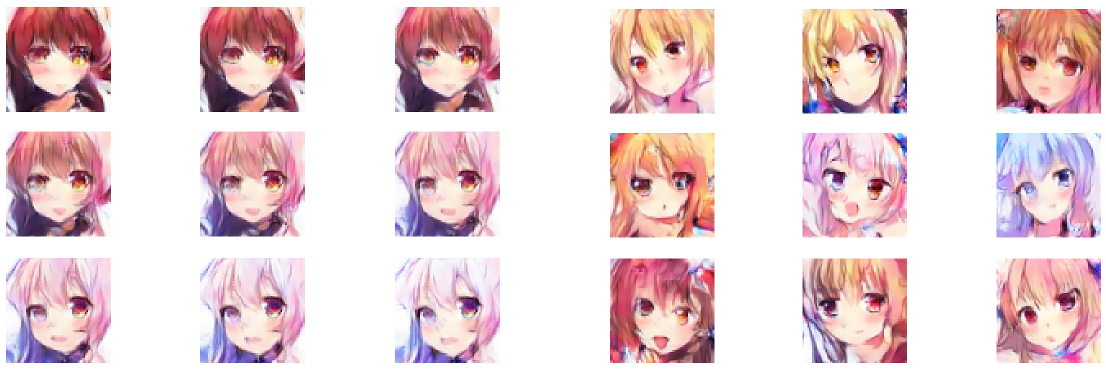
(a) Linearly interpolate between two random latent vector

(b) Different learning rates

Figure 8: Random generation of VAE

4.2 GAN

Although the VAE and GAN all belong to generative model, the idea of GAN is different from VAE. In fact, the VAE never learns to recognize the image, it may just memorize the existing images, instead of generating new images. As for GAN, it also contains two parts: generator and discriminator. The generator will receive the random input vectors and generate images. The discriminator will try to distinguish generated images from the real images. In the training process, they work like enemies to each other until one of them can't perform better. Finally, the generator can be used for image generation.



(a) Linearly interpolate between two random latent vectors

(b) 9 random normal latent vectors

Figure 9: Random generation of GAN

4.3 Comparison of two generative models

In the Figure 8 & 9, we found that VAE performance worse than GAN, while it has a quick train speed than that of GAN. That's bacause VAE never learns to recognize the image, it may just memorize the existing images, instead of generating new images. As for the plots generated by Linearly interpolating between two random latent vectors, the hair and the eye are the features that changes in VAE model, while the plots are vague. As for GAN model, it generated more beautiful plots, and the features of the anime face changes a lot.

5 Summary

In this project, we learned the basics of Keras API for TensorFlow by experimenting with various initializations, activations, optimizers, and regularizations. Also, we tried several model architecture and learning rates on Fashion MNIST dataset. Based on their performance, the first three models of each network were chosen and applied to CIFAR-10 dataset.

We discussed several methods to solve the 'tell-the-time' problem in the second part of this report. Regression model, classification model and multi-head model were compared, and several methods, like gamma-enhancing, periodic MAE, and label transformation with its custom MSE were used to optimize their performance. Finally, we found the Multi-head Model performs best on custom sense loss.

Finally, two generative models, VAE and GAN were discussed in the third part of the report. We found an anime face dataset, and used it to train the model. The decoder of VAE and the generator of GAN were used to generate new pictures.

6 Appendix

Table 9: Models

(a) CNN Model

Layer type	**Output Shape**	**Param**
Conv2D	(None, 26, 26, 32)	21952
Conv2D	(None, 24, 24, 64)	18496
MaxPooling2D	(None, 12, 12, 64)	0
Dropout	(None, 12, 12, 64)	0
Flatten	(None, 9216)	0
Dense	(None, 128)	1179776
Dropout	(None, 128)	0
Dense (output)	(None, 10)	1290

(b) MLP Model

Layer type	**Output Shape**	**Param**
Dense	(None, 512)	401920
Dropout	(None, 512)	0
Dense	(None, 512)	262656
Dropout	(None, 512)	0
Dense (output)	(None, 10)	5130

(c) custom CNN Model

Layer type	**Output Shape**	**Param**
Conv2D	(None, 26, 26, 32)	21952
Conv2D	(None, 24, 24, 64)	18496
Conv2D	(None, 22, 22, 128)	73856
MaxPooling2D	(None, 11, 11, 128)	0
Dropout	(None, 11, 11, 128)	0
Flatten	(None, 15488)	0
Dense	(None, 128)	1982592
Dropout	(None, 128)	0
Dense (output)	(None, 10)	1290

(d) MLP Model

Layer type	**Output Shape**	**Param**
Dense	(None, 512)	401920
Dropout	(None, 512)	0
Dense	(None, 512)	262656
Dropout	(None, 512)	0
Dense	(None, 512)	262656
Dropout	(None, 512)	0
Dense (output)	(None, 10)	5130

(e) Multi-head Model

Layer type	**Output Shape**	**Param**
InputLayer	(None, 150, 150, 1)	0
Conv2D	(None, 148, 148, 32)	320
MaxPooling2D	(None, 74, 74, 32)	0
BatchNormalization	(None, 74, 74, 32)	128
Dropout	(None, 74, 74, 32)	0
Conv2D	(None, 72, 72, 64)	18496
MaxPooling2D	(None, 36, 36, 64)	0
BatchNormalization	(None, 36, 36, 64)	256
Dropout	(None, 36, 36, 64)	0
Conv2D	(None, 34, 34, 128)	73856
MaxPooling2D	(None, 17, 17, 128)	0
BatchNormalization	(None, 17, 17, 128)	512
Dropout	(None, 17, 17, 128)	0
Flatten	(None, 36992)	0
Dense	(None, 256)	9470208
Dense	(None, 128)	32896
Dense (output1)	(None, 12)	1548
Dense (output2)	(None, 1)	129