STAT 4012 – Statistical Principles of Deep Learning with Business
Group Project

KWOK Ho Hin (SID: 1155126159)
LAI Tsz Chun (SID: 1155125208)
LAM Wai Chiu (SID: 1155152095)
LAW Yiu Leung Eric (SID: 1155149315)
TSOI Tung Sing (SID: 1155127274)
LO Chak Kin Steven (SID: 1155125491)

May 7, 2022

Contents

List of Figures

List of Tables

1 Real Time Emotion Detection Deep Learning Model

This section is contributed by LAI Tsz Chun (SID: 1155125208), LAM Wai Chiu (SID: 1155152095) and LAW Yiu Leung Eric (SID: 1155149315).

1.1 Introduction

The study of Convolutional Neural Network (CNN) in this course has inspired us the importance of CNN in image classification. We further extend our inspiration to facial expression recognition. Emotion serves as an immediate means for humans to interact and sometimes it also affects human behavior. Hence, we aim to replicate human emotional recognition using CNN in a real time manner.

We designed to build an artificial intelligence (AI) that consists of two models. The first model is a face detection model borrowed from MTCNN. The second model is our custom CNN model for facial expression recognition. The main objective is to recognize seven basic emotional expressions (Angry, Disgust, Fear, Happy, Sad and Surprise, Neutral) by our custom model with a satisfactory accuracy rate. A demonstration using our model in real time and a model evaluation are also covered to see the performance of the model.

1.2 Dataset

FER-2013 (1) dataset from Kaggle is selected for model training. The data consists of 48×48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image. The faces are categorized into seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Neutral, 5=Sad, 6=Surprise). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

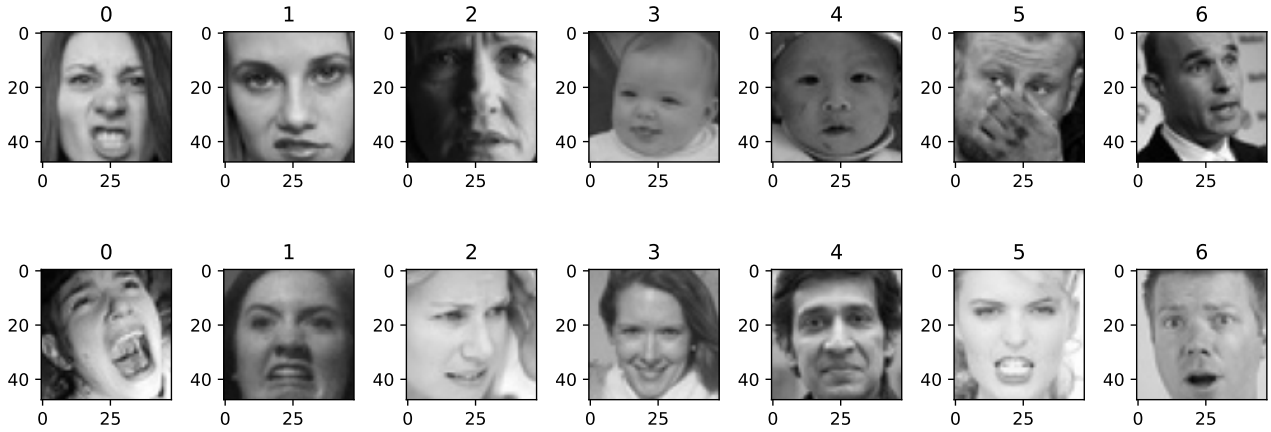

Figure 1: Training Dataset

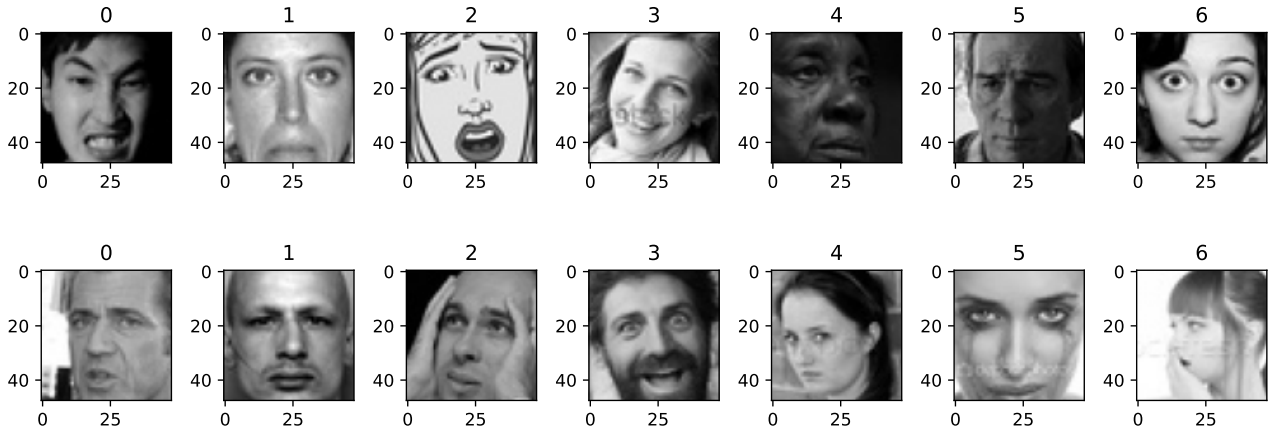

Figure 2: Testing Dataset

1.3 Data Preparation for Model Training

The data are stored in `.jpg` format, in order to use them, we must perform pre-processing to the dataset. We acknowledge that `Tensorflow` does provide a function `tf.keras.preprocessing.image.ImageDataGenerator`[1] for loading image files from folders into data generator, which only load the image into tensor when in needed. However, we insist to save the data in `NumPy` format in order to speed up training process. Yet, this method have some drawbacks, which occupies about 2 GB of RAM for loading datasets before training, and extra storage space in hard disk.

1. Read the images using `OpenCV`.
2. Normalized the grayscale from [0, 255] to [0, 1] for increasing training speed.
3. Stack up 1-channel grayscale array into 3-channel RGB array, as the final use would be RGB pictures.
4. Encode the labels.
5. Save the array to `NumPy` array, then save as `Pickle` file.

Noted that the dataset have already split into training and testing datasets from Kaggle, we do not need to split them.

Implementation: `emotion_detection\src\read_data.ipynb`

1.4 Model Design

We intended to build a real-time emotion detection AI, so there are two separated models needed, namely face detection model and emotion detection model. Flow of the model:

1. Detect face(s) in a photo.
2. Capture pixels of the face(s) and resize to 48×48.
3. Classify emotion of the faces(s).

[1] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

1.4.1 MTCNN Face Detection Model

MTCNN is a python (pip) library written by Github user ipacz (2), which implements the paper by Zhang, Kaipeng et al. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks." (3) In this paper, they propose a deep cascaded multi-task framework using different features of "sub-models" to each boost their correlating strengths. We choose MTCNN model for detecting faces in picture.

Figure 3: Face Detection using MTCNN

1.4.2 Augmentation

Too avoid over fitting, we use a technique called augmentation to increase the diversity of your training set by applying random flip and slight rotation.

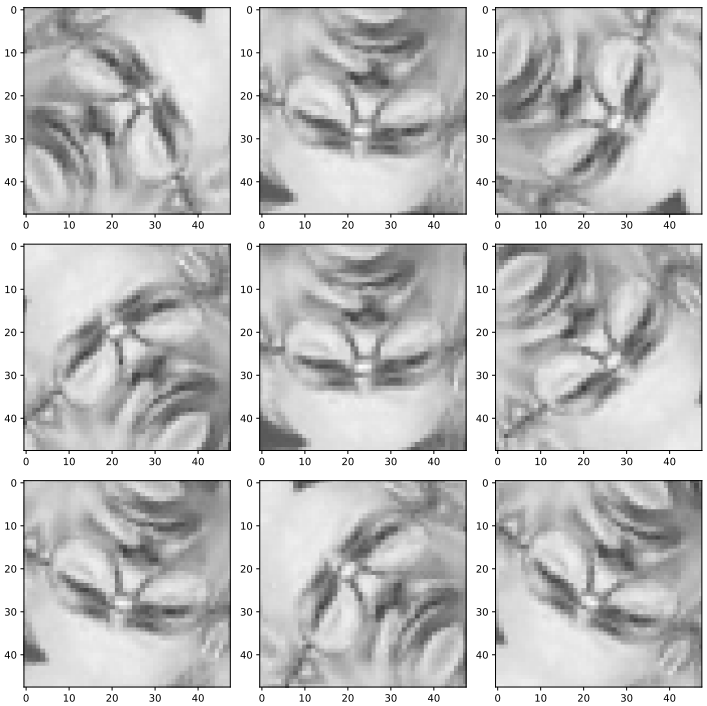

Figure 4: Augmentation

1.4.3 Emotion Detection Model

We create and train a custom CNN model. After the augmentation function mentioned above, the pixels are passing through 5 blocks of convolution layers and 2 hidden fully-connected layers. All convolution layers and hidden layers are using RELU activation, expect the output layer is using softmax activation.

Block 1 and 2 each consists 2 layers of CNN layers, while block 3 to 5 each consists 3 layers of CNN layers, the filter size is increasing block by block from 64 to 512. In additions, each block has it own batch normalization layers, and from block 2 to 5, l2 regularization is used for kernel regularization.

Implementation: `emotion_detection\src\model_train.ipynb`

1.4.4 Compile Model

The loss function is set to be sparse categorical cross-entropy, which is most suitable for multi-classes classification artificial neural network (ANN). Also, Adam is used as the optimizer for model training, with a common learning rate = 10^{-4}.

In total, this model consists 33,631,815 parameters, in which 33,628,871 of them are trainable and 2,944 are non-trainable (parameters for batch normalization).

1.4.5 Model Architecture

The architecture of our CNN model is as next page.

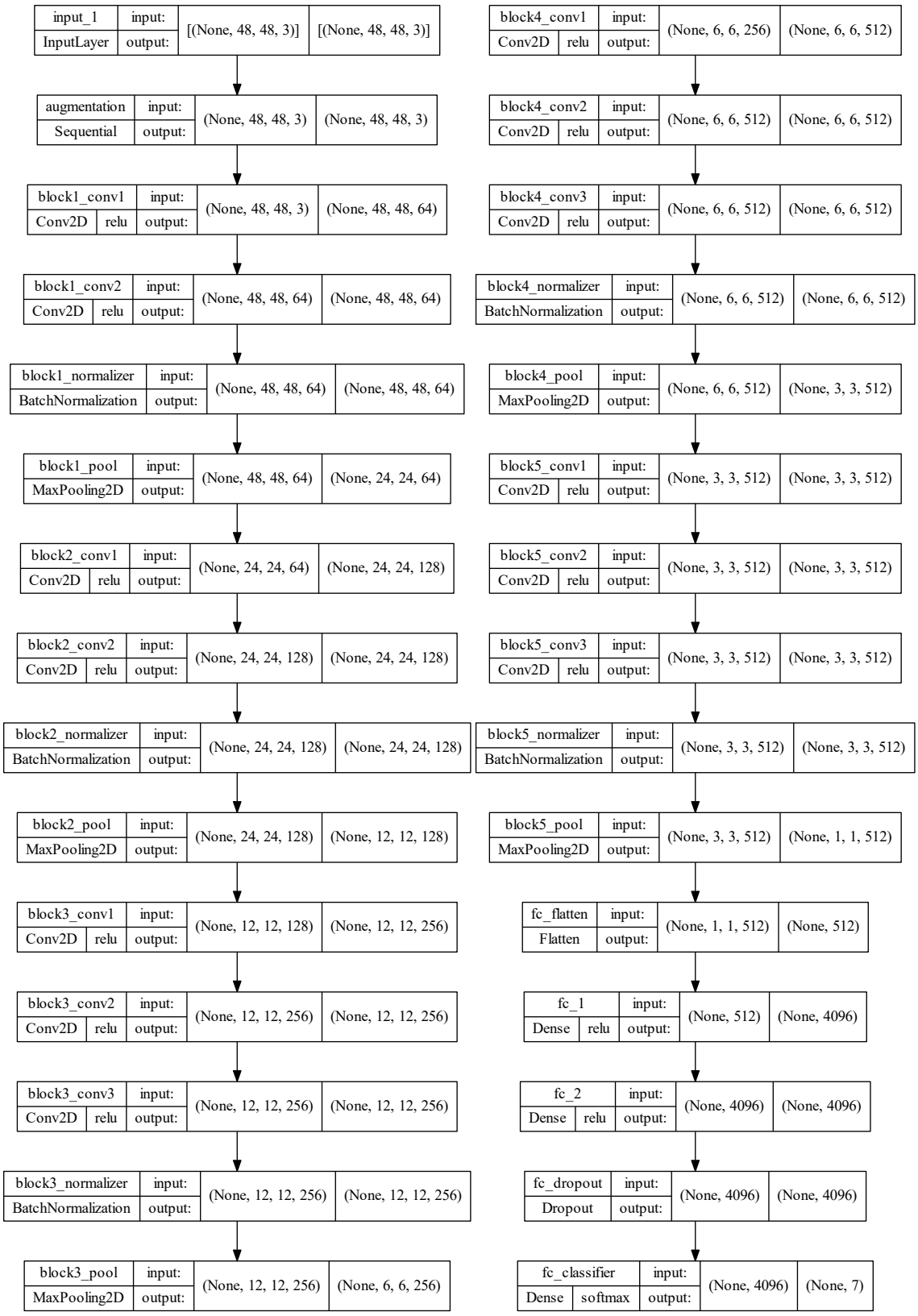

Figure 5: Model Architecture

1.5 Model Training

1.5.1 Training Specifications

We aim to train the model to an optimal point, neither overfiting nor underfitting. Thus, we set to train 200 epochs, but training procedure is expected to be stopped early, if the validation accuracy does not improve after 15 epochs. The final model would the one which has best validation accuracy.

Training a very deep CNN requires huge amount of GPU power, in which non of our group member could afford it. Thanks to Google, they offer Colab which let us to use their GPU for free.

1.5.2 Model Evaluation

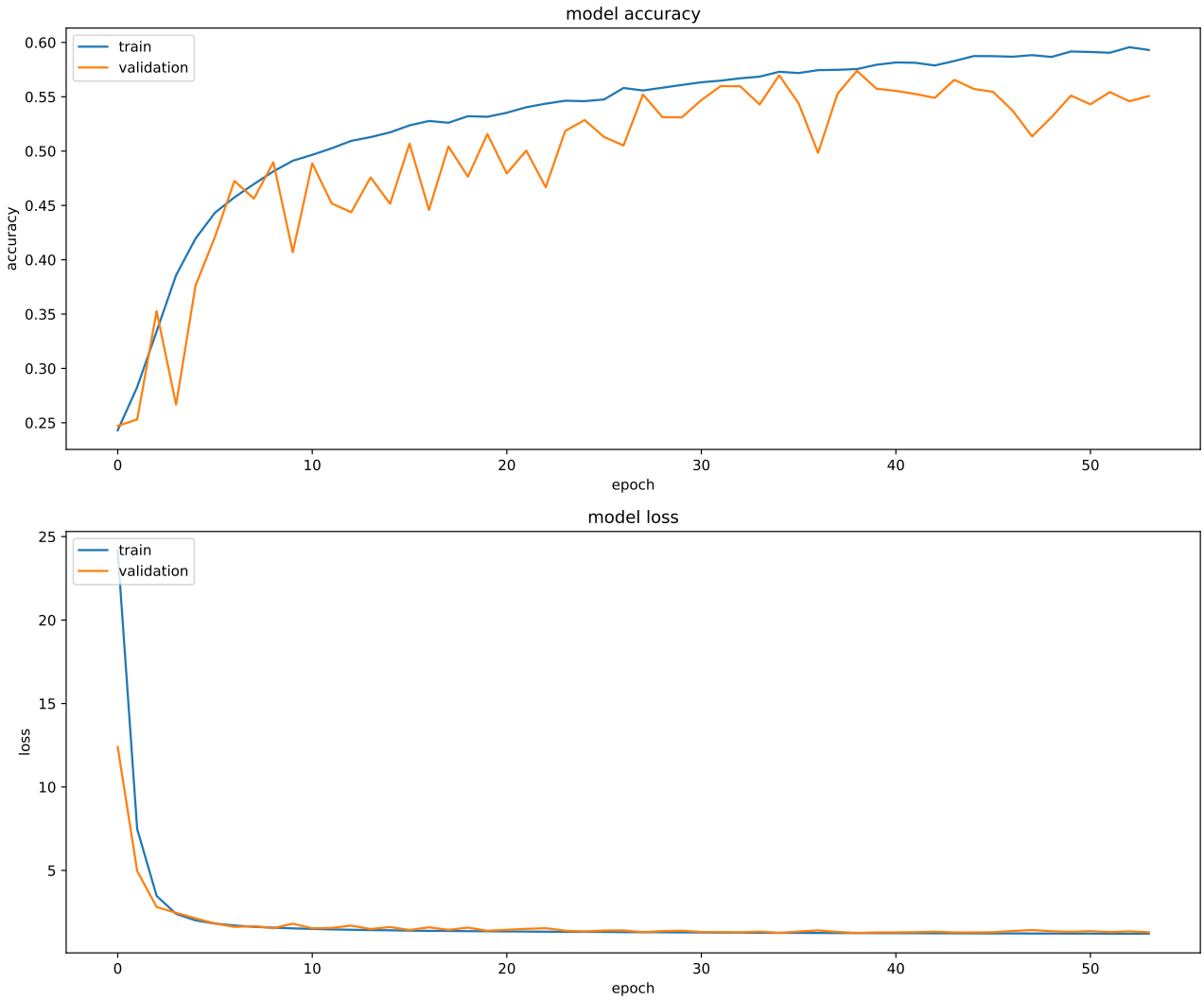

Figure 6: Loss and Accuracy

Early stopping was trigger after 54 epochs, the model eventually achieved valid accuracy at around 0.58.
With a Tesla T4[2] GPU, each epoch requires around 70 seconds, in total around an hour for complete training, which is reasonable.

[2]Nvidia Tesla T4 is optimized for mainstream computing environments and features multi-precision Turing Tensor Cores and new RT Cores

1.6 Demonstration

This AI emotion detection solution not only support real time detection, it also allows end-users input a video for detection. For demonstration, we use a video clip from 94th Academy Awards[3]. With sufficient resolutions, the AI could easily detect happy faces.

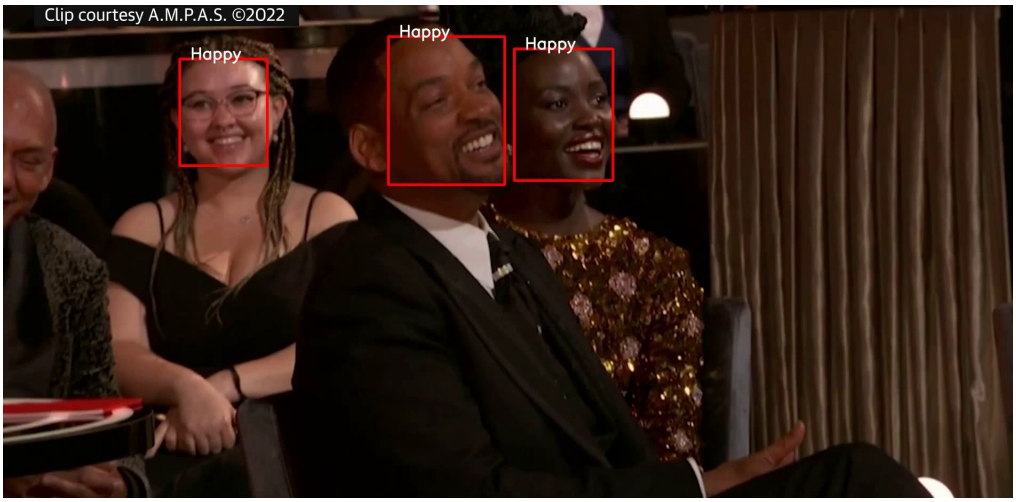

Figure 7: Happy Faces

Implementation: `emotion_detection\src\video.py`

Back to our original objective of real time emotion detection, we use laptop webcam to capture real-time image, then pass it through into the model. The model runs pretty well in terms of performance, expect this demonstration is running on CPU, which results in very low frame rate.

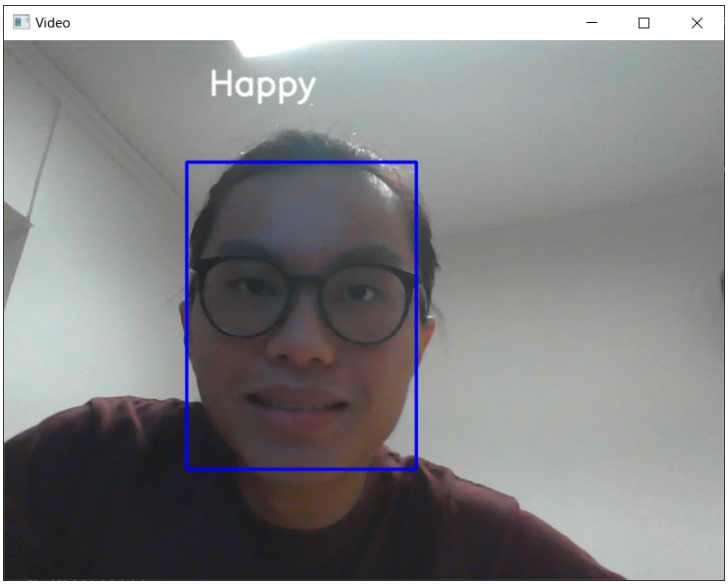

Figure 8: Live Webcam Demonstration

Implementation: `emotion_detection\src\real_time_webcam.py`

[3] the video clip is adjusted to 3 fps

1.7 Discussion

1.7.1 Basic Concept of Classification Evaluation

In this part, we will evaluate the performance of our CNN model by different indicators, namely confusion matrix, accuracy, precision, recall and F1-score.

Confusion matrix: confusion matrix can show the summary of prediction and will be used to calculate accuracy, precision, recall and F1-score.

	True Class	
Predicted Class	True Positive (TP)	False Positive (FP)
	False Negative (FN)	True Negative (TN)

Table 1: Confusion Matrix

Since our project is performing multi-class classification, an example of multi-class classification confusion matrix that an angry image is predicted as angry will be given as follows:

		True Class						
		Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Predicted Class	Angry	TP	FP					
	Disgust	FN	TN					
	Fear							
	Happy							
	Neutral							
	Sad							
	Surprise							

Table 2: Multi-Class Confusion Matrix

1.7.2 Model Evaluation

		True Class							
		Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise	
Predicted Class	Angry	557	9	59	53	134	127	19	958
	Disgust	35	51	8	4	3	10	0	111
	Fear	143	11	293	46	184	260	87	1024
	Happy	45	2	19	1542	94	47	25	1774
	Neutral	63	4	21	76	888	169	12	1233
	Sad	136	12	72	51	317	647	12	1247
	Surprise	32	1	74	58	70	14	582	831
		1011	90	546	1830	1690	1274	737	7178

Table 3: Prediction Outcome

	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Accuracy	0.8809	0.9862	0.8629	0.9276	0.8402	0.8291	0.9437
Precision	0.5814	0.4595	0.2861	0.8692	0.7202	0.5188	0.7004
Recall	0.5509	0.5667	0.5366	0.8426	0.5254	0.5078	0.7897
F1-score	0.5658	0.5075	0.3732	0.8557	0.6076	0.5133	0.7423

Table 4: Accuracy, Precision, Recall and F1-score

In terms of accuracy, each class has great performance with all over 82% accuracy. However, only using accuracy could not truly evaluate the overall performance, thus, we take F1-score to access performance among different classes. Form the table above, we can see that the performance of class **Fear** is the worst while the performance of class **Happy** is the best. The performance different of them is relatively large 0.3732 vs 0.8557. The reason may be **Fear** faces have similar characteristic with **Angry**, **Sad** and **Surprise**, we can see in Table 3 that the mentioned three classes also have many samples mislabeled as **Fear**. In such case, the CNN model is not able to identify unique characteristics to differentiate those classes.

1.8 Conclusion

In this part of our project, we have developed a CNN model to perform facial recognition instantly. There are seven emotions can be classified, namely, angry, disgust, fear, happy, sad, surprise and neutral. The FER-2013 dataset is used which contain 28,709 examples in training set and 3,589 examples in testing set. After we train the model, around 60% testing images can be correctly classified. There are some factors will affect the classification accuracy rate of the model, like the quality of the images, the resolution of the images in each emotion class, mislabeling in the dataset, etc. To improve the accuracy, we can relabel the dataset to avoid misleading result, which, however, is extremely time consuming. The other way to make some possible improvements is to try other models, like recurrent neural network (RNN) or use some advance techniques, like Cycle Generative Adversarial Network (Cycle GAN), VGG-Face model and residual neural network (resnet).

Facial recognition is used in different area nowadays which bring a huge benefit to our society. One big usage is customer service. Facial recognition can allow to record the emotion of the customer. The company can provide more training to the staffs if a certain proportion of customers get bad emotions after the service. Facial recognition can also assist to interview candidate. The trained model can determine the emotion of candidate automatically and report to the human recruiters which can reduce the cost. Our group believe that facial recognition will be more commonly used in the future.

2 Cryptocurrencies Pairs Trading

This section is contributed by KWOK Ho Hin (SID: 1155126159), TSOI Tung Sing (SID: 1155127274) and LO Chak Kin Steven (SID: 1155125491).

2.1 Introduction

With the rapid advancement in Deep Learning, experts have been searching for applications of deep learning models in various industries, especially in Finance. Deep Learning models, particularly LSTM model, have been used extensively in developing quantitative trading strategies. Typically, researchers tries to use LSTM to predict stock prices, or implied volatility in the case of options.

In this project, we aimed to incorporate LSTM model in a Crypto Pairs Trading Strategy (4). We first identified potential cointegrated pairs (token0, token1) and their respective hedge ratio (β). Then we tried to use LSTM to predict the spread of the pair (i.e. token0 – $\beta \cdot$ token1) using only price data. A simple pair trading strategy can be designed by keeping track of the long-term mean (μ) and setting a threshold (e) – we may buy the spread when it is below $\mu - e$ and sell it when it is above $\mu + e$. It is under the assumption that the cointegrated relationship between the two token will remain unchanged, hence the spread will be a stationary time series and will revert to its long-term mean mu.

2.2 Dataset

We have collected 10 months of minute-frequency data for 16 tokens with high market capitalization and trading volumes from Binance api. It includes candlesticks data, stored in OHLCV[4] format, aggregated in minute timeframe.

BTC	ETH	DOGE	BNB
XRP	TRX	ZIL	ADA
WAVES	ETC	LTC	MATIC
LINK	EOS	ATOM	XLM

Table 5: Full List of Tokens

2.3 Data Pre-processing / Feature Engineering

Before getting into the model, we processed the data in the following ways:

1. Front-filling missing data (if there's any)
2. Compute log returns
3. Generate addition features (RSI, EMA, rolling-STD of different window length)
4. Split the dataset into pre-data (for cointegration test, 1 month) and data (for training and testing, remaining 9 months)

2.4 Model Design

We adopt a three-step process for our complete model. The flow of our model is as follows:

1. Use nearest-neighbour algorithm with a cross-sectional snapshot (the first observation in pre-data for all 16 tokens) to find the two nearest neighbours for each of the tokens.
2. Pair tokens up with their corresponding neighbours and conduct augmented Engle-Granger Two-Step Cointegration Test on each pair. Rank them in ascending order by the t-statistics.
3. Compute the spread and use a LSTM model in predicting the upcoming movement of the spread

[4]OHLCV is an aggregated form of market data standing for Open, High, Low, Close and Volume.

2.4.1 Nearest Neighbour

To ease our computation, instead of looping through all the combination of pairs for the 16 tokens (total of $16 \times 15 \ / \ 2 = 120$ pairs) and conduct cointegration test one by one, we form potential pairs by matching them with their two nearest neighbours. By feeding in a cross-sectional snapshot of our data into the nearest-neighbour Algorithm, we identify 38 potential cointegrated pairs, which is less than a third of the original pairs.

DOGE - ZIL	LINK - BNB	ADA - XRP
MATIC - BNB	ATOM - ZIL	LTC - DOGE

Table 6: Example of the Pairs

2.4.2 Cointegration Test

For each of the 38 potential pairs, we conduct an augmented Engle-Granger two step cointegration test which, by the name, consists of two steps.

1. Estimating the long-run equilibrium of the two series with OLS regression
2. Conduct a unit root test (Augmented Dicky Fuller Test) on the OLS residuals

The null hypothesis of the test is that there is no cointegrating relationship while the alternative hypothesis suggests that there is cointegrating relationship between the two. We reject the null hypothesis at a 5% significance level when the p-value is smaller than 0.05.

There are a total of 13 pairs that we successfully reject the null hypothesis at 5% significance level.

pairs	tstat
[EOS, ZIL]	-5.388334
[EOS, DOGE]	-5.311655
[XRP, BNB]	-4.886678
[XLM, ZIL]	-4.370721
[LINK, BNB]	-4.352107
[LTC, ZIL]	-4.210387
[ADA, LINK]	-4.151885
[DOGE, ZIL]	-4.104011
[LTC, DOGE]	-3.956726
[ADA, XRP]	-3.704049

Table 7: Sorted in Ascending Order by t-statistics

2.4.3 LSTM

LSTM is one of the neural networks that is commonly used in financial analysis. LSTM consists of four different parts of the components, namely forget gate, input gate, cell state and the output cell. The forget gate is the part which is responsible for deciding whether the information from the previous step is kept or not by using the sigmoid layer. Also, the cell state enables LSTM model to deal with a long sequence problem. Those two parts make LSTM different from other neural networks.

2.5 LSTM Model Description

2.5.1 Model Setting

Input:
S1_volume, S2_volume, S1_30_ema, S2_30_ema, S1_30_rsi, S2_30_rsi,
S1_30_vol, S2_30_vol, S1_60_ema, S2_60_ema, S1_60_rsi, S2_60_rsi,

S1_60_vol, S2_60_vol, S1_240_ema, S2_240_ema, S1_240_rsi, S2_240_rsi,
S1_240_vol, S2_240_vol, S1_480_ema, S2_480_ema, S1_480_rsi, S2_480_rsi,
S1_480_vol, S2_480_vol, S1_720_ema, S2_720_ema, S1_720_rsi, S2_720_rsi,
S1_720_vol, S2_720_vol, S1_1440_ema, S2_1440_ema, S1_1440_rsi,
S2_1440_rsi, S1_1440_vol, S2_1440_vol, Spread_open, Spread_high,
Spread_low, Spread_close (S1 stands for ZIL S2 stands for EOS)

Layers:

1. LSTM layer, 256 neurons with return sequences enabled
2. LSTM layer, 256 neurons
3. Dropout layer, rate = 0.2
4. Dense layer, output shape = 1

Loss function: mean square error (mse)
Total trainable parameters: 831,745 Total non-trainable parameters: 0

2.5.2 Optimizers Selection

As the standard deviation of the volume is larger than that of the spread of the closing price, using volume as the output variable can evaluate performance of the LSTM model. Using different optimizers can widen the performance differences. The optimizers used in the evaluation are Adam, SGD and the RMSprop. The evaluation metrics are the time taken[5] to train, the accuracy (root mean square error) of the prediction in the testing data, and the detail that can be predicted in the testing data.

1. **Adam**:
   Adam is the algorithm that combines the concept of the momentum and the RMSprop algorithm. The advantage is that since it is the upgrade version of the momentum and RMSprop, the variation when approaching to the target will be smaller and the convergence rate will be higher. However, since the number of parameters needed to calculate is greater, the time taken to do the calculation in each step should be greater than the former.

   Evaluation:
   Accuracy: 0.0324
   Time taken: 5 hours and 12 minutes

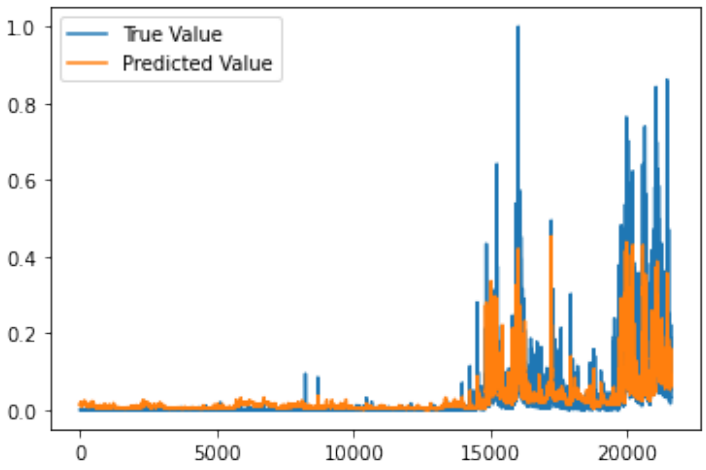

Figure 9: LSTM with Adam

[5]The model is trained solely on a CPU, 7-th generation Intel i5

Using Adam can catch the overall movement of the true value. However, it cannot catch a sudden increase between 5,000 and 10,000 time steps. The model using Adam can catch the change between 15,000 and 20,000 time steps although the predicted magnitude and the true magnitude has an observable difference. In conclusion, using Adam gives an overall nice prediction and can predict the general shape of the true value.

2. **SGD**:
Stochastic Gradient Descent is the algorithm that is the upgrade of the Gradient Descent in terms of the computational power required in each update step. It applies the theory of the Law of Large Number so that although the number of data required in each step is very small, as the iteration is large, the result will converge to a nice manner. It is a computational power focus algorithm but not a performance focus algorithm.

Evaluation:
Accuracy: 0.0461
Time taken: 6 hours and 14 minutes

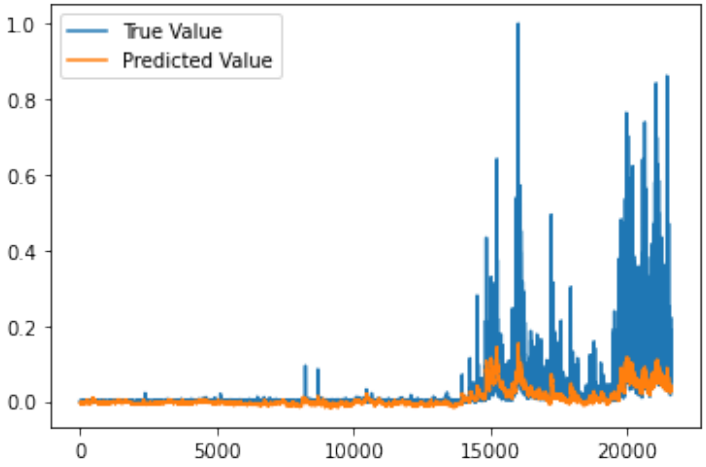

Figure 10: LSTM with SGD

The model using SGD is performing worst the using Adam. Although it still cannot catch the sudden increase between 5,000 and 10,000 time steps, the magnitude difference between the predicted value and the true value is very large between 15,000 and 20,000 time steps. The model using SGD cannot give a prediction of the overall shape of the true value. It can just tell the direction of the movement of the true value but perform poorly in magnitude. In other words, it just gives a very conservative prediction.

3. **RMSprop**:
RMSprop is the upgrade version of the momentum algorithm. It tackles the large variation problem when converging to the target and increases the convergence rate using the combined concept of the weighted mean and differentiation. Compared to the Adam, it calculated less parameter in each time steps, so the computational power required for each time step is smaller than that of the Adam.

Evaluation:
Accuracy: 0.0458
Time taken: 6 hours and 1 minute

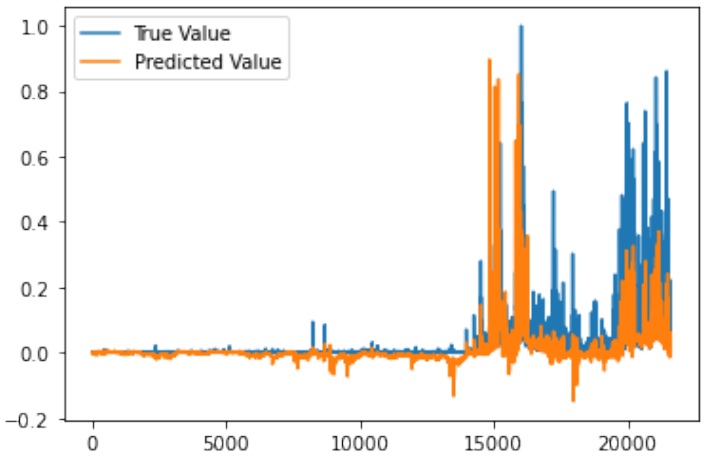

Figure 11: LSTM with RMSprop

Using RMSprop gives the predicted value in an opposite side of the true value in many parts. In the time steps between 15,000 and 20,000, it gives a negative predicted value although the true value has a great positive value. Also, it gives an opposite direction prediction in the time step between 5,000 and 10,000. However, it performs better than the other model in the second large increase of the value. The predicted magnitude is very close to the true value. In general, the model does not perform well since the shape of the predicted value deviates a lot from the true value.

After comparing three optimizers, we are going to use Adam as the optimizer for our LSTM model.

2.5.3 Final Model Training

From the previous section, the optimizer used for training the model is the Adam. The time taken for training is consistent to the previous that it takes around 5 hours. The root mean square error is 0.0025.

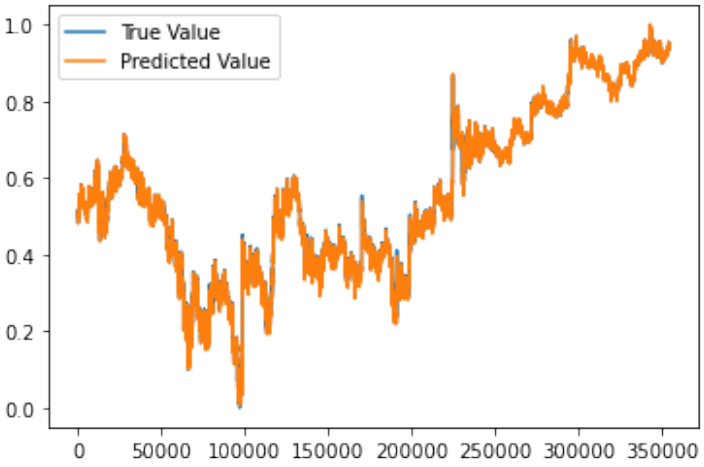

Figure 12: Final Model Training

From the graph, the model fits very well that the model gives overall shape of the training data. Therefore, the complexity is enough for the model to fit the training dataset.

2.5.4 Final Model Testing

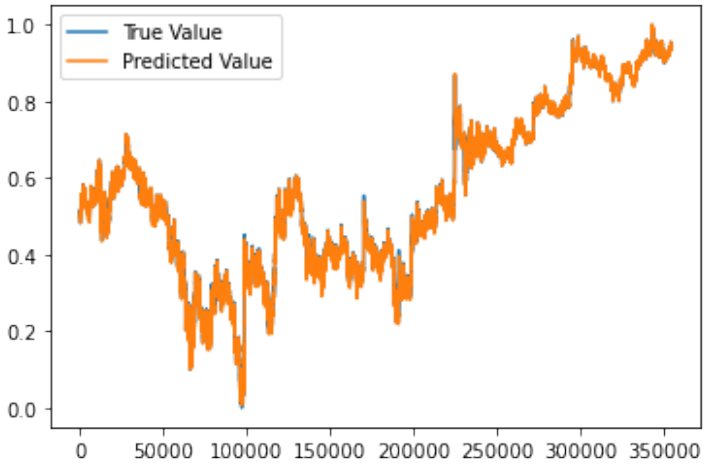

Figure 13: Final Model Testing

The root mean square error is 0.0214
From the graph, the model can predict the overall shape of the true value. The prediction deviates a little bit from the true value in the time steps around 10,000. However, the model can predict the sharp drop of the true in the time steps around 15,000. The overall performance of the model is satisfactory.

2.6 Backtesting

2.6.1 Strategy

To utilize our model prediction, the following strategy would be used. After we found the most cointegrated pair based on t-statistic, we calculate the coming 30 minutes percentage change (p) of the spread every 10 minutes and set a threshold (e) that is equal to 2 times the standard deviation of p. We buy the spread when $p > e$ and sell the spread when $p < -e$. If we are having positive position and predicting $p < -e$, we will sell all the position and sell a unit of the spread, and vice versa. The table below shows the first 3 and the last 3 trades where the return has considered the transaction cost (0.1%).

TradeID	Action	Open_time	Open_price	Close_time	Close_price	return
1	Buy	18/3/2022 11:11:00	-78.38774	18/3/2022 11:41:00	-78.425718	0.99851552
2	Sell	20/3/2022 09:01:00	-85.064713	20/3/2022 09:31:00	-83.951233	0.9859102
3	Buy	20/3/2022 09:31:00	-83.951233	20/3/2022 10:01:00	-83.98921	0.99854763
94	Buy	30/3/2022 14:41	-113.35343	30/3/2022 15:11	-111.25685	1.01749594
95	Sell	30/3/2022 15:31	-111.40513	30/3/2022 16:01	-113.0473	1.0137405
96	Sell	31/3/2022 4:01	-112.35409	31/3/2022 4:31	-112.69271	1.00201393

Table 8: Samples of Trading Detail

2.6.2 Out-of-sample Testing Performance

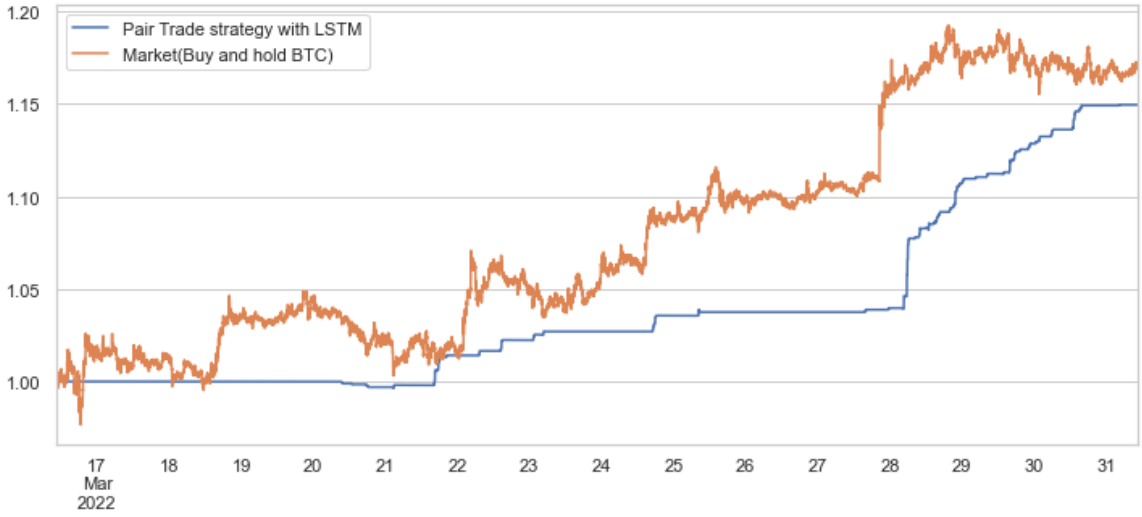

Figure 14: Pairs Trading Backtesting

	Pair Trade strategy with LSTM	Buy and Hold Bitcoin
Return in 15 days	14.9556%	17.0747%
Market (BTC) Corr. (using daily return)	0.51879	1
Daily return sd.	0.1666%	1.9125%
Annualized Sharpe (risk-free rate = 0)	1920	1009
Maximum drawdown	0.356%	11.4711%

Table 9: Portfolio Statistics of the Strategies

We set the initial capital to be equal to 1200 which enough us to buy 15 units of spread at the beginning minute. We emphasize that the equity curve above is out-of-sample and transaction cost included.

From the above table, both strategies performed very well in these 15 days. Both having a double-digit return within half of a month. It is not so surprising that Buy and Hold Bitcoin outperforms our proposed strategy. One explanation could probably be that the pair trade strategy trades frequently which cost a lot transaction fee. Yet it is a surprise that the pair trade strategy has extremely small daily return standard deviation and Max. drawdown. The return sd of this strategy is 11.5 times less than the market.

The market correlation of the proposed strategy is satisfactory. From figure 14, our strategy did not lose much when the market is not performing well during March 17 to March 18 and even make some money during March 22 to March 23 when there is a 3% drawdown in market.

2.7 Conclusion

We proposed a new perspective of constructing a market neutral strategy for algo-trading. This strategy gives a decent return with a small volatility. But since we are having an extraordinary bull market in the test period. The strategy should be further tested for a longer period. Finally, we would like to mention that our strategy is not restricted to a weekly manner nor one single pair. It can be combined with a portfolio management algorithm to diversify asset specific risk.

3 Bibliography

References

[1] Pierre-Luc Carrier and Aaron Courville. Facial expression recognition challenge. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data, 2013. Accessed: 2022-04-18.

[2] ipacz. mtcnn. https://github.com/ipazc/mtcnn, 2016. Accessed: 2022-04-16.

[3] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. 2016.

[4] Simerjot Kaur. Quantitative trading strategies using deep learning: Pairs trading. https://cs230.stanford.edu/projects_fall_2018/reports/12446738.pdf, 2018.