# Lending Club Loan Default Detection

# CONTENT

## 01 Project Overview

- Project Introduction
- Data Overview

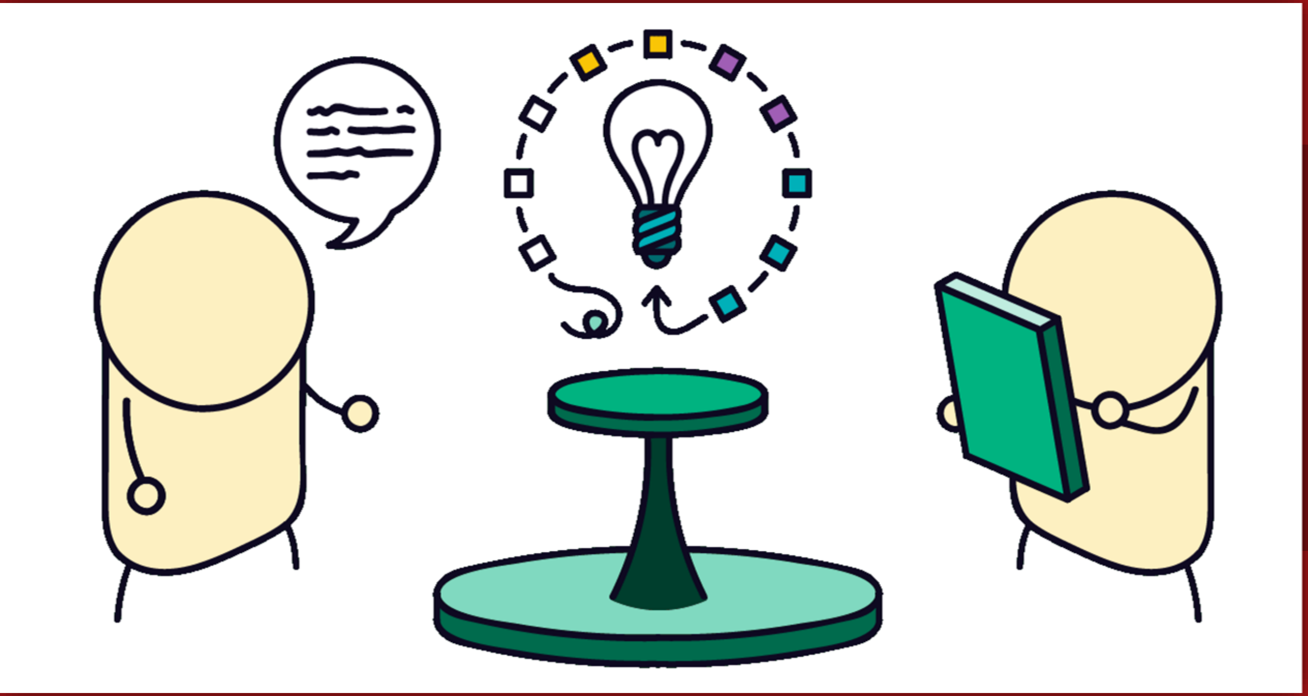
## 02 Data Preprocessing

## 03 Loan Default Prediction (modeling)

- Supervised Learning: Probability of Default Estimation
- Self-supervised Learning: Anomaly Detection

## 04 Model Performance Comparison

## 05 Conclusions



# 1 Project Overview

Project Introduction

Problem Description

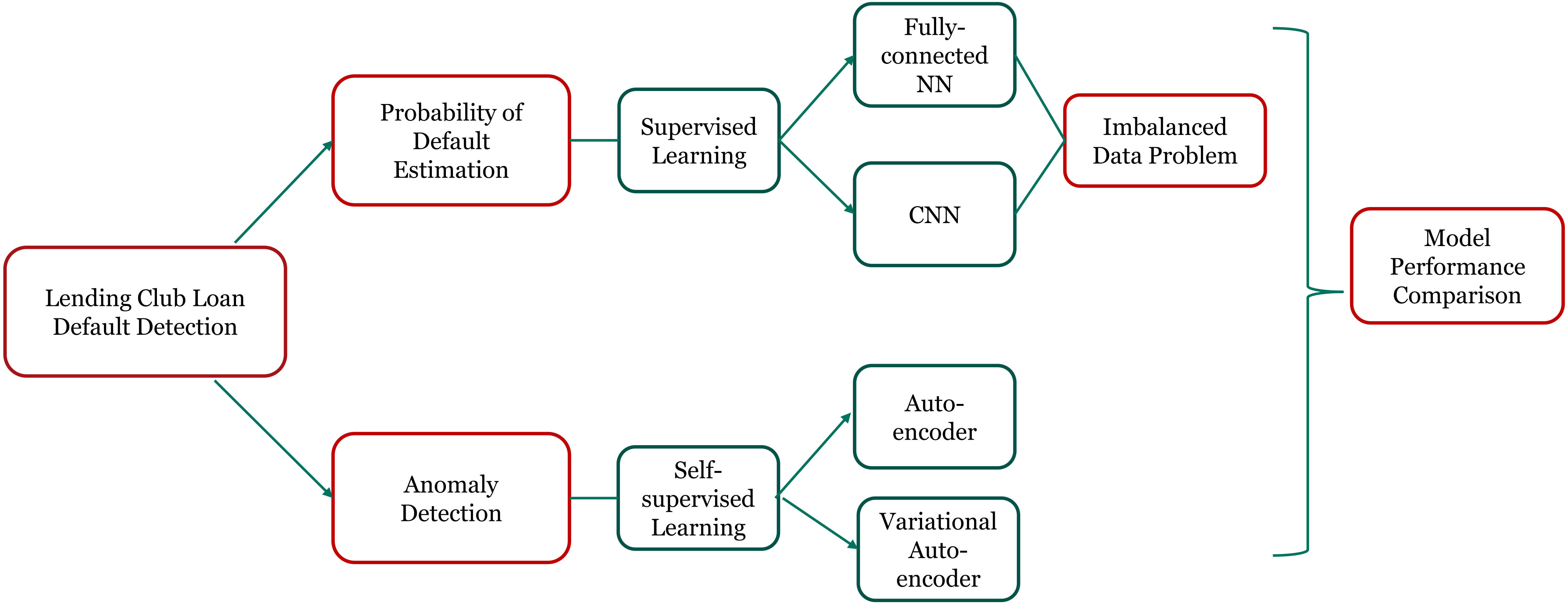
Data Overview

# 1.1 Project Introduction

- Peer to peer lending & financial inclusion

- Objectives: Loan repayment defaulter detection

# 1.2 Problems Description

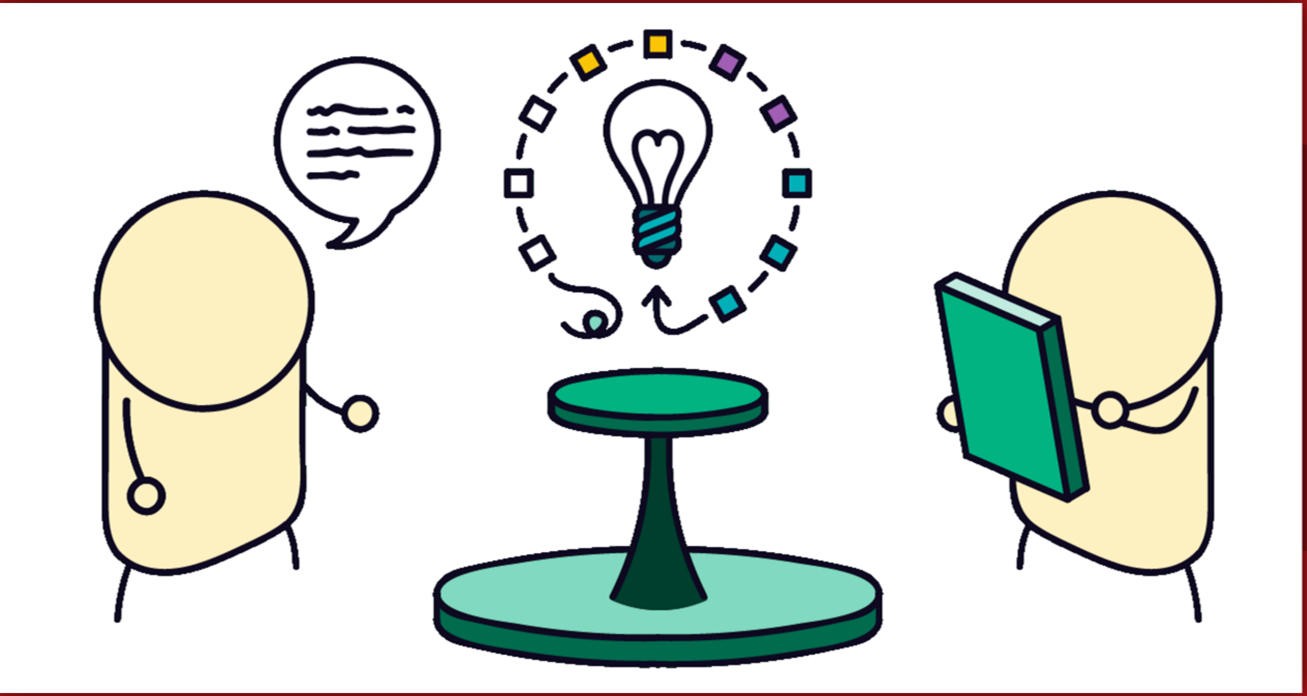


# 1.3 Database Introduction

LendingClub Issued Loans Data: https://www.kaggle.com/husainsb/lendingclub-issued-loans

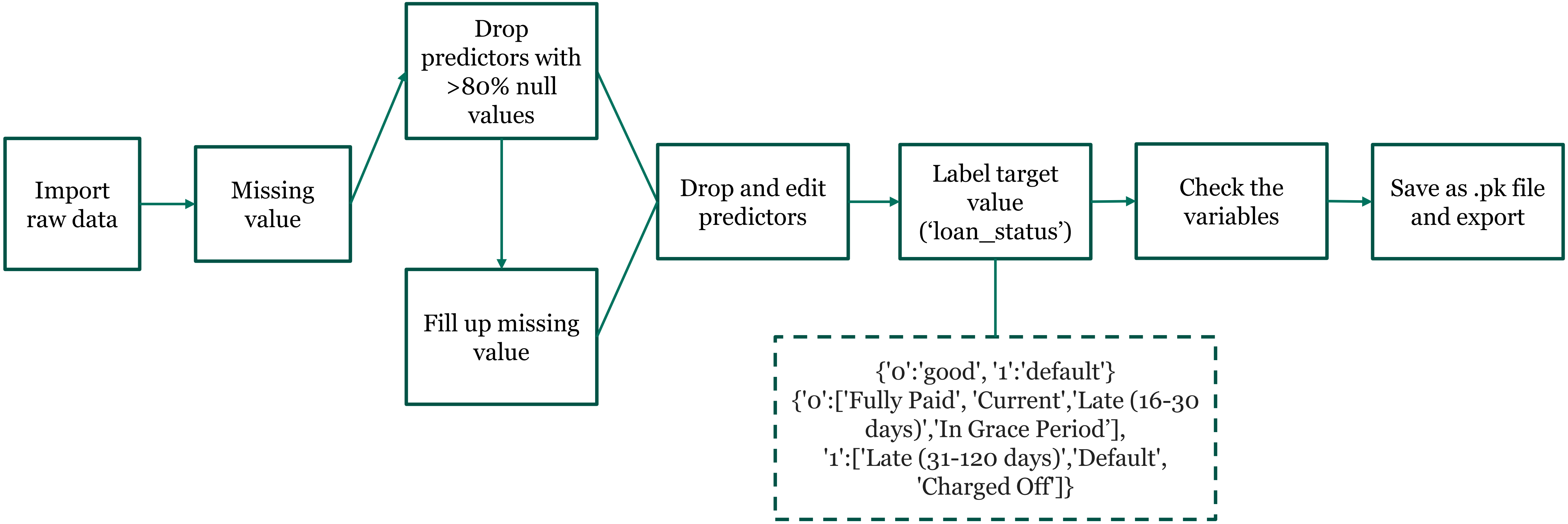
- Training Data set: lc_loan.csv
  Contains loans issued from 2007-2015
  74 columns

- Test Data set: lc_2016_2017.csv
  Contains loans issued from 2016-2017
  72 columns (missing the columns of 'open_il_6m', 'url’)



# 2 Data preprocessing

# Data Preprocessing



Import raw data

Missing value

Drop predictors with >80% null values

Fill up missing value

Drop and edit predictors

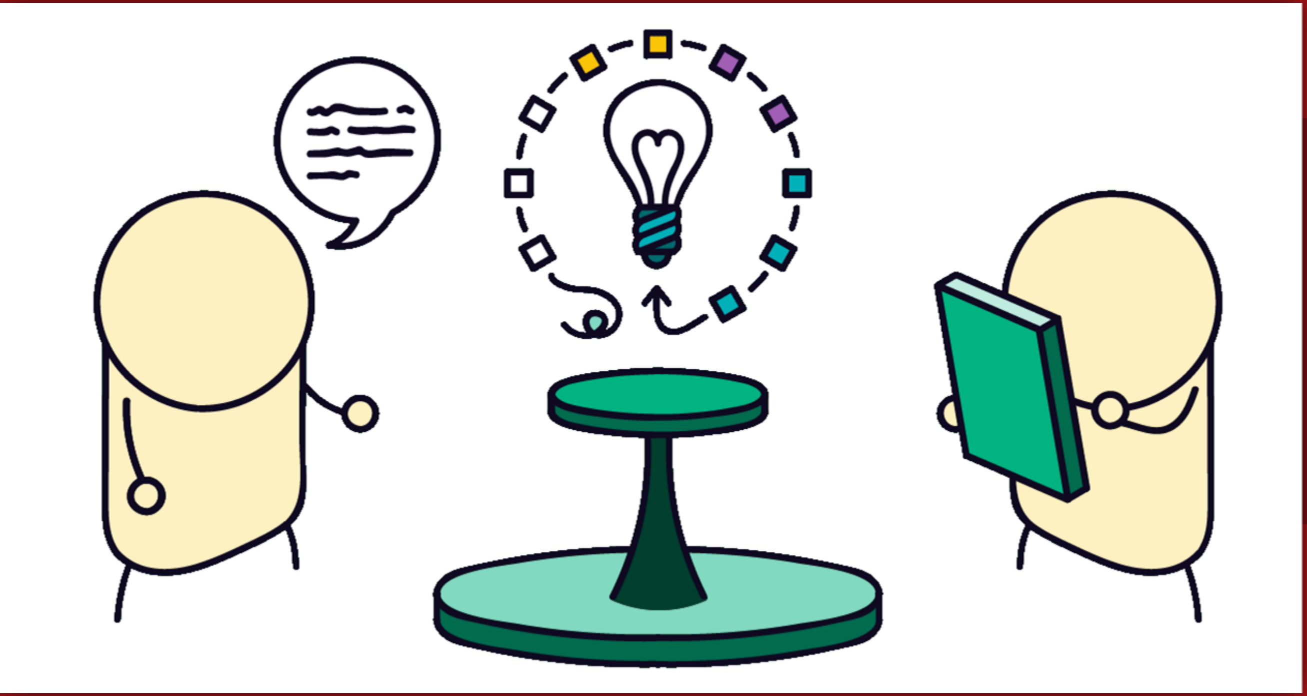
Label target value (‘loan_status’)

Check the variables

Save as .pk file and export

{'0':'good', '1':'default'}
{'0':['Fully Paid', 'Current','Late (16-30 days)','In Grace Period’],
'1':['Late (31-120 days)','Default', 'Charged Off']}

- Predictors (X): from 73 to 41
- Target Variables (Y): {'0':'good', '1':'default'}



# 3 Modeling -- Loan Default Prediction

1) Supervised Learning
2) Unsupervised Learning

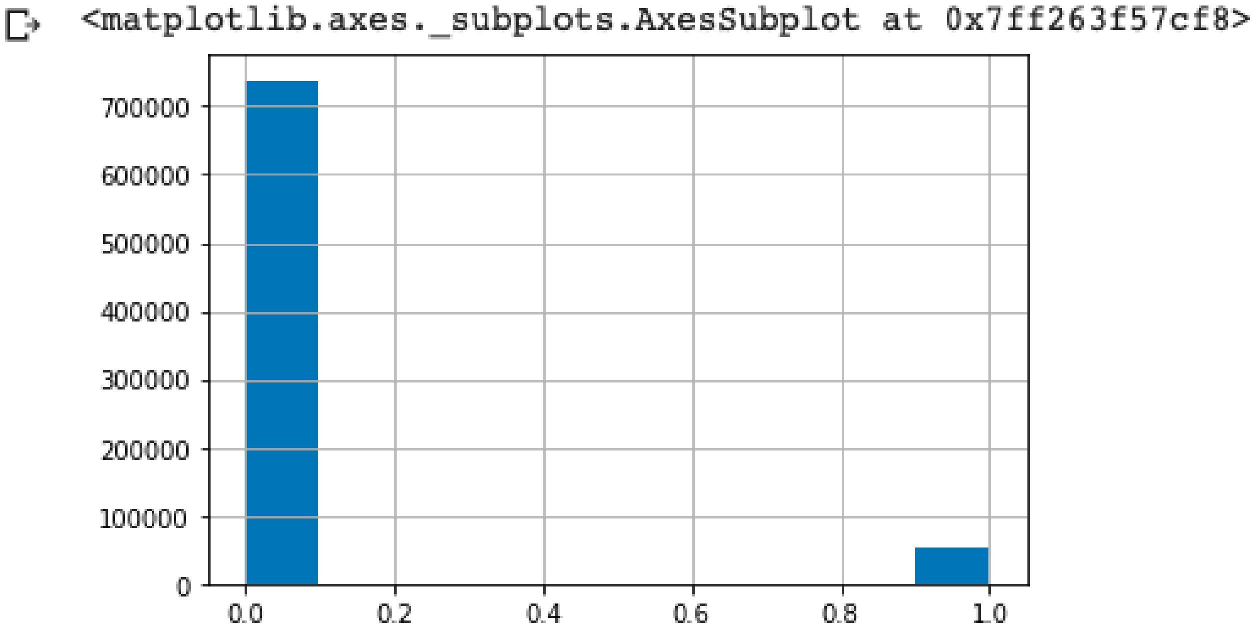
# 3.1 Probability of Default Estimation

# -- Fully-connected Neural Network

- **Supervised Learning**
- **Fully-connected Neural Network**
- **Undersampling**
- **Oversampling**

# Problem of Imbalanced Classification

- Target column -- **“Loan_Status”** 0 is good 1 is default
- The number of default transactions in training data is **52937** and number of transactions which do not default is **738087**;
- The number of default transactions in testing data is **52564** and number of transactions which do not default is **706270**;

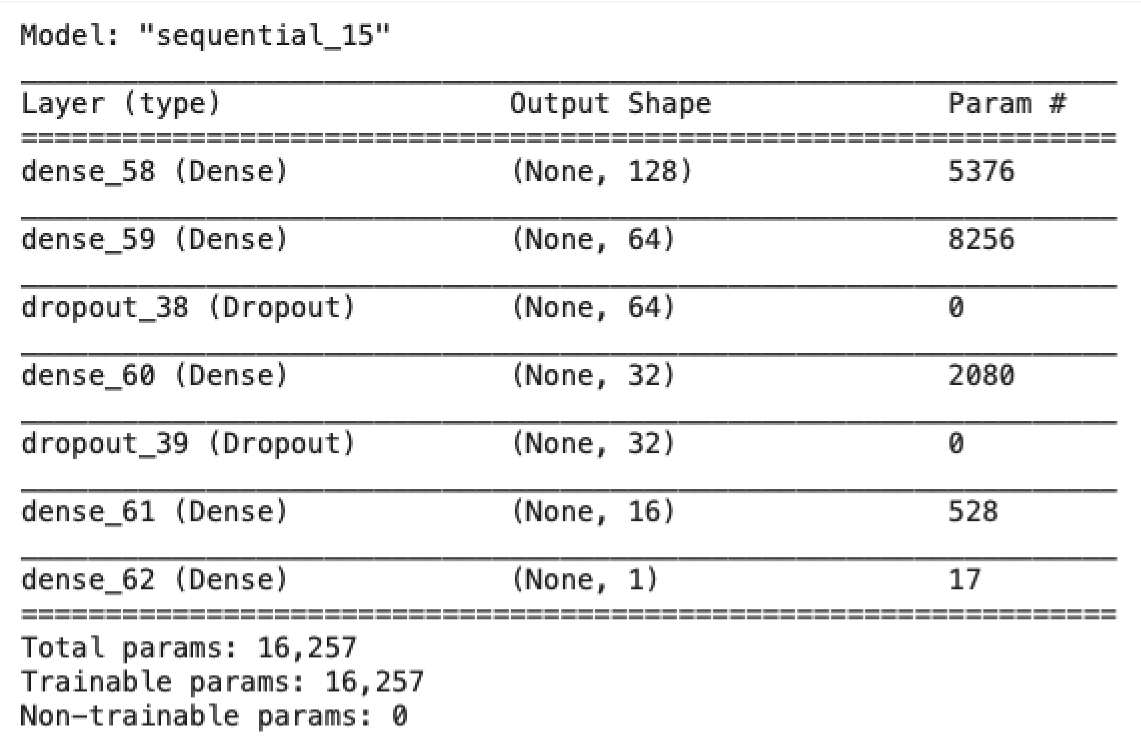


```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff263f57cf8>
```

# 1 Basic Model -- Fully-connected Neural Network

## 1.1 Modeling

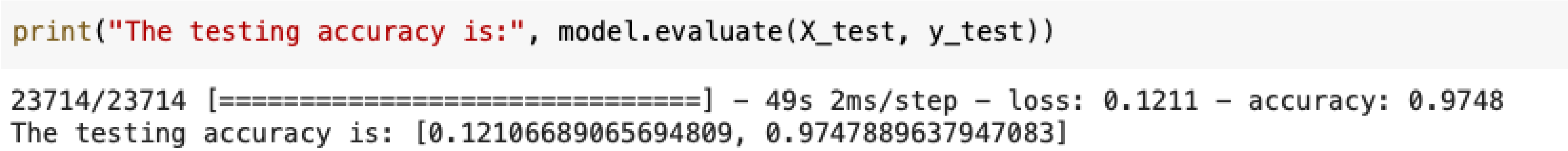
- Standardization
- Dropout
- Regularization
- Change architecture
- Early Stopping



```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_58 (Dense)             (None, 128)               5376      
_________________________________________________________________
dense_59 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_38 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_60 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_39 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_61 (Dense)             (None, 16)                528       
_________________________________________________________________
dense_62 (Dense)             (None, 1)                 17        
=================================================================
Total params: 16,257
Trainable params: 16,257
Non-trainable params: 0
```

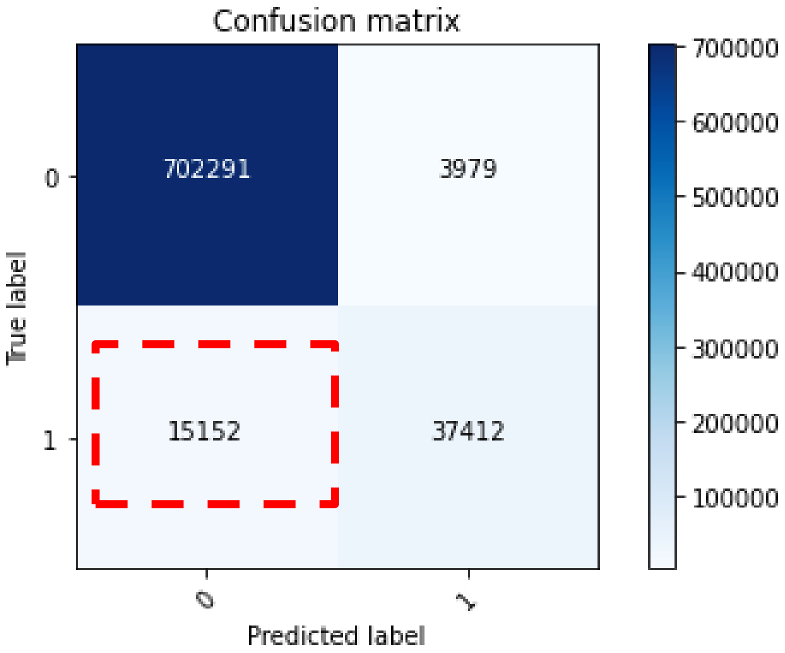
# 1 Basic Model -- Fully-connected Neural Network

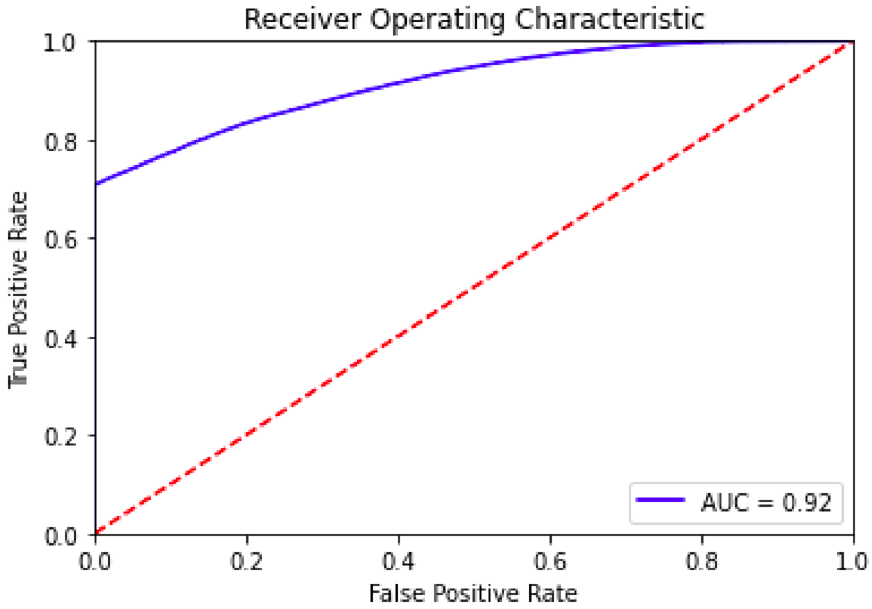
## 1.2 Evaluation



```
print("The testing accuracy is:", model.evaluate(X_test, y_test))
```

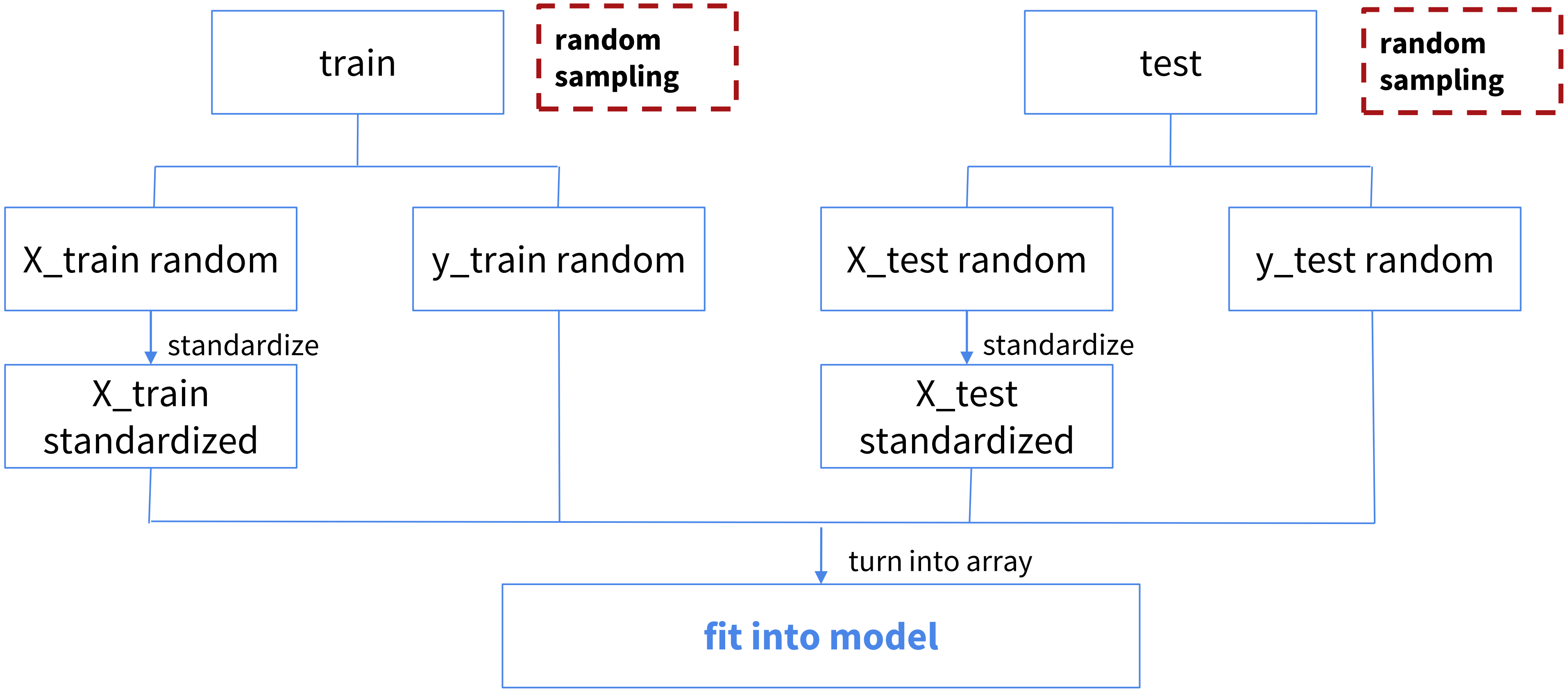
```
23714/23714 [==============================] - 49s 2ms/step - loss: 0.1211 - accuracy: 0.9748
The testing accuracy is: [0.12106689065694809, 0.9747889637947083]
```





# 2 Undersampling

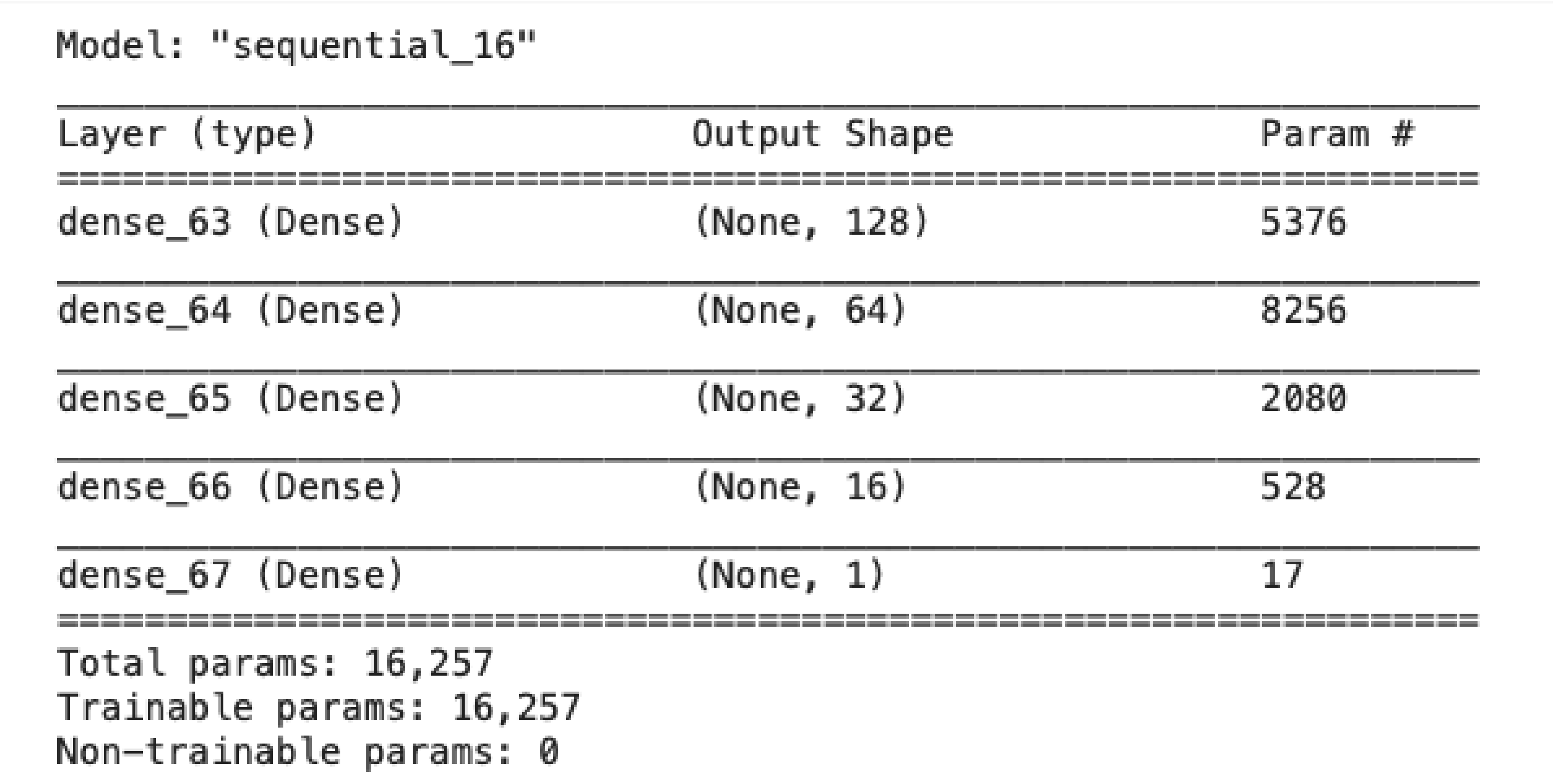
- reducing the data by eliminating examples belonging to the majority class



# 2 Undersampling

## 2.1 Modeling

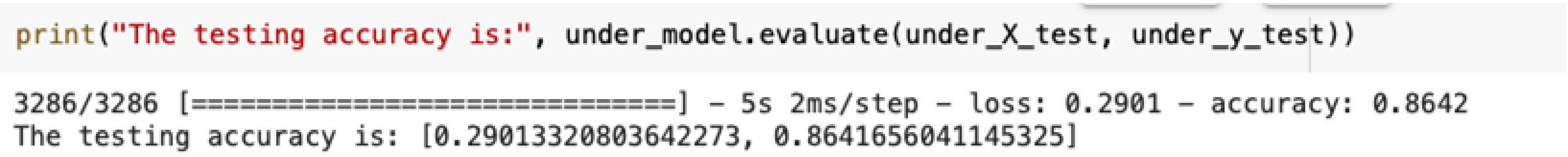
- Standardization
- Regularization
- Change architecture
- Early Stopping



```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 128)               5376
_________________________________________________________________
dense_64 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_65 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_66 (Dense)             (None, 16)                528
_________________________________________________________________
dense_67 (Dense)             (None, 1)                 17
=================================================================
Total params: 16,257
Trainable params: 16,257
Non-trainable params: 0
```

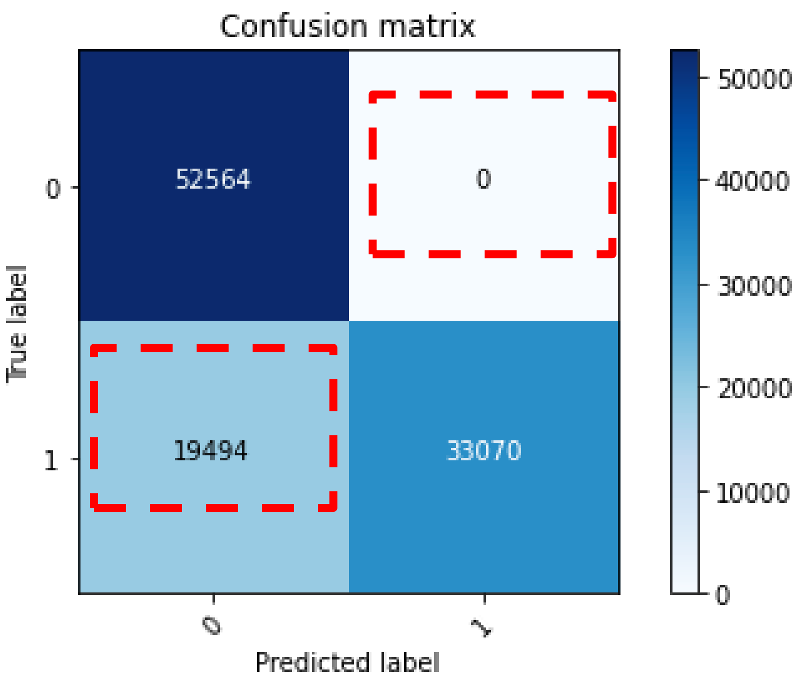
# 2 Undersampling

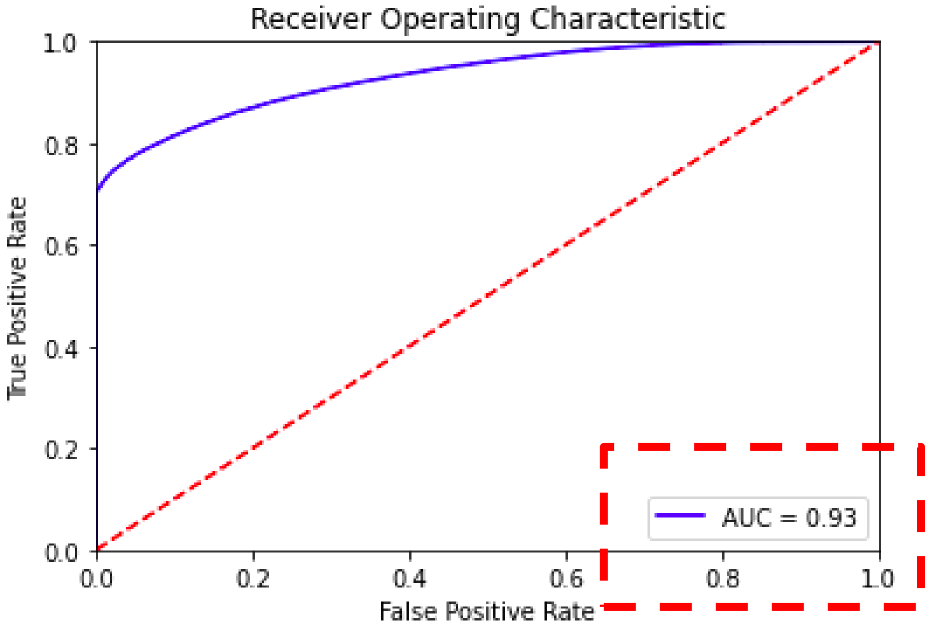
## 2.2 Evaluation



```
print("The testing accuracy is:", under_model.evaluate(under_X_test, under_y_test))
```

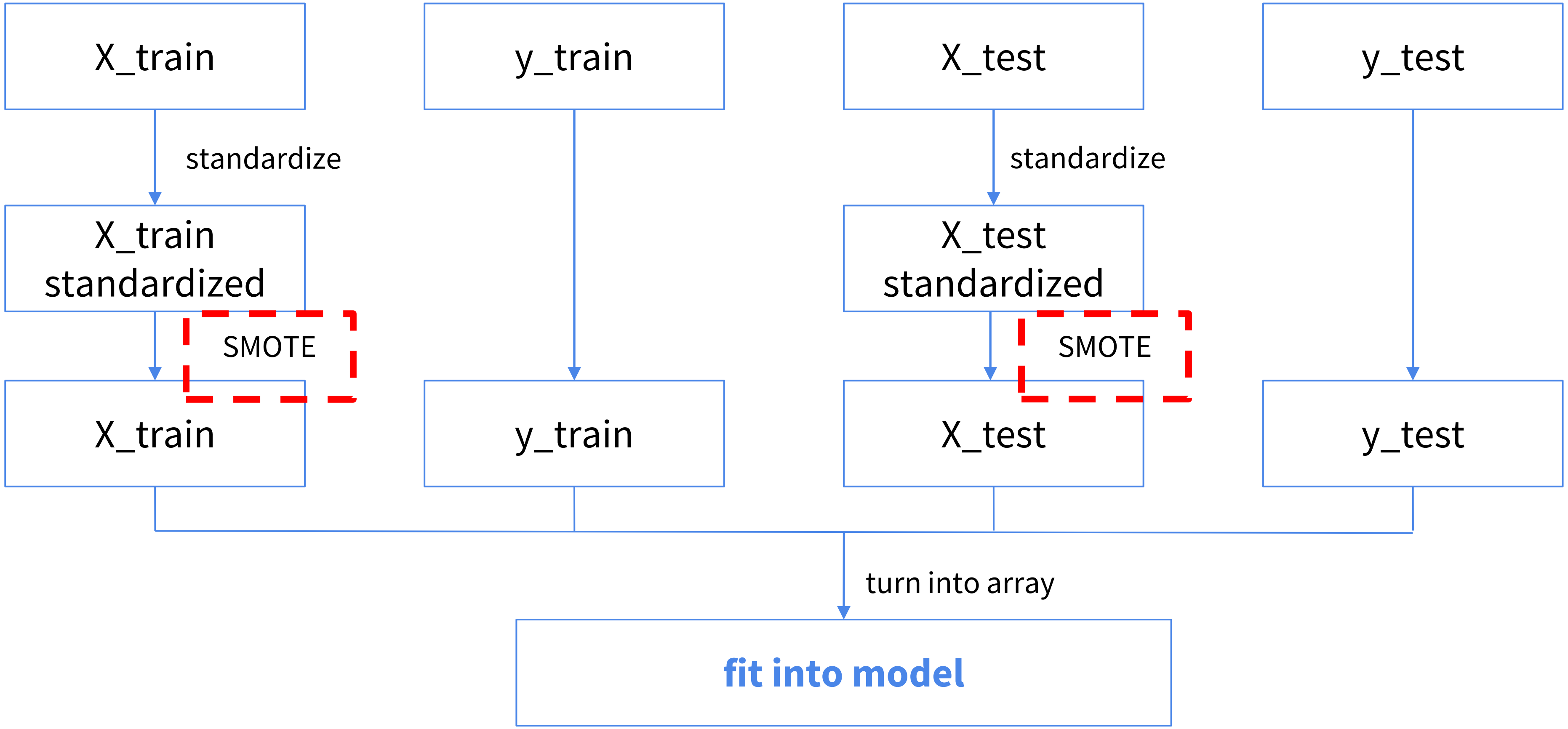
```
3286/3286 [==============================] - 5s 2ms/step - loss: 0.2901 - accuracy: 0.8642
The testing accuracy is: [0.29013320803642273, 0.8641656041145325]
```





# 3 Oversampling

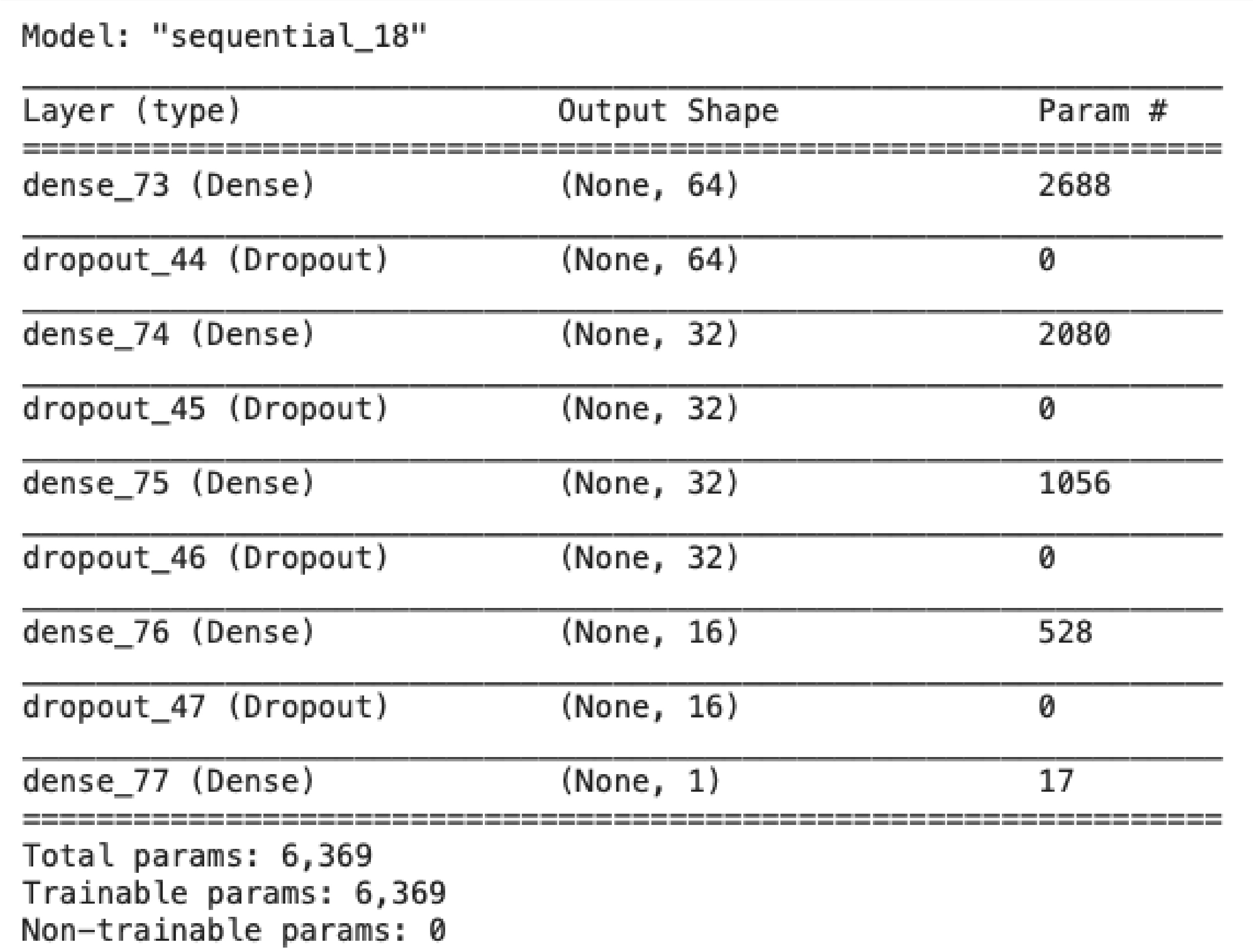
- selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line -- increase the percentage of minority class



# 3 Oversampling

## 3.1 Modeling

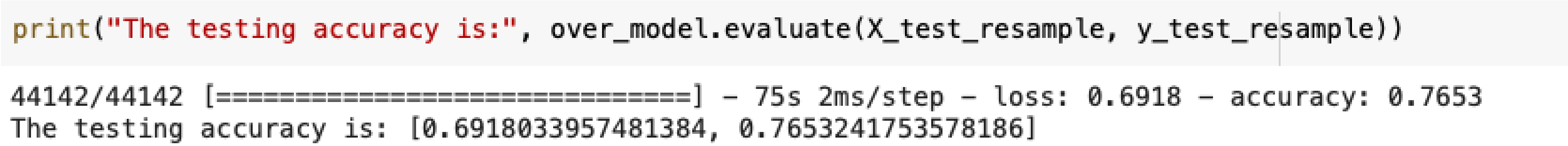
- Standardization
- Dropout
- Regularization
- Change architecture
- Early Stopping



```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_73 (Dense)             (None, 64)                2688      
_________________________________________________________________
dropout_44 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_74 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_45 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_75 (Dense)             (None, 32)                1056      
_________________________________________________________________
dropout_46 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_76 (Dense)             (None, 16)                528       
_________________________________________________________________
dropout_47 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_77 (Dense)             (None, 1)                 17        
=================================================================
Total params: 6,369
Trainable params: 6,369
Non-trainable params: 0
```

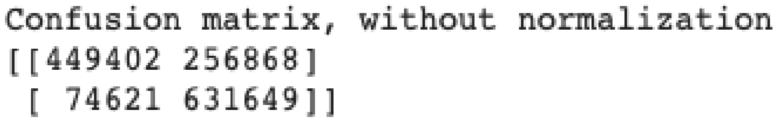
# 3 Oversampling

## 3.2 Evaluation

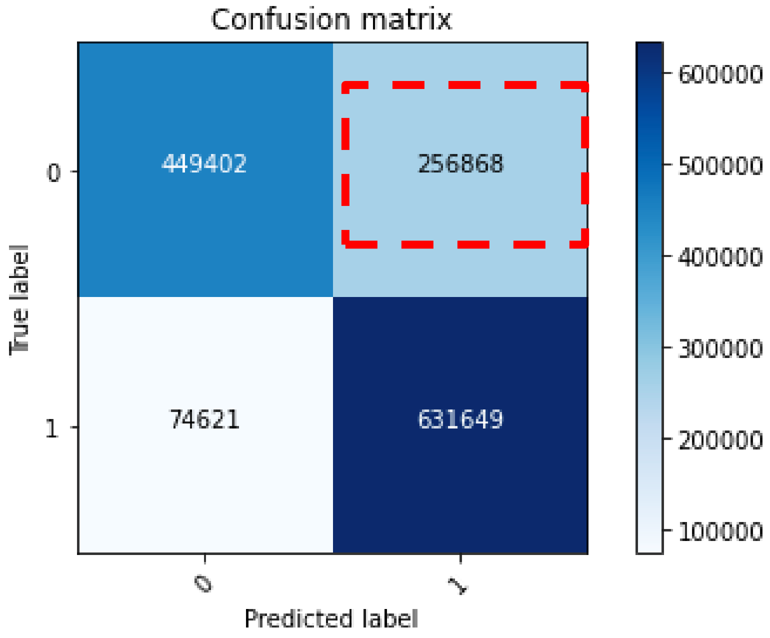


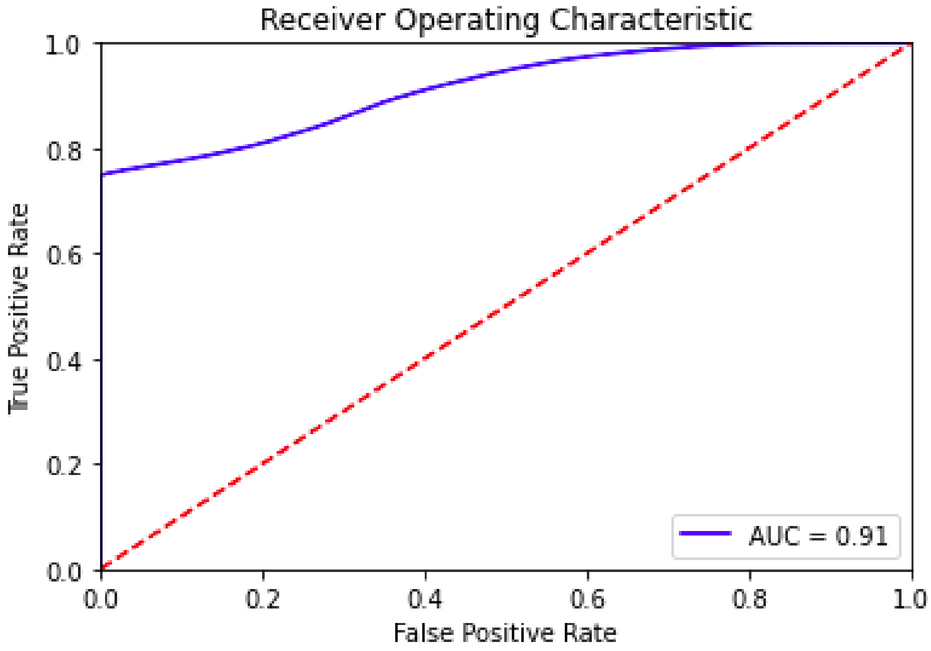
```
print("The testing accuracy is:", over_model.evaluate(X_test_resample, y_test_resample))
```

```
44142/44142 [==============================] - 75s 2ms/step - loss: 0.6918 - accuracy: 0.7653
The testing accuracy is: [0.6918033957481384, 0.7653241753578186]
```



```
Confusion matrix, without normalization
[[449402 256868]
 [ 74621 631649]]
```





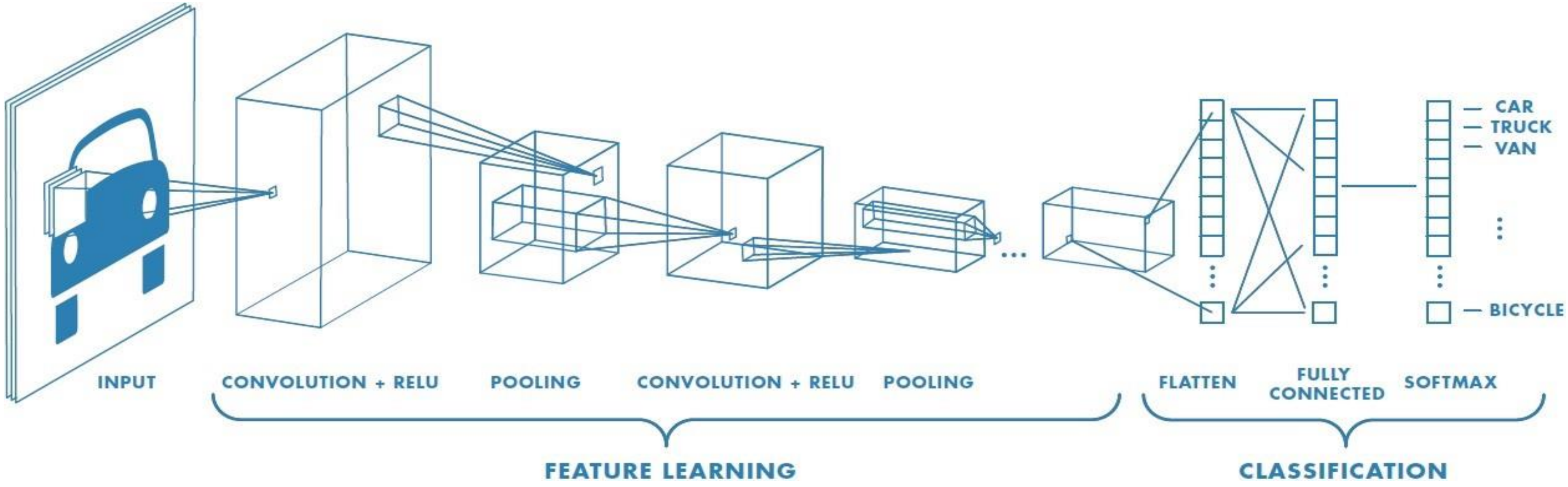
# 4 Conclusion

- Utilizing dropout, regularization, change architecture, and early stopping, overfitting problem can be solved well
- Fully connected neural model has a good accuracy
- Undersampling would ease imbalance problem
- Oversampling might not be appropriate to solve the problem in this dataset

# 3.2 Probability of Default Estimation -- Convolutional Neural Network

- **Convolutional Neural Network**
- **Undersampling**
- **Oversampling**

# Convolutional Neural Network Overview

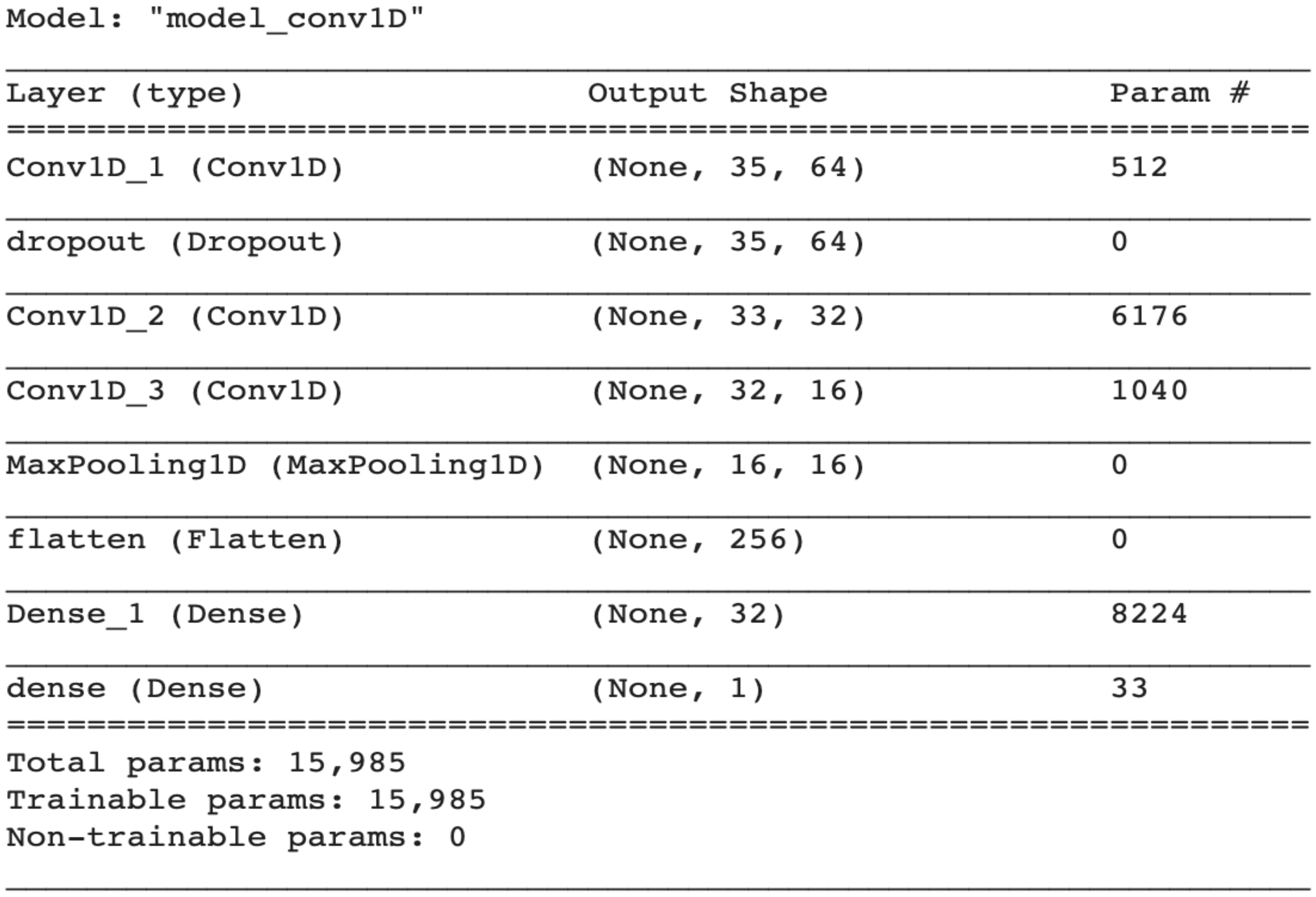


- One of the main categories to do images recognition, images classifications

- Also can be used for non-image data set by reshaping the data with the input dimension = 1

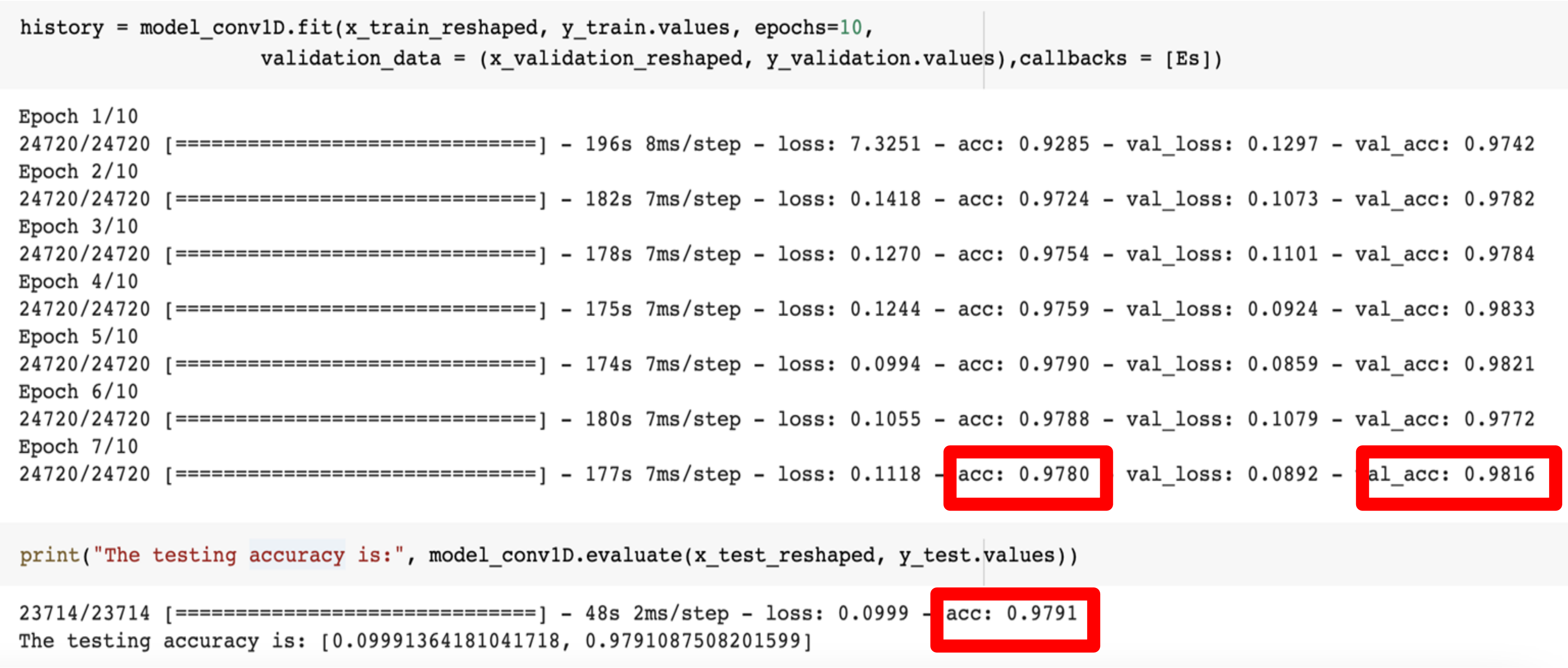
# 1 Modeling

- Convolutional layer
- Dropout
- MaxPooling
- Flatten
- Early Stopping



```
Model: "model_conv1D"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv1D_1 (Conv1D)            (None, 35, 64)            512
_________________________________________________________________
dropout (Dropout)            (None, 35, 64)            0
_________________________________________________________________
Conv1D_2 (Conv1D)            (None, 33, 32)            6176
_________________________________________________________________
Conv1D_3 (Conv1D)            (None, 32, 16)            1040
_________________________________________________________________
MaxPooling1D (MaxPooling1D)  (None, 16, 16)            0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
Dense_1 (Dense)              (None, 32)                8224
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 15,985
Trainable params: 15,985
Non-trainable params: 0
_________________________________________________________________
```

# 2 Evaluate



```
history = model_conv1D.fit(x_train_reshaped, y_train.values, epochs=10,
                    validation_data = (x_validation_reshaped, y_validation.values),callbacks = [Es])
```

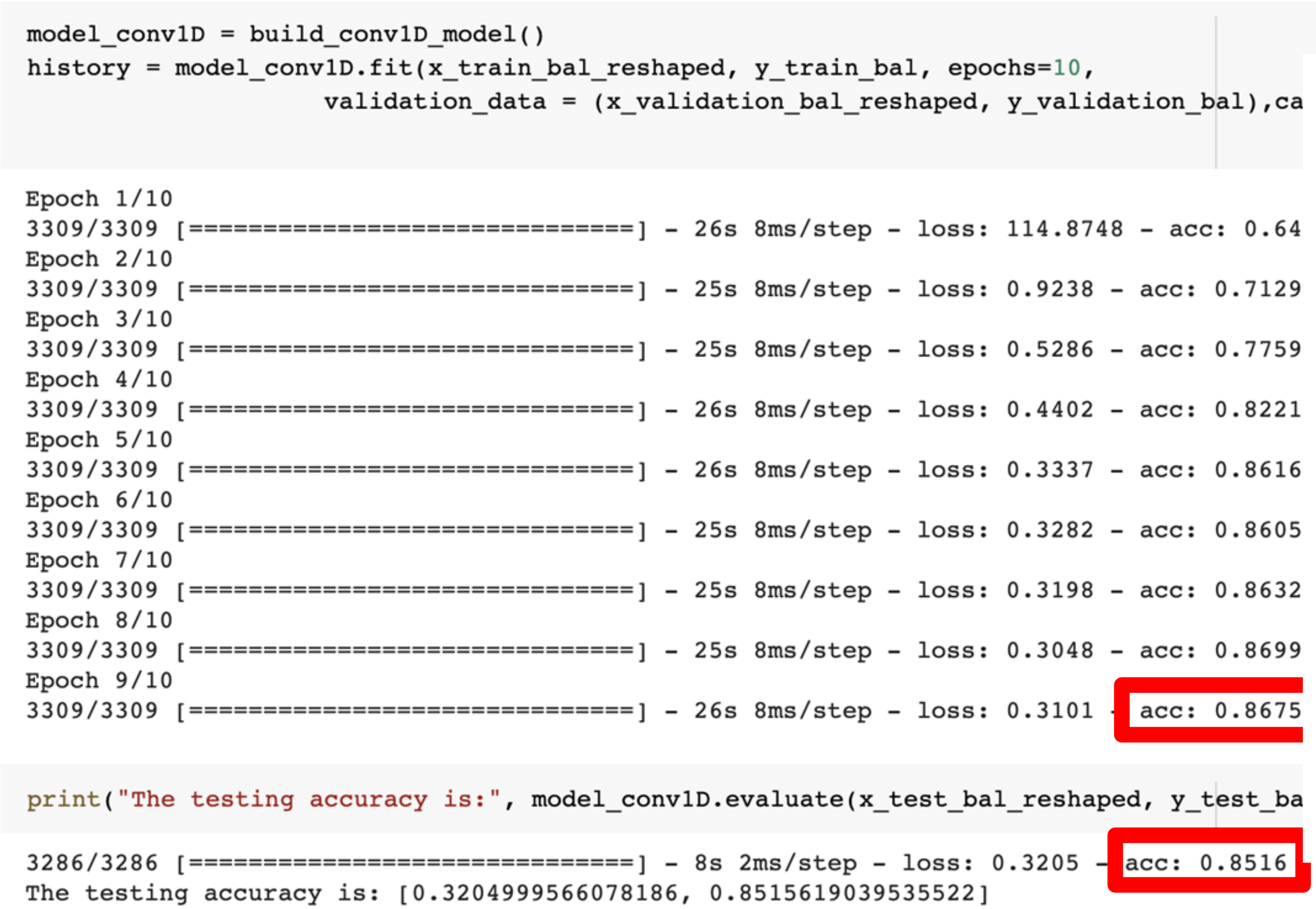
```
Epoch 1/10
24720/24720 [==============================] - 196s 8ms/step - loss: 7.3251 - acc: 0.9285 - val_loss: 0.1297 - val_acc: 0.9742
Epoch 2/10
24720/24720 [==============================] - 182s 7ms/step - loss: 0.1418 - acc: 0.9724 - val_loss: 0.1073 - val_acc: 0.9782
Epoch 3/10
24720/24720 [==============================] - 178s 7ms/step - loss: 0.1270 - acc: 0.9754 - val_loss: 0.1101 - val_acc: 0.9784
Epoch 4/10
24720/24720 [==============================] - 175s 7ms/step - loss: 0.1244 - acc: 0.9759 - val_loss: 0.0924 - val_acc: 0.9833
Epoch 5/10
24720/24720 [==============================] - 174s 7ms/step - loss: 0.0994 - acc: 0.9790 - val_loss: 0.0859 - val_acc: 0.9821
Epoch 6/10
24720/24720 [==============================] - 180s 7ms/step - loss: 0.1055 - acc: 0.9788 - val_loss: 0.1079 - val_acc: 0.9772
Epoch 7/10
24720/24720 [==============================] - 177s 7ms/step - loss: 0.1118 - acc: 0.9780 - val_loss: 0.0892 - val_acc: 0.9816
```

```
print("The testing accuracy is:", model_conv1D.evaluate(x_test_reshaped, y_test.values))
```

```
23714/23714 [==============================] - 48s 2ms/step - loss: 0.0999 - acc: 0.9791
The testing accuracy is: [0.09991364181041718, 0.9791087508201599]
```

- High accuracy with no overfitting
- But,  data imbalance might lead to the “perfect” outcome
  → **Need oversample/undersample**

# 3 Undersampling

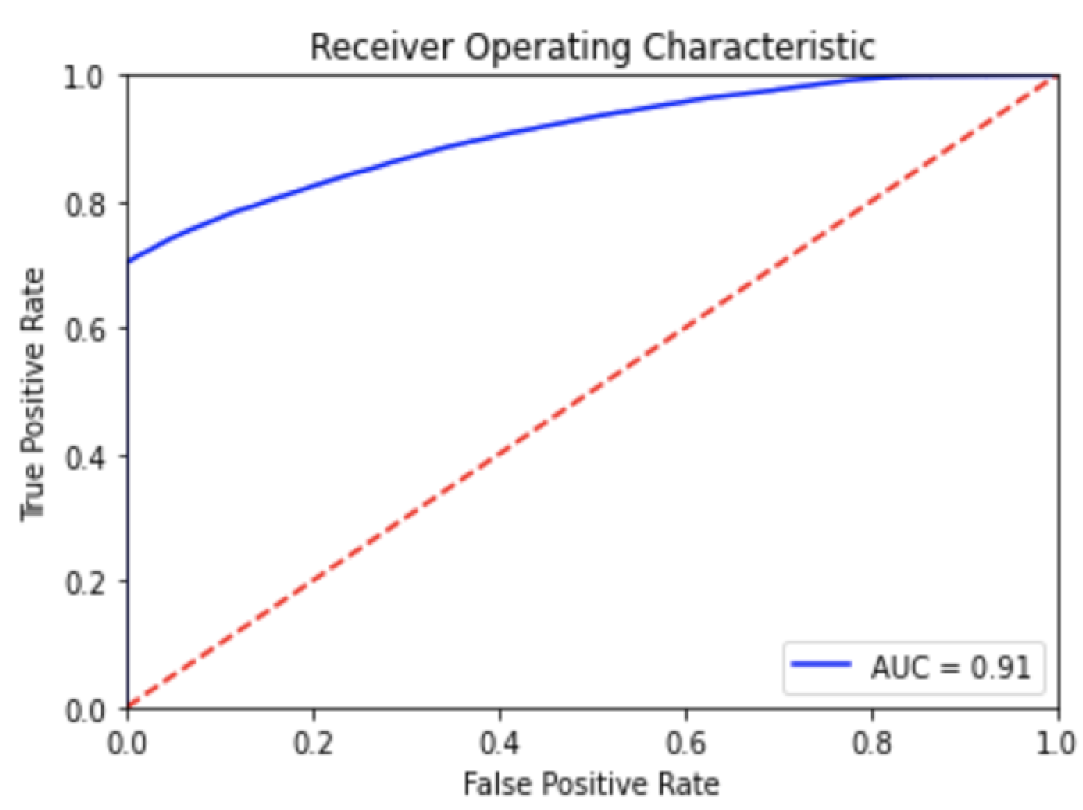


```
model_conv1D = build_conv1D_model()
history = model_conv1D.fit(x_train_bal_reshaped, y_train_bal, epochs=10,
                    validation_data = (x_validation_bal_reshaped, y_validation_bal),ca
```

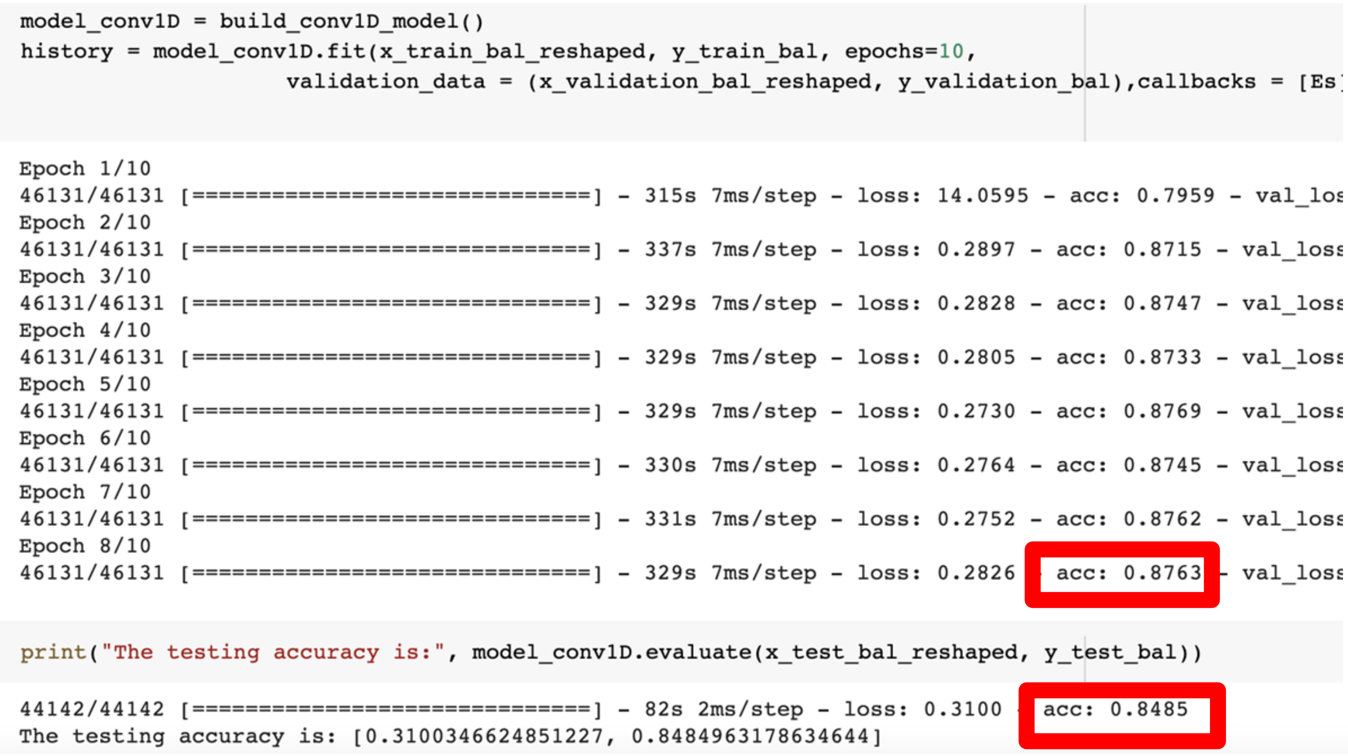
```
Epoch 1/10
3309/3309 [==============================] - 26s 8ms/step - loss: 114.8748 - acc: 0.64
Epoch 2/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.9238 - acc: 0.7129
Epoch 3/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.5286 - acc: 0.7759
Epoch 4/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.4402 - acc: 0.8221
Epoch 5/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.3337 - acc: 0.8616
Epoch 6/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3282 - acc: 0.8605
Epoch 7/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3198 - acc: 0.8632
Epoch 8/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3048 - acc: 0.8699
Epoch 9/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.3101 - acc: 0.8675
```

```
print("The testing accuracy is:", model_conv1D.evaluate(x_test_bal_reshaped, y_test_ba
```

```
3286/3286 [==============================] - 8s 2ms/step - loss: 0.3205 - acc: 0.8516
The testing accuracy is: [0.3204999566078186, 0.8515619039535522]
```



# 4 Oversampling

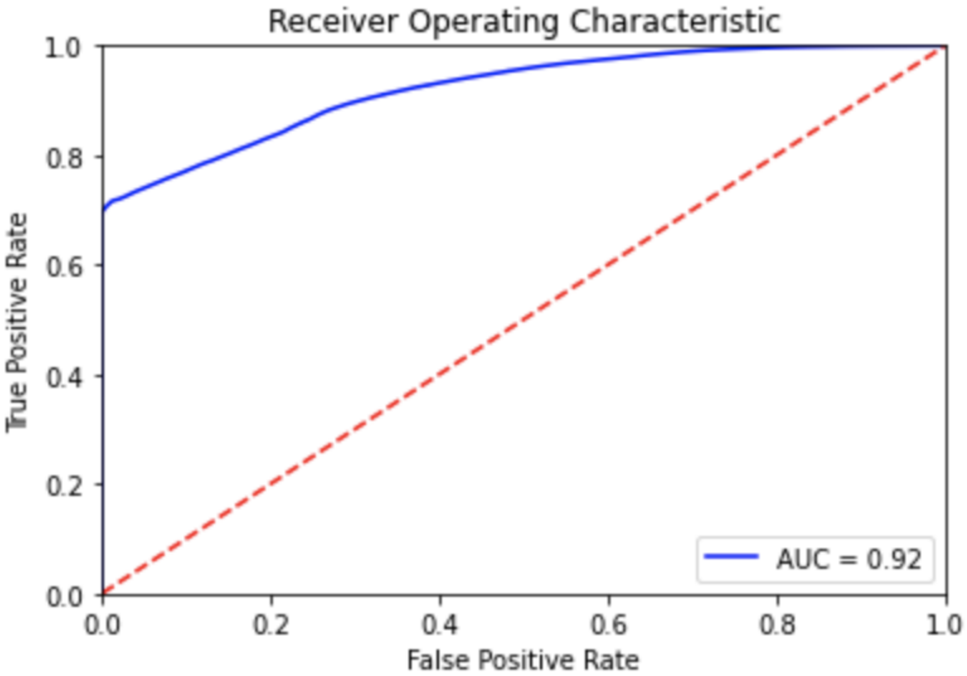


```
model_conv1D = build_conv1D_model()
history = model_conv1D.fit(x_train_bal_reshaped, y_train_bal, epochs=10,
                   validation_data = (x_validation_bal_reshaped, y_validation_bal),callbacks = [Es
```

```
Epoch 1/10
46131/46131 [==============================] - 315s 7ms/step - loss: 14.0595 - acc: 0.7959 - val_los
Epoch 2/10
46131/46131 [==============================] - 337s 7ms/step - loss: 0.2897 - acc: 0.8715 - val_loss
Epoch 3/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2828 - acc: 0.8747 - val_loss
Epoch 4/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2805 - acc: 0.8733 - val_loss
Epoch 5/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2730 - acc: 0.8769 - val_loss
Epoch 6/10
46131/46131 [==============================] - 330s 7ms/step - loss: 0.2764 - acc: 0.8745 - val_loss
Epoch 7/10
46131/46131 [==============================] - 331s 7ms/step - loss: 0.2752 - acc: 0.8762 - val_loss
Epoch 8/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2826 - acc: 0.8763 - val_loss
```

```
print("The testing accuracy is:", model_conv1D.evaluate(x_test_bal_reshaped, y_test_bal))
```

```
44142/44142 [==============================] - 82s 2ms/step - loss: 0.3100 - acc: 0.8485
The testing accuracy is: [0.3100346624851227, 0.8484963178634644]
```



# 5 Conclusion

- Though CNN result for original data works pretty well, it cannot be taken into account due to imbalance data set.

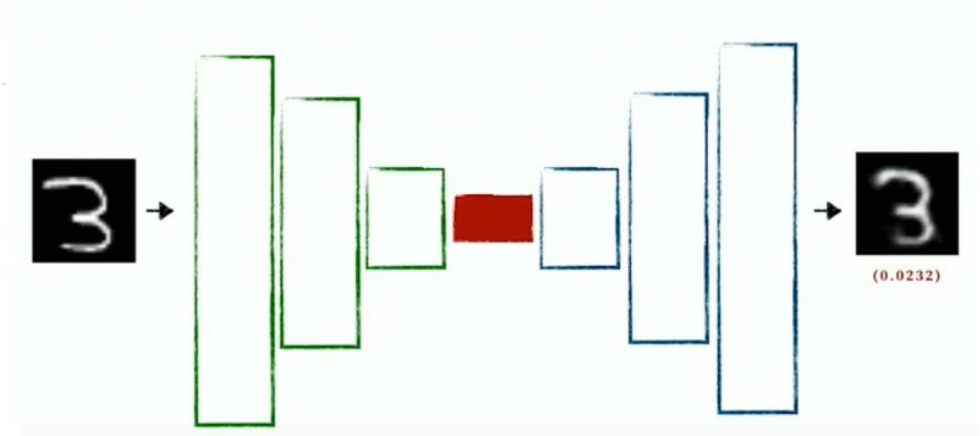
- Oversampled data performs better with the CNN model in both accuracy and AUC/ROC curve.

# 3.3 Anomaly Detection
# -- self-supervised Learning

- **Anomaly Detection**
- **Autoencoder**
- **Variational autoencoder**

# 1 Anomaly Detection

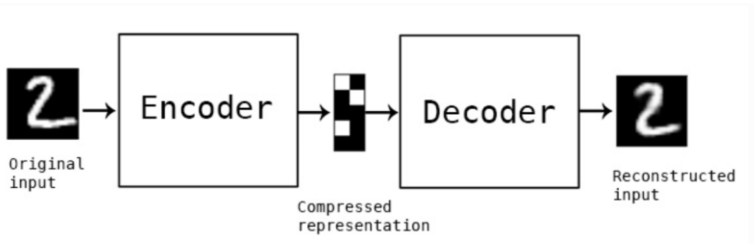
## Self-supervised Learning -- Autoencoder/VAE



- Train model using **normal training data set**
- Find the threshold of the anomaly using reconstruction error (mean-squared-error)
- If reconstruction error > threshold, it’s an anomaly

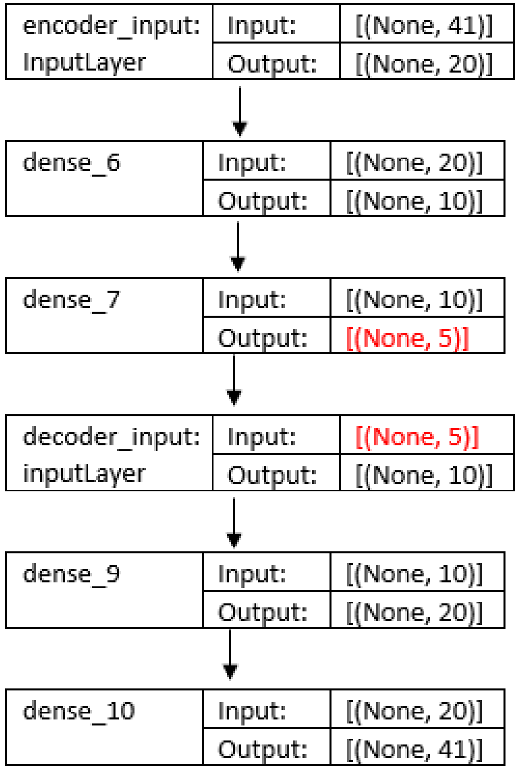
# 2 Autoencoder

## Modeling



Parameter: loss='MSE', optimizer='adam'

### Basic Model



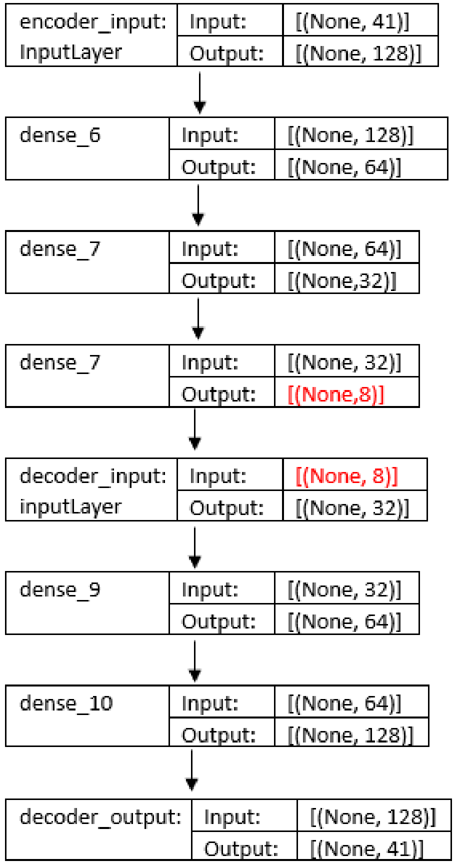
| | | |
|---|---|---|
| encoder_input: InputLayer | Input: | [(None, 41)] |
| | Output: | [(None, 20)] |
| dense_6 | Input: | [(None, 20)] |
| | Output: | [(None, 10)] |
| dense_7 | Input: | [(None, 10)] |
| | Output: | [(None, 5)] |
| decoder_input: inputLayer | Input: | [(None, 5)] |
| | Output: | [(None, 10)] |
| dense_9 | Input: | [(None, 10)] |
| | Output: | [(None, 20)] |
| dense_10 | Input: | [(None, 20)] |
| | Output: | [(None, 41)] |

val_loss : 0.0032
val_accuracy: 0.5886

- Change architectures
  - Bottleneck
  - Hidden layers
- Parameters
  - Loss function
- Early Stopping

Model Improvement

### Improved Model



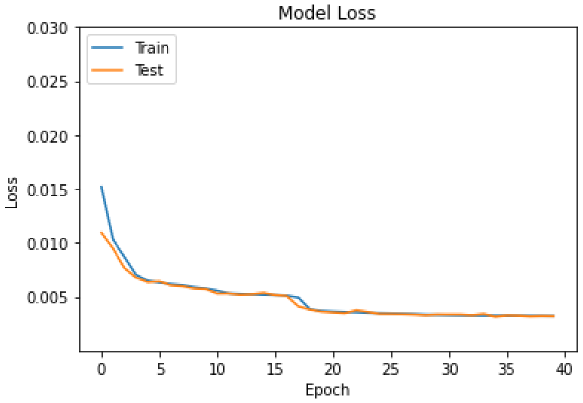
| | | |
|---|---|---|
| encoder_input: InputLayer | Input: | [(None, 41)] |
| | Output: | [(None, 128)] |
| dense_6 | Input: | [(None, 128)] |
| | Output: | [(None, 64)] |
| dense_7 | Input: | [(None, 64)] |
| | Output: | [(None,32)] |
| dense_7 | Input: | [(None, 32)] |
| | Output: | [(None,8)] |
| decoder_input: inputLayer | Input: | [(None, 8)] |
| | Output: | [(None, 32)] |
| dense_9 | Input: | [(None, 32)] |
| | Output: | [(None, 64)] |
| dense_10 | Input: | [(None, 64)] |
| | Output: | [(None, 128)] |
| decoder_output: | Input: | [(None, 128)] |
| | Output: | [(None, 41)] |

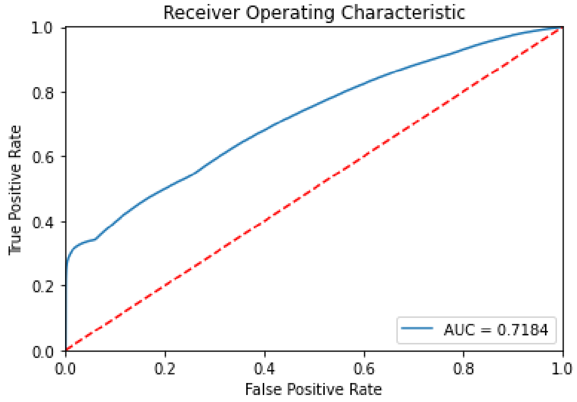
val_loss (mse): 3.6713e-04
val_accuracy: 0.7190

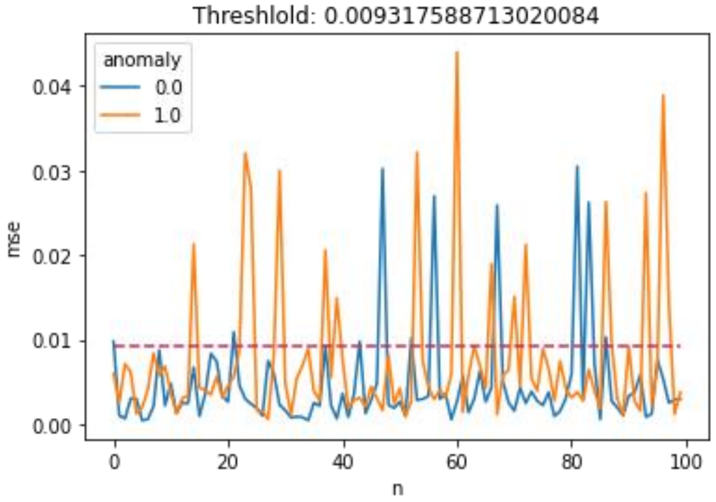
# 2 Autoencoder

## Evaluation

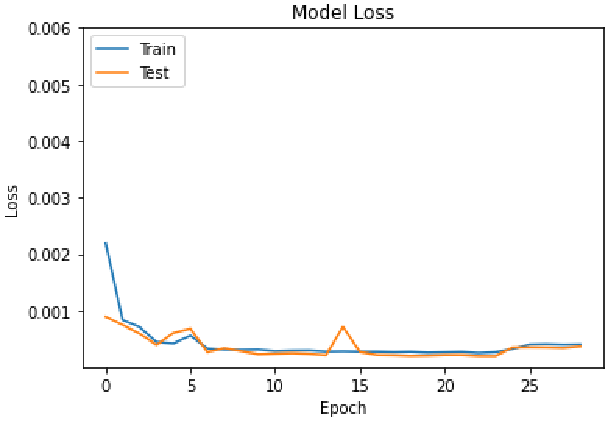
### Basic Model

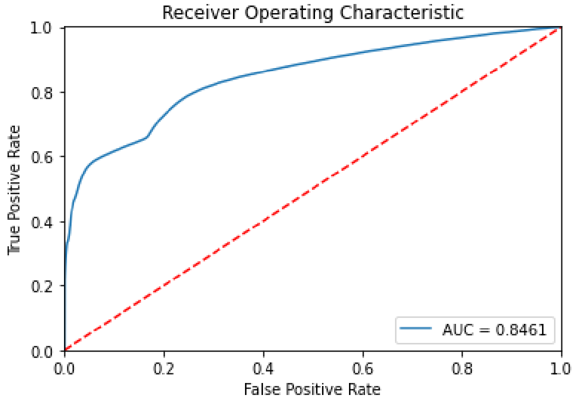


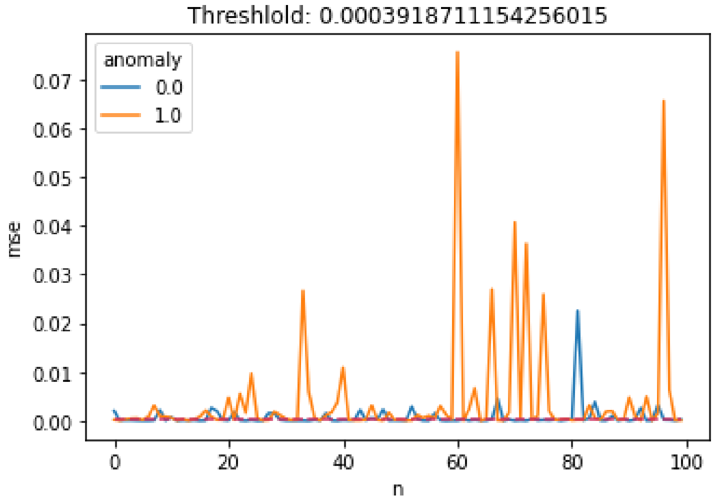




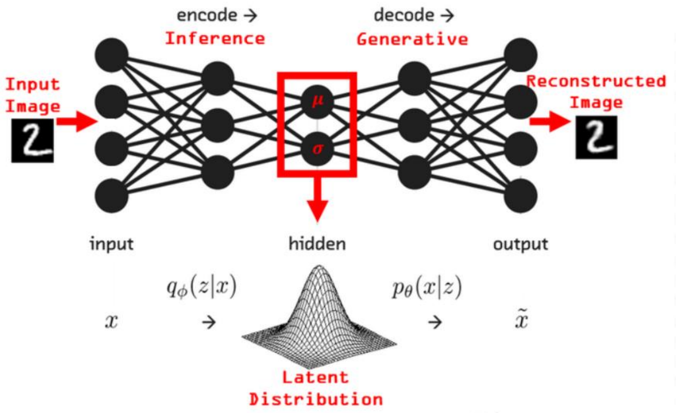
### Improved Model







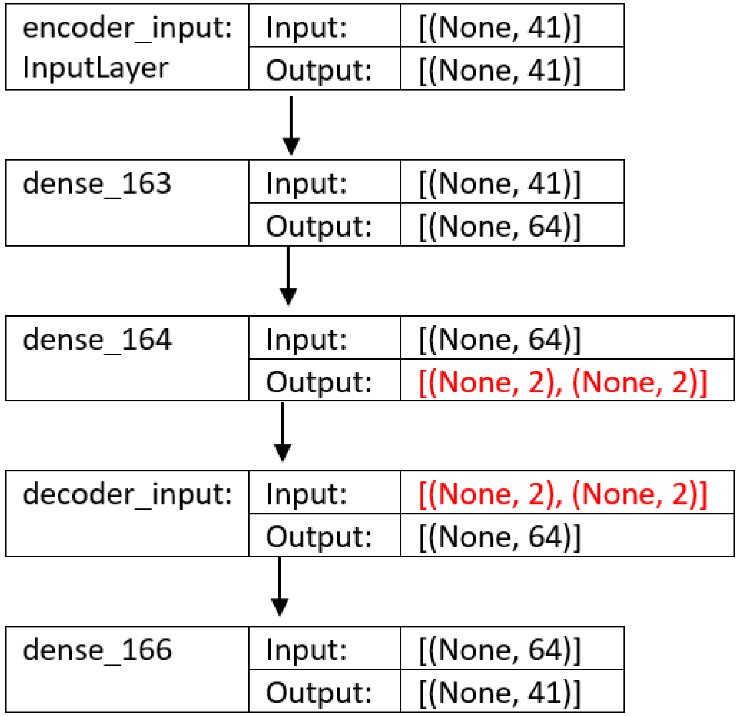
# 3 Variational Autoencoder



## Modeling

Parameter: loss=reconstruction_loss + kl_loss, optimizer='adam'

Basic Model



| | | |
|---|---|---|
| encoder_input: InputLayer | Input: | [(None, 41)] |
| | Output: | [(None, 41)] |

| | | |
|---|---|---|
| dense_163 | Input: | [(None, 41)] |
| | Output: | [(None, 64)] |

| | | |
|---|---|---|
| dense_164 | Input: | [(None, 64)] |
| | Output: | [(None, 2), (None, 2)] |

| | | |
|---|---|---|
| decoder_input: | Input: | [(None, 2), (None, 2)] |
| | Output: | [(None, 64)] |

| | | |
|---|---|---|
| dense_166 | Input: | [(None, 64)] |
| | Output: | [(None, 41)] |

val_loss : 10.6706
val_accuracy: 0.3098

- Change architectures
  - ○ Hidden layers

More complex architecture:
Adding hidden layer functions little

- Parameters
  - ○ Batch size
  - ○ optimizer

Optimizer: (1) RMSprop doesn't work
(2) Adamax is sort of better than Adam

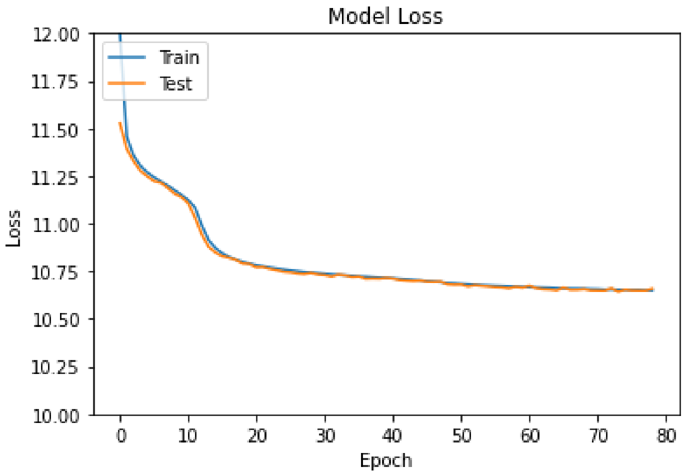
- Early Stopping

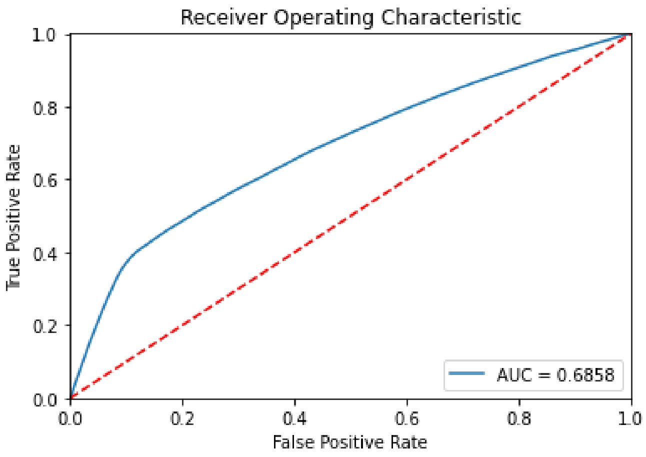
**Models improve little**

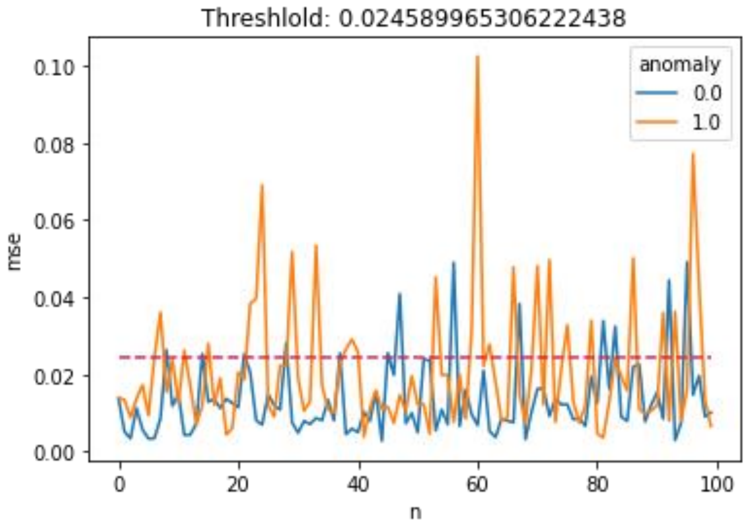
# 3 Variational Autoencoder

## Evaluation

Basic Model







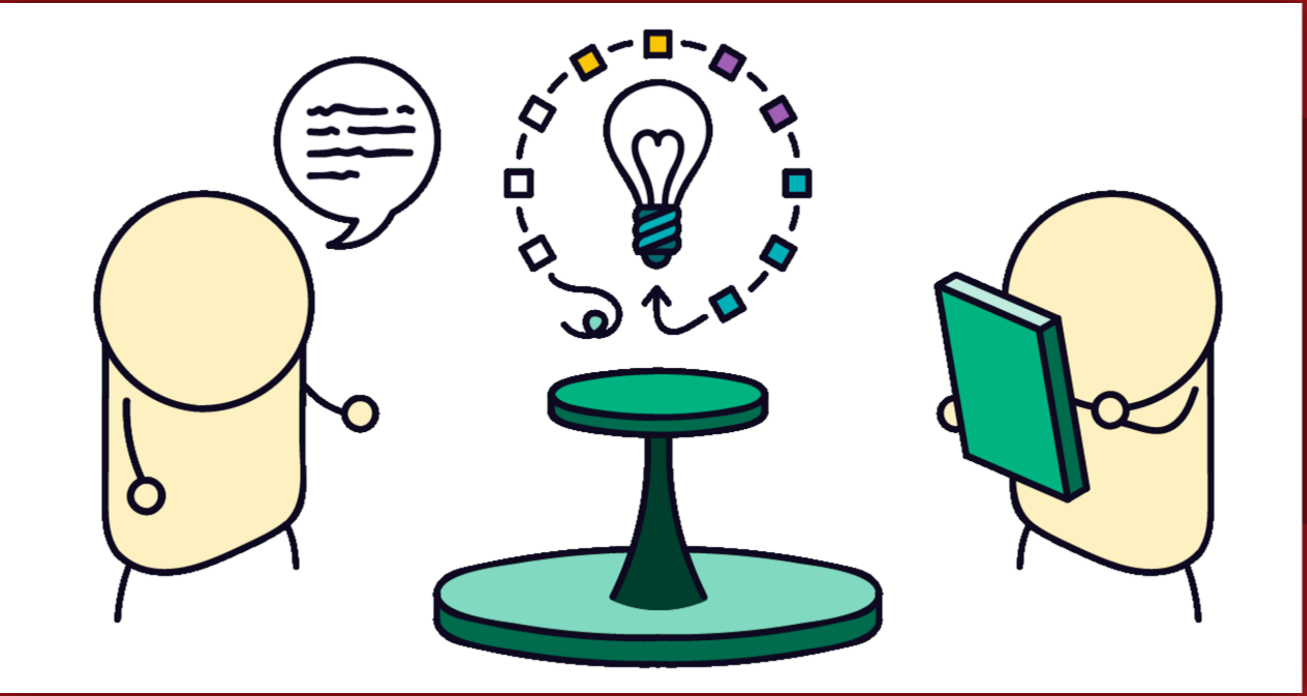
# 4 Conclusion

- Autoencoder can perform well in this anomaly detection problem after improving model using  change architectures, parameters, and early stopping
- Potential Method:

(1) Use pretraining model to train the autoencoder

(2) Use other optimizers to train model (simulated annealing, genetic algorithm, particle swarm optimization, etc.)

- VAE doesn’t work well so far
- Potential Solutions: Use other optimizers to train model

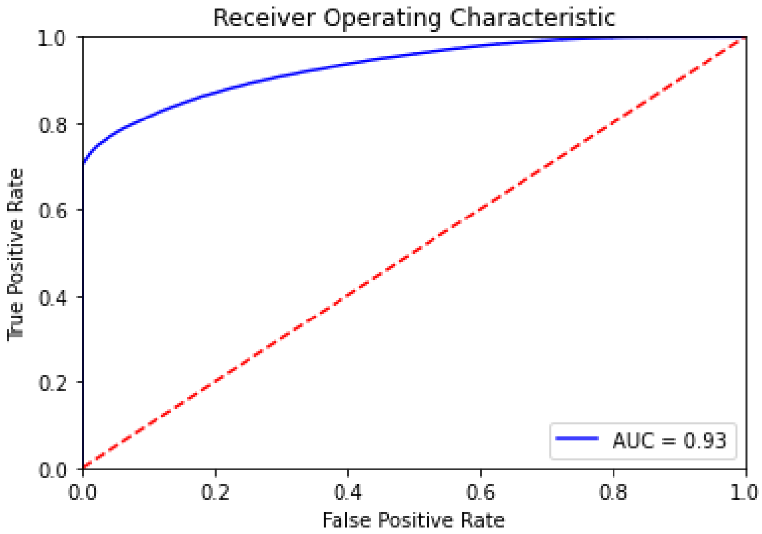


# 4 Model Performance Comparison

# Model Performance Comparison

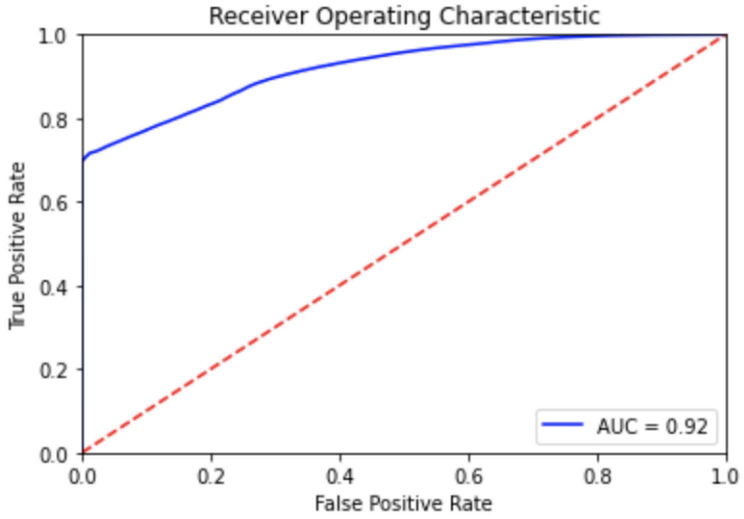
(1) Fully-connected NN-Undersampling

Accuracy: 0.8642



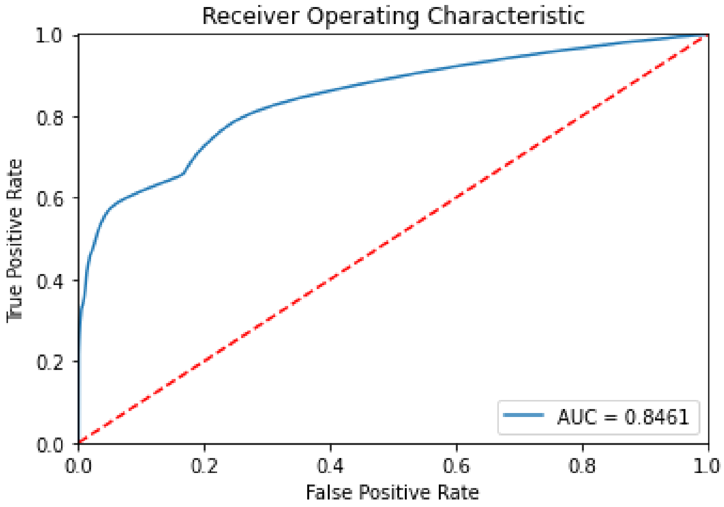
(2) CNN-Oversampling

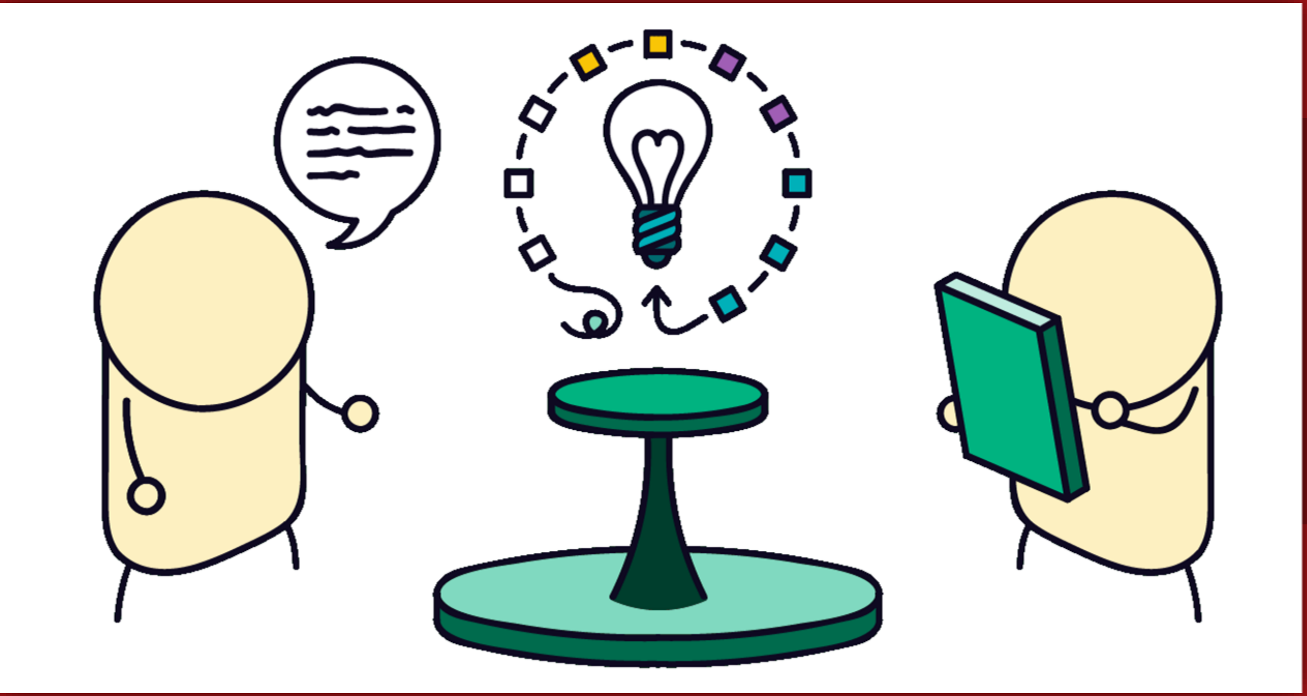
Accuracy: 0.8485



(3) Autoencoder

Accuracy: 0.6988





# 5 Conclusions

# Conclusion

**Fully-connected NN:**

- **Undersampling** would ease imbalance problem, and functions better than the oversampling tools for fully-connected NN and this dataset

**CNN:**

- **Oversampled** data performs better with the CNN model in both accuracy and AUC/ROC curve

**Autoencoder/VAE:**

- Based on current algorithm, autoencoder performs better than VAE

For current deep learning algorithms and dataset, **undersampled fully-connected NN** works best for this loan default detection problem.

# Sources

[1] Brownlee, Jason. “SMOTE for Imbalanced Classification with Python.” Machine Learning Mastery, 20 Aug. 2020, machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

[2] Berk Gokden, Applying Anomaly Detection with Autoencoders to Fraud Detection, towards data science 2020: https://towardsdatascience.com/applying-anomaly-detection-with-autoencoders-to-fraud-detection-feaaee6b5b09

[3] Deep Dense Convolutional Networks for Repayment Prediction in Peer-to-Peer Lending, http://sclab.yonsei.ac.kr/publications/Papers/IC/2018_SOCO_JYK.pdf

[4] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[5] LendingClub Issued Loans Data: https://www.kaggle.com/husainsb/lendingclub-issued-loans

[6] Renström, Holmsten: Fraud Detection on Unlabeled Data with Unsupervised Machine Learning, KTH 2018

[7] The Keras Blog-Building Autoencoders in Keras: https://blog.keras.io/building-autoencoders-in-keras.html

[8] Tom Sweers, Autoencoding Credit Card Fraud, Radbound University 2018

[9] Understanding of Convolutional Neural Network (CNN) — Deep Learning

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. ACM Comput. Surv., 41(3):15:1–15:58, July 2009

[10] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau. Autoencoder-based network anomaly detection. In 2018 Wireless Telecommunications Symposium (WTS), pages 1–5, April 2018.

[11] Packages: Python 3.0 numpy; pandas; matplotlib; seaborn; sklearn; mlxtend.plotting; pathlib2; pickle; keras; tensorflow;