國立政治大學應用數學系

碩士學位論文

深度學習在不平衡數據集之研究

Survey on Deep Learning with Imbalanced Data Sets

研究生：蔡承孝　撰

指導教授：蔡炎龍　博士

中華民國 108 年 8 月

DOI:10.6814/NCCU201901175

致謝

能夠完成這篇論文首先要感謝的就是我的指導教授炎龍老師，從他收我當學生的那一刻起，我就像是更生人般重獲新生。雖然幾經波折，但還是在八月底完成口試，不違背朋友家人以及自己的期許，真的很感謝老師！

接著是我的口委老大以及張媽，感謝你們順利讓我完成口試，無數次的四川、大團圓、敘園、新馬辣是我們共同的回憶，綠色瓶子的鏗鏗鏘鏘是我們共同的作詞作曲，有機會再來一杯吧！

感謝師姊佳琪帶我進入炎龍幫，不只在論文這條路上在前方為我開拓蠻荒，在人生也時時激勵著我推我向前，若沒有你我可能還在自怨自艾躊躇不前；感謝同學 7 年的振維，這輩子能喝到你調的酒是我上輩子戒酒的福分，感謝你讓我更認識我自己，但比起酒精我還是選擇你，你不開個酒吧真的可惜；感謝原同門師兄弟黃賴，你的細心與堅持常常砥礪我自己必須更認真，讓我知道我還有所不足，我真心希望你能畢業！

感謝我的父母，都 25 歲了還沒有開始工作，沒有給我經濟跟時間上的壓力，能讓我完成碩士學位，並讓我誕生在這世界上，這種父母哪裡找？我由衷地感謝並希望未來我也有能力讓你們做自己想做的事！

最後感謝出現在我生命中的所有人，是你們造就了現在的我。我常常把進入政大這件事視為我人生中一大錯誤這句話掛在嘴邊，但是在我打這致謝熱淚盈眶地當下，我不再後悔進入政大，因為能遇見各位的我是最幸福的人，謝謝！

DOI:10.6814/NCCU201901175

中文摘要

本文旨在回顧利用深度學習處理不平衡數據集和異常偵測的方法，我們從 MNIST 生成兩個高度不平衡數據集，不平衡比率高達 2500 並應用在多元分類任務跟二元分類任務上，在二元分類任務中第 0 類為少數類；而在多元分類任務中少數類為第 0、1、4、6、7 類，我們利用卷積神機網路來訓練我們的模型。在異常偵測方面，我們用預先訓練好的手寫辨識 CNN 模型來判斷其他 18 張貓狗的圖片是否為手寫辨識圖片。

由於數據的高度不平衡，原始分類模型的表現不盡理想。因此，在不同的分類任務上，我們分別利用 6 個和 7 個不同的方法來調整我們的模型。我們發現新的損失函數 Focal loss 在多元分類任務表現最好，而在二元分類任務中隨機過採樣的表現最佳，但是成本敏感學習的方法並不適用於我們所生成的不平衡數據集。我們利用信心估計讓分類器成功判斷所有貓狗圖片皆不是手寫辨識圖片。

關鍵字：深度學習、卷積神經網路、不平衡數據集、異常偵測、圖像分類

DOI:10.6814/NCCU201901175

Abstract

This paper is a survey on deep learning with imbalanced data sets and anomaly detection. We create two imbalanced data sets from MNIST for multi-classification task with minority classes $0, 1, 4, 6, 7$ and binary classification task with minority class 0. Our data sets are highly imbalanced with imbalanced rate $\rho = 2500$ and we use convolutional neural network(CNN) for training. In anomaly detection, we use the pretrained CNN handwriting classifier to decide the 18 cat and dog pictures are handwriting pictures or not.

Due to the data set is imbalanced, the baseline model have poor performance on minority classes. Hence, we use 6 and 7 different methods to adjust our model. We find that the focal loss function and random over-sampling(ROS) have best performance on multi-classification task and binary classification task on our imbalanced data sets but the cost sensitive learning method is not suitable for our imbalanced data sets. By confidence estimation, our classifier successfully judge all the pictures of cat and dog are not handwriting picture.

Keywords: Deep Learning, CNN, Imbalanced Data Sets, Anomaly Detection, Image classification

DOI:10.6814/NCCU201901175

Contents

DOI:10.6814/NCCU201901175

DOI:10.6814/NCCU201901175

DOI:10.6814/NCCU201901175

List of Tables

DOI:10.6814/NCCU201901175

DOI:10.6814/NCCU201901175

List of Figures

DOI:10.6814/NCCU201901175

DOI:10.6814/NCCU201901175

Chapter 1

Introduction

Deep learning is kind of machine learning and its development is mature and completed than before. [24] We can see that many fields are using deep learning to address their problem such as translation, finance, network and image recognition. [1, 14, 15, 29] Take an example, using deep learning to translate Chinese in English and input an car image then deep learning model will give the output which is label of car image. This paper using deep learning to recognize the handwriting number which is 0 to 9 but the data set is imbalanced and it will cause the bad predict result of the model. In anomaly detection, we use deep learning to pretrain a handwriting classifier and hope it can judge the input is handwriting picture or not.

Convolutional Neural Network(CNN) is one of structure of deep learning and it is good at image recognition [20],video analysis [19] and natural language processing [8]. There are two main layers in CNN which are convolutional layers and pooling layers, convolutional layers can capture the features of pictures such as shape or line and these features will through the pooling layer to reduce the dimension. Usually, they will flat the result as above and connect to the fully connected layer to get the output which is the probability of the labels.

In real life, there are many different situations and some of them are bad or rare happened, we may called it abnormal condition. For example, the machine malfunction [31], system failure in IT services [39]. Abnormal condition as above does not happen often but when it happen it may cost a lot of time and money to fix it. Usually, identify abnormal condition is a binary classification problem in deep learning which we use to classify this condition is abnormal or normal.

Imbalanced data set is a common situation in real life. For example, we want to predict

DOI:10.6814/NCCU201901175

someone will get the cancer in 3 months by their medical record. Usually, the number of health people is much larger than cancer patient and we assume their ratio is $9:1$. Hence, our model just predict all people are health then the accuracy still get 90%. If we just use accuracy as our criterion, the model seems like good enough but actually, it does not reach our goal to predict someone will get the cancer or not in 3 months. This situation is common in our life and we usually called the uncommon event or bad event as abnormal condition such as machine malfunction [31] and system failure in IT services [39]. In this paper, we create two imbalanced subset from MNIST which has 10 classes of handwriting number images from 0 to 9. In multi-class task, $0, 1, 4, 6, 7$ is our minority class and its imbalance ratio is $\rho = 2500$. On the other hand, binary class task also has imbalance ratio $\rho = 2500$, and the minority class and majority class is 0 and 1, respectively.

In anomaly detection, there are two type of training data, with labels and without labels. If we already have a classifier to classify normal samples, then we want to let the classifier judge the input is normal or anomaly. We can use confidence estimation to let classifier output the confidence score and determine the input is anomaly or not. On the other hand, if we do not have the labels of inputs, we may assume that the data obey the gaussian distribution and believe that the outlier is the anomaly samples. We use confidence estimation on handwriting CNN classifier and successfully judge the 18 cat and dog pictures are not handwriting pictures.

To address the imbalanced data problem, we sort out the 7 different methods which divided into two categories, data-level methods and algorithm-level methods. In data-level methods we will introduce ROS [12, 30], SMOTE [5], RUS [10, 12, 25] which used sampling to increase or decrease the number of samples to balance the data set. Different from data-level methods, we will introduce three loss function, MFE [36], MFSE [36] and focal loss [26], and cost sensitive learning [11, 13, 16, 22, 27, 40]. There methods adjust our output or loss function to let the model be sensitive to minority class samples in algorithm-level methods. In this paper, we use these methods to improve the performance of two baseline model on imbalanced MNIST. We will compare the modified model with baseline model in different task and choose which one is better solution to address the problem.

DOI:10.6814/NCCU201901175

Chapter 2

Deep Learning

Deep learning is just like a machine learning, it can let the computer learning by itself. According to the training data, deep learning is a way to find the most suitable function about input data. Let's take some example to figure out how the function work.

$$f(\text{“Image of tree”}) = \text{“Tree”}$$
$$f(\text{“我很帥”}) = \text{“I am handsome.”}$$
$$f(\text{“How are you ?”}) = \text{“I'm fine, thank you.”}$$

When using deep learning in image recognition [14], we hope that computer can find a function which can distinguish the images. If we send a tree image to the function as an input, then computer can recognize it and give the label of image "tree" as an output which shown as above. In first function above, the input are images and the output is the label of these images. When we use deep learning in translation [1], we expect that machine can translate the words into the language which we want. As the second part of cases, " 我很帥", the description that " i am handsome " in Taiwan, is the input of the function. Then the function shows that " I am handsome." as an output. In this example, the Taiwan characters are input, and the English sentence translated from Taiwan characters are the output. As the deep learning is applied in prediction about conversation [38], we hope that computer can tell us what the next step is. We set the sentence " How are you ? " as an input. Then " I'm fine, thank you." is the output of the function which is the answer we should respond. Therefore, in the function shown in third case above, the input of the function is the current sentence and the responding sentence you should reply is the output. After figuring out these three cases above, we understand deep learning

DOI:10.6814/NCCU201901175

much more than before and it is time to see how it works.

Actually, deep learning can be divided into three steps which are building the model, selecting loss function and training the model as shown in figure 2.1. Building the model means that construct a neural networks and set a function according to the structure. In this step, we need to decide the structure of neural networks and design the program about that. To determine which function is the best, we need to select a loss function to evaluate the goodness of those functions. So we need to define what is the goodness of the function in last step. Hence, according to the training data, computer can choose the best function by training the model.

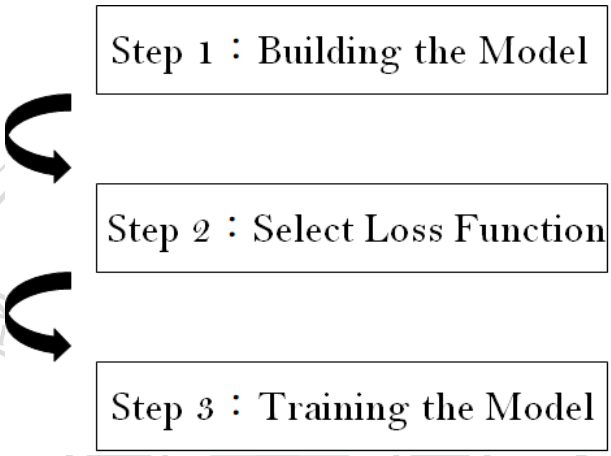


Figure 2.1: Three steps of deep learning

2.1 Neurons and Neural Networks

Because of artificial neural networks are inspired by biological neural networks, the structure of artificial neural networks is similar to the real human brains. [17] Neurons are connected with neural networks in human brains, so artificial neural networks in deep learning also have neural networks around the neurons. In this section, we will show that the structure of neurons and the operation between the neurons in neural networks.

As in figure 2.2, a basic neuron is consisted of input, output, weights, bias and activation function. $x_1,\ x_2,\ x_3$ in the left side of figure 2.2 are the input of neuron. In the right side, y_1 is the output of neuron. Circle in figure 2.2 is a neuron and its symbol σ is the activation function of neuron which will introduce later. Between the input and neuron are weights, denoted as $w_1,\ w_2,\ w_3$. Finally, the bottom of figure 2.2 is the bias of the neuron, b_1.

DOI:10.6814/NCCU201901175

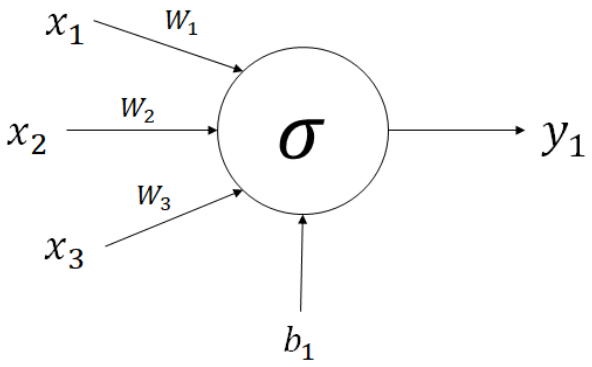


Figure 2.2: The Structure of a Neuron

Now, we introduce the operation of neurons. First, input value is multiplied by the corresponding value of weights, respectively. Then add the product values and bias together. Finally, let the value we get through the activation function, then we will get the output. Take a simple instance, we set $x_1,\ x_2,\ x_3,\ w_1,\ w_2,\ w_3,\ b_1$ as $4,\ 2,\ (-3),\ (-1),\ 3,\ 2,\ 3$, respectively and activation function is ReLU which will introduce in detail later. According to the operation we introduced, we have $4 \times (-1) + 2 \times 3 + (-3) \times 2 + 3 = -1$. Since ReLU will send negative value to 0, then the out put is

$$\sigma(4 \times (-1) + 2 \times 3 + (-3) \times 2 + 3) = \sigma(-1) = 0.$$

In deep learning, weights and bias are called parameters. Parameters are adjustable, when training the model by training data, computer will adjust the parameters automatically to fit the training data.

After discussing the structure of a neuron and its operation, we will introduce the neural network. There are a lot of neurons in neural network, and they can connect to other neuron freely. The figure 2.3 shows that the typical model of neural network in deep learning.

DOI:10.6814/NCCU201901175

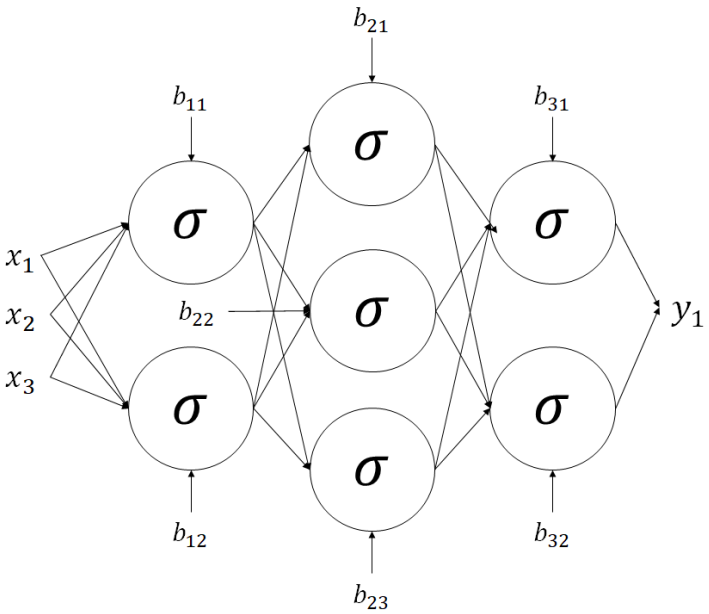


Figure 2.3: Fully Connected Feedforward Network

Fully connected feedforward network is the simplest and first model in deep learning as shown in figure 2.3. [35] In figure 2.3, there are 3 layers, first and third layers contain 2 neurons, second layer contains three neurons. The number of layers and neurons are arbitrary which is decided by ourselves. The input layer and out layer are the the left and right layers in figure 2.3, respectively. Between input layer and output layer are hidden layers, the number of hidden layers is 3 or more than 3 will be called in deep structure. The neurons in fully connected feedforward network are connected to all neurons of adjacent layers which is showed in figure 2.3 and explained what is ” fully connected ”. But in this model, it does not have any circle or loop. We will introduce other model which contains circle or loop later.

If we decide a structure of neuron networks, then this structure will define a set of functions. Machine will find the most suitable parameters according to the training data, this procedure we call it training and it is equivalent to choose a solution of set of functions. But there may be no or bad solution of set of functions because the set of functions defined by the model of structure is not suitable to the problem. For different problem, we have different model to fit it. It is difficult to let machine decided its model by itself, so this task is still done by ourselves.

DOI:10.6814/NCCU201901175

2.2 Activation Function

The following activation functions we introduced are non-linear functions. Because without activation functions the output of neurons in every layers are linear combination of the input, so activation function give them non-linear relationship. Let's see three most common activation functions in detail.

1. Rectified linear unit (ReLU)

   Equation:

$$f(x) = \begin{cases} x, & \text{if} \quad x \geq 0 \\ 0, & \text{if} \quad x < 0 \end{cases}$$

   Range: $[0, \infty)$

   Graph:

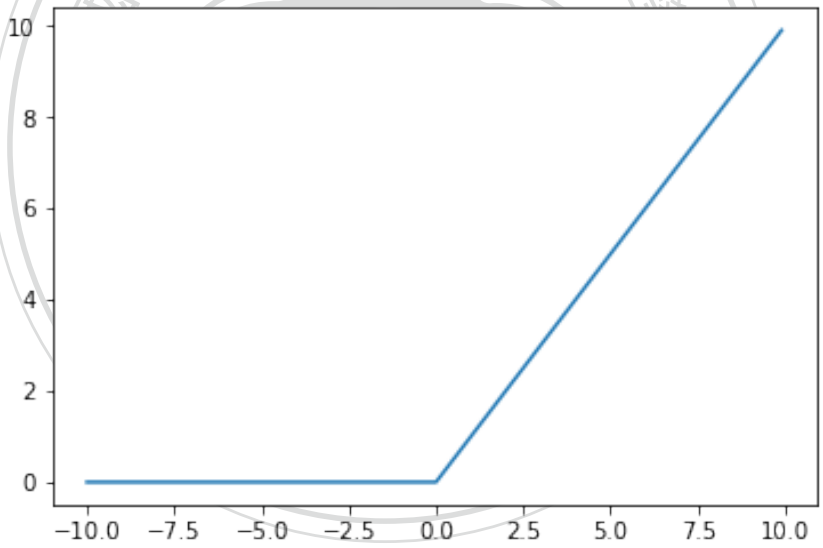


Figure 2.4: Rectified linear unit (ReLU)

2. Sigmoid function

   Equation:

$$f(x) = \frac{1}{1 + e^{-x}}$$

   Range: $(0, 1)$

   Graph:

DOI:10.6814/NCCU201901175

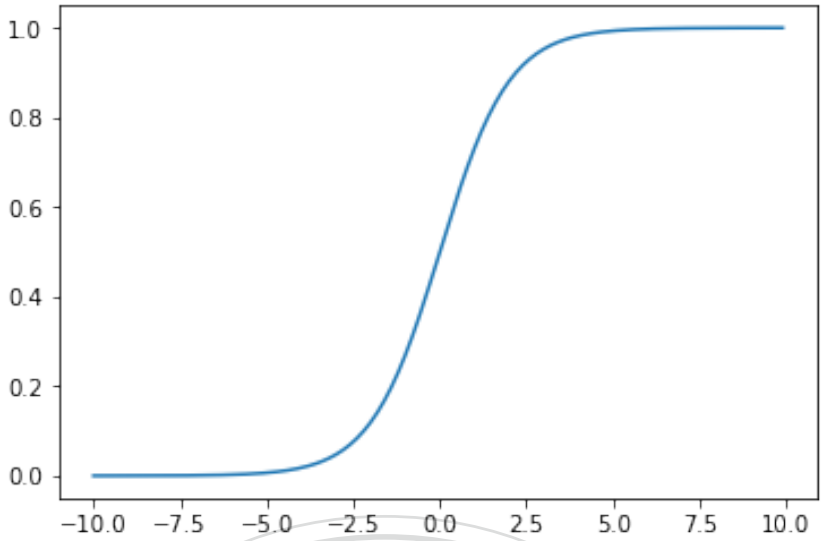


Figure 2.5: Sigmoid function

3. Hyperbolic tangent (tanh)

   Equation:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   Range: $(-1, 1)$

   Graph:

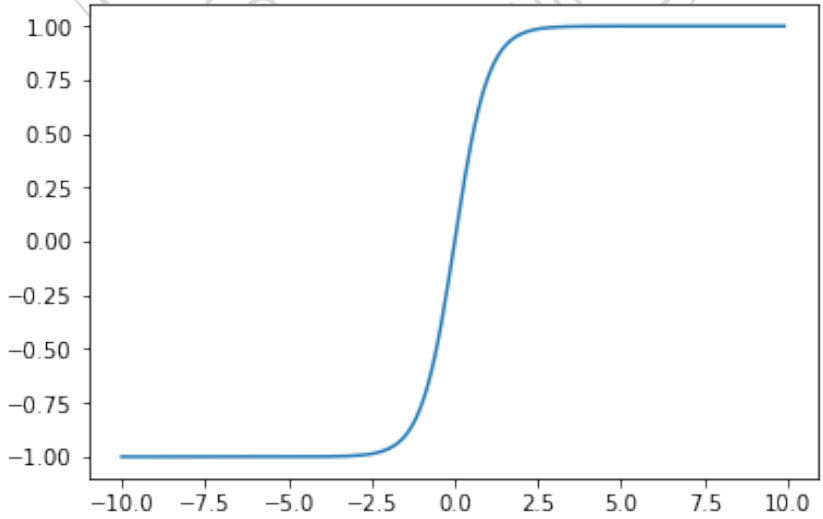


Figure 2.6: Hyperbolic tangent (tanh)

DOI:10.6814/NCCU201901175

2.3 Loss Function

After we decided the connection between the neurons, we need to adjust parameters i.e. weights and bias. The set which contains this parameters are called θ and every θ defined a function, the set collect all this function is denoted by $\{F_\theta\}$. We want to find the best function of problem i.e. find the most suitable parameter, the optimal function is denoted by F_{θ^*}. Define the loss function is the way to find the optimal function.

Loss function map from parameter space to real number. For example, the number of training data is k namely, $(x_1, y_1), (x_2, y_2), ..., (x_n, y_k)$ and θ is the set of neural network parameters i.e. $\theta = \{w_1, w_2, ..., w_n, b_1, b_2, ..., b_m\}$ then the loss function is defined by $L : R^{n+m} \mapsto R$. We use loss function to evaluate the difference between the real value y and predicted value $F_\theta(x)$. Of course, we hope that the predicted value is close to the real value, so we need to minimize the loss function.

Choose a correct loss function is an important thing, like classification problem, it usually uses binary cross-entropy as its loss function. The following are the three basic and simple loss functions.

1. Mean absolute error (MAE)

$$L(\theta) = \frac{1}{k}\sum_{i=1}^{k} ||y_i - F_\theta(x_i)||$$

2. Mean squared error (MSE)

$$L(\theta) = \frac{1}{k}\sum_{i=1}^{k} ||y_i - F_\theta(x_i)||^2$$

3. Binary cross-entropy

$$L(\theta) = -\frac{1}{k}\sum_{i=1}^{k} [y_i \log(F_\theta(x_i)) + (1 - y_i) \log(1 - F_\theta(x_i))]$$

DOI:10.6814/NCCU201901175

2.4 Gradient Descent Method

In deep learning, the optimization is different from other common case. Generally, when optimizing, we know how data exactly looks like and what goal we want to reach. But in deep learning, we do not know how new cases look like, so we optimize our training data and use validation data to test its performance.

The way we use to optimize training data is "gradient descent" which is usually used in deep learning. Assume that the parameters of our neural networks are $\theta = \{w_1, w_2, ..., w_n, b_1, b_2, ..., b_m\}$, we want to find the optimal solution θ^* such that minimizes the loss function $L(\theta)$. The following are the step of gradient descent method. First, we choose a random value as an initial value for w_1 denoted as $w_1^{(1)}$. Second, compute its first derivation $\frac{\partial L}{\partial w_1^{(1)}}$ then update the parameter i.e.

$$w_1^{(2)} = w_1^{(1)} - \eta \frac{\partial L}{\partial w_1^{(1)}}$$

which η is learning rate and will introduce later. Continuing this way, until $\frac{\partial L}{\partial w_1}$ is approach to zero. As same as the other parameters, so the iteration is like the following form in figure 2.7 and we will do this iteration until all the parameter of norm is small enough.

$$\begin{bmatrix} w_1^{(t+1)} \\ w_2^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \\ b_1^{(t+1)} \\ b_2^{(t+1)} \\ \vdots \\ b_m^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(t)} \\ w_2^{(t)} \\ \vdots \\ w_n^{(t)} \\ b_1^{(t)} \\ b_2^{(t)} \\ \vdots \\ b_m^{(t)} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial L}{\partial w_1^{(t)}} \\ \frac{\partial L}{\partial w_2^{(t)}} \\ \vdots \\ \frac{\partial L}{\partial b_m^{(t)}} \end{bmatrix}$$

Figure 2.7: Gradient Descent

Learning rate is like a extent which represents a level we approach to minimum value in every iteration and is set by ourselves. Usually, learning rate is small, because if learning rate is large, then it may miss the minimum value or jump back and forth around the minimum value. However, when learning rate is too small, the speed of training will too slow and cost too many time. Therefore, setting learning rate is an important thing in deep learning.

DOI:10.6814/NCCU201901175

Chapter 3

Convolutional Neural Network(CNN)

Neural network (NN), Convolutional neural network (CNN) and Recurrent neural network (RNN) are three based model in deep learning. [24] NN is the standard model of deep learning and other models are modified edition according to it. CNN is named by its convolutional layer and is good at image recognition and classification problems. On the other hand, since the output of the previous layer is the input of the next layer, the whole model has recursiveness, so RNN is good at processing time-related or sequential data.

The standard structure of CNN consisted of two main layers, convolutional layers and pooling layers. The input image will be captured feature in convolutional layers and be reduced the dimension in pooling layers. After that, the result will be flattened and through the fully connected layers which use softmax as activation function to get the probability from of output. Figure 3.1 shows the detail of structure of CNN and the following section we will introduce two main layers in CNN.

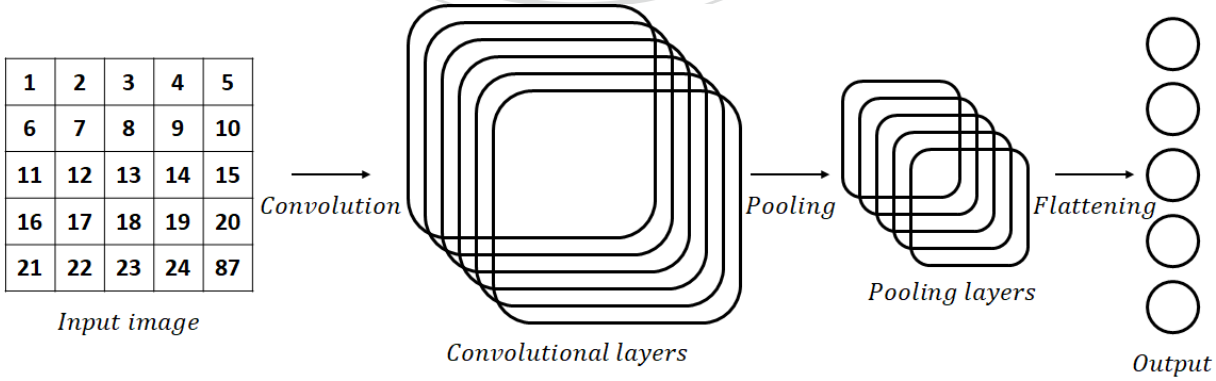


Figure 3.1: Structure of CNN

DOI:10.6814/NCCU201901175

3.1 Convolutional Layer

Convolutional layer uses the filters to capture the features of input images such as the shape or line of the pictures and one filter corresponds to one features. The following is the convolutional operation which is notation is $\otimes$ as shown in figure 3.2. It multiplies each elements of filter and each elements of 3×3 matrix in the upper left of the input image. Then add them together and we will get the element of feature map in the upper left. For example, we get 6 in upper left of feature map by

$$1 \times 1 + 1 \times 2 + 1 \times 3 + 0 \times 6 + 0 \times 7 + 0 \times 8 = 6$$

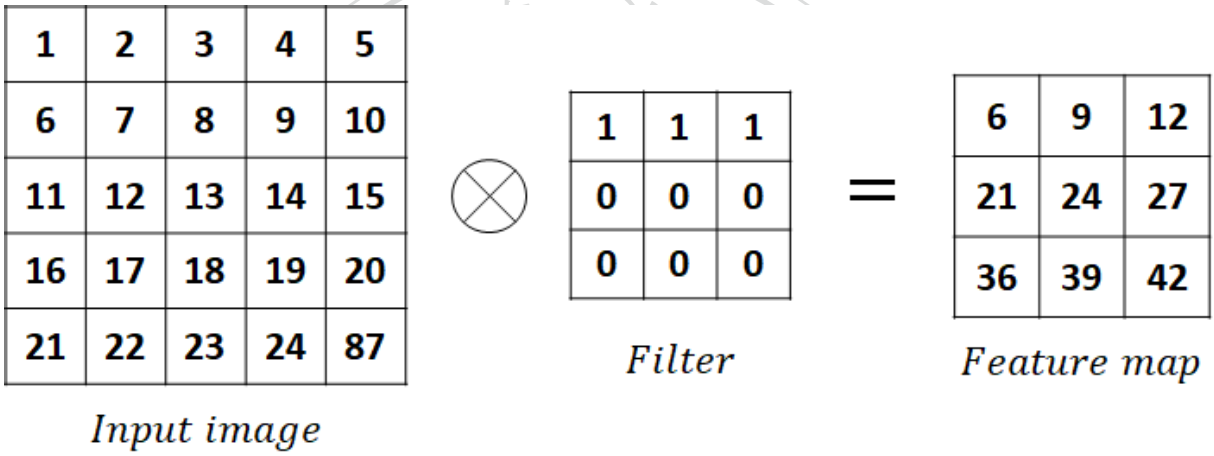


Figure 3.2: Convolutional operation

After we get the output 6, the filter will move one grid of input image and do convolutional operation again then we will get the output 9. Continuing this way, until all 3×3 matrix in the input image convoluted by filter so that we can get the complete feature map shown in figure 3.2.

3.2 Max Pooling Layer

After we get the feature map from convolutonal layer in section 3.1, we use pooling layer to let our feature more streamlined. There are three advantages of max pooling, reduce the dimension of input image, image-denoising and translating few pixels in image will not change the result.

DOI:10.6814/NCCU201901175

Figure 3.3 shows the operation of max pooling layers. Usually, we choose the pooling size as 2×2 and select the maximum value in 2×2 matrix. Hence, we will get the upper left element in pooled feature map 24. Next, move one grid of feature map and do the operation again until we get the complete pooled feature map.

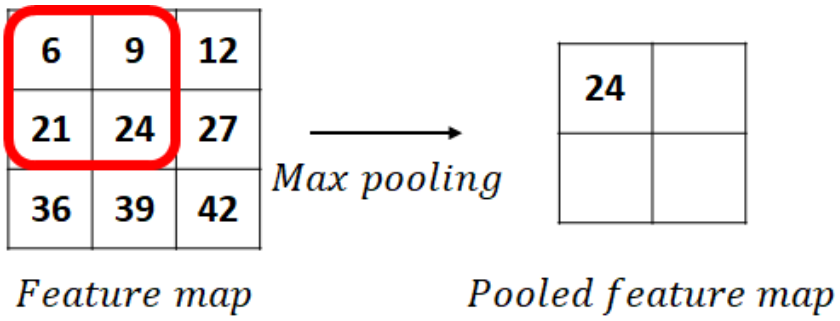


Figure 3.3: Operation of max pooling layer (a)

Different from convolutional operation, the next matrix in feature map can not overlap the previous matrix and it can out of bonds as shown in figure 3.4

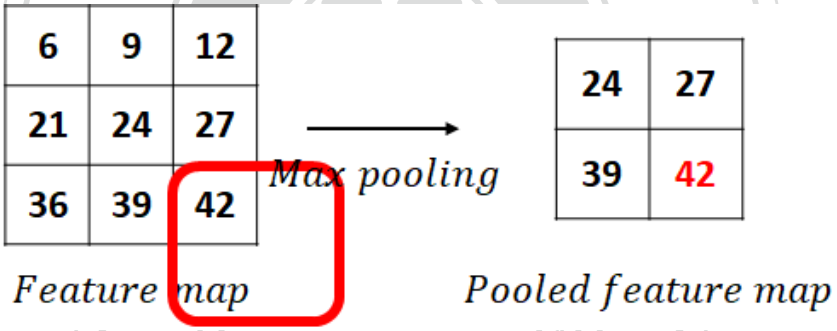


Figure 3.4: Operation of max pooling layer (b)

Finally, we will flatten the pooled feature map which obtain from max pooling layer and send it to fully connected layers to get the output of whole model.

DOI:10.6814/NCCU201901175

Chapter 4

Abnormal Condition and Imbalanced Data Set

In this paper, we want to use deep learning to recognize the handwriting number images. But the data set is imbalanced so not all predictions are we interested, we care more about the accuracy of minority class no matter in multi-class task or binary class task. Hence, this chapter will introduce the abnormal condition and imbalanced data set and describe the problem they caused in deep learning.

4.1 Abnormal Condition

In real life, we will encounter many situations, they may be good or bad or just nothing and we usually call bad situation as abnormal condition. For example, the machine malfunction [31], system failure in IT services [39], unexpected situation in radar data corresponding to six military targets [9] and heart failure in electronic health records (EHRs) [6]. These abnormal condition as above does not happen often but when it happened it may cost a lot of money and time to fix and address it.

As above, we usually want to predict, detect and classify the abnormal condition. Therefore, we use deep learning to help us to find the abnormal condition in condition domain and we regarded it as an binary classification task. In this binary classification task, we call normal condition as positive class and abnormal condition as negative class. Usually, this binary classification task has data imbalanced problem and it will led the performance of the model

DOI:10.6814/NCCU201901175

terrible.

In order to judge the imbalanced level of dataset the following formula [2] represents the imbalance ratio of dataset, that is, the ratio of maximum class size and minimum class size.

$$\rho = \frac{\max_i(|C_i|)}{\min_j(|C_j|)}$$

where C_i is a set of samples in class i. For example, if our dataset's largest class has 48 samples and smallest class has 16 samples, then the imbalance ratio $\rho = 3$. Note that the imbalanced data set not just happen in binary classification task, it also happen in multi-classification task and we will introduce it in next section.

In our work, we create an binary imbalanced data set from MNIST and it consisted two classes, 0 and 1. The sample number of class 0 and 1 are 2 and 5000, respectively and it cause the imbalanced ratio $\rho = 2500$.

4.2 Imbalanced Data Set

Usually, the abnormal condition can regard as a binary classification task which consisted of one positive group and one negative group i.e. normal condition and abnormal condition. But imbalanced data set not just happen in binary classification task, it will happen in multi-classification task too. This section will discuss the imbalanced data in multi-classification task and description the problem in this paper.

Imbalanced data arise in many different application which have rare frequency on positive class such as computer security [7], disease diagnosis [32], image recognition [21]. In this paper, we create an imbalanced multi-class data set from MNIST with minority classes $0, 1, 4, 6, 7$ and imbalanced ratio $\rho = 2500$.

If we do not do something to adjust the model, it may get good accuracy but in fact this model does not achieve the purpose we want at first. For example, we want to predict someone will get the cancer in 3 months or not by their medical record. So our purpose is classify someone is health or get the cancer. Usually, the number of health people is much larger than cancer patient and we assume their ratio is $9 : 1$. Hence, our model just predict all people are health then the accuracy still get 90%. If we just use accuracy as our criterion, the model seems like good enough but actually, it does not reach our goal to predict someone will get the cancer or

DOI:10.6814/NCCU201901175

not in 3 months and if someone has cancer and the model tell him that he is health, it may cause irreparable consequences. To prevent this situation we will introduce 7 different methods to address this problem in chapter 6.

DOI:10.6814/NCCU201901175

Chapter 5

Anomaly Detection

Anomaly detection is just like an abnormal condition, it can find the outliers, novelties, noise, deviations and exceptions from the data set. Anomaly detection is applied on many fields such as bank fraud [37], medical problems [33], structural defect [3] and errors in a text [28].

We can regard anomaly detection as a binary classification task, it will classify the samples to normal and anomaly. Usually, we are interesting the anomaly condition but it often happened rarely or even not happen in data set and cause highly imbalanced in data set. For example, if we want to detect the existence of the cancer cell [23], the health cell is normal and the anomaly condition is that detect to the cancer cell.

Although anomaly detection can be regard as binary classification, but there are some different and difficult about anomaly detection. First, the distribution of anomaly sample is unknown because the number of sort of anomaly condition is too much. For example, in cancer detection, the cancer cell is anomaly. But if we feed a car or a tree to classifier, it will be regard as anomaly condition too. Second, is hard to collect the anomaly data because most sample in data set is normal. Hence, there are two training set, one is clean for all data , that is, all data is normal, the other one contains some anomaly data.

5.1 Confidence Estimation

If we already have a classifier for health cell, we want to use it to do anomaly detection, then beside the output label of health cell, the classifier must output a confidence score [4]. The confidence score is used to determine the sample is health cell or not. Let $\boldsymbol{x}$ be the input,

DOI:10.6814/NCCU201901175

function f is classifier and c is the confidence score. Given a threshold λ we have

$$f(\boldsymbol{x}) \begin{cases} \text{is normal, if } c(\boldsymbol{x}) > \lambda \\ \text{is anomaly, if } c(\boldsymbol{x}) \leq \lambda \end{cases}$$

For example, given $\lambda = 0.5$, in figure 5.1, the health cell classifier has highly confident for the blood cell input image is blood cell and $0.98 > 0.5$ so the input is normal.

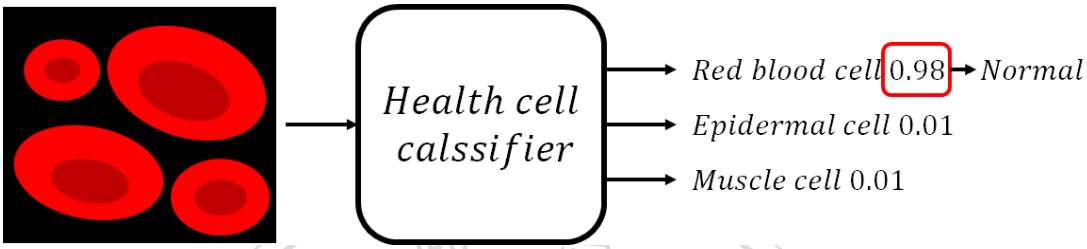


Figure 5.1: Normal condition for heath cell classifier

But as in figure 5.2, when using cancer cell image as input, we have low confidence and $0.34 < 0.5$ so the input is anomaly.

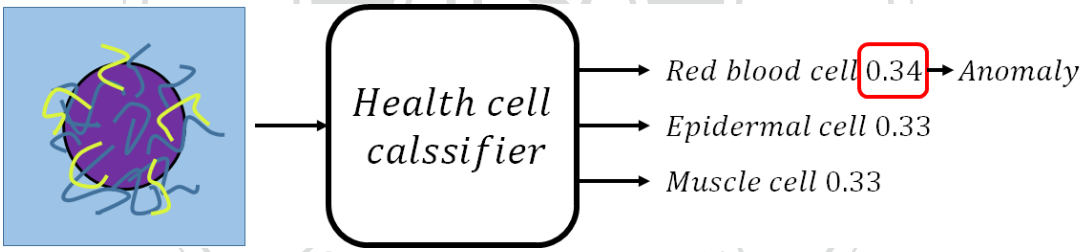


Figure 5.2: Anomaly condition for heath cell classifier

Hence, we can use the maximum of probability which classifier predicted to be the confidence score and determine whether it is greater than λ or not so that we can judge the condition of input.

5.2 Gaussian Distribution

In this section, we will introduce the method that find the outlier when the training data without the label. Since the training data without the label, it is hard to classify which sample is anomaly. We believe that the anomaly condition is rare to happen, so we assume that the data

DOI:10.6814/NCCU201901175

set obey the gaussian distribution and believe that the anomaly sample is the outlier of gaussian distribution [34].

For example, we want to know that the car engine form the production line is a defective or not and we have some information which is the temperature and rotating speed of engine. We use the information to draw a scatter diagram, each points $\boldsymbol{x}$ represent an engine as shown in figure 5.3

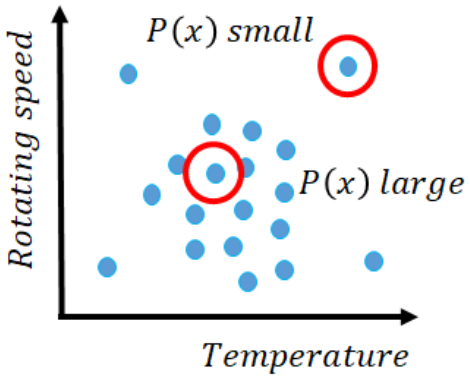


Figure 5.3: Car engine scatter diagram

In figure 5.3, we can find that the point in the middle has higher probability to happen than the point at upper right corner. Hence, given a threshold ε, we can define the normal and anomaly condition as following:

$$\begin{cases} \boldsymbol{x} \text{ is normal, if } P(\boldsymbol{x}) > \varepsilon \\ \boldsymbol{x} \text{ is anomaly, if } P(\boldsymbol{x}) \leq \varepsilon \end{cases}$$

Furthermore, we assume that the data set obey the gaussian distribution and we can use it to calculate the $P(\boldsymbol{x})$ we want. We need to calculate the mean and covariance for each feature x_i which in our example are temperature and rotating speed. The formula for mean and covariance is as following:

$$u_i = \frac{1}{m}\sum_{j=1}^{m} x_i^j$$

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_i^j - u_i)^2$$

, where m is the number of samples. After obtain the mean and covariance for each feature,

DOI:10.6814/NCCU201901175

given any new data $\boldsymbol{x}$, we can use gaussian distribution to calculate its probability as following:

$$P(\boldsymbol{x}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} exp(-\frac{(x_i - u_i)^2}{2\sigma_i^2})$$

Hence, given a threshold ε, we can find the outlier in the data set which means $P(\boldsymbol{x}) < \varepsilon$. Continue the previous example, we let the red line as a threshold ε and then we get 4 anomaly samples as shown in figure 5.4.

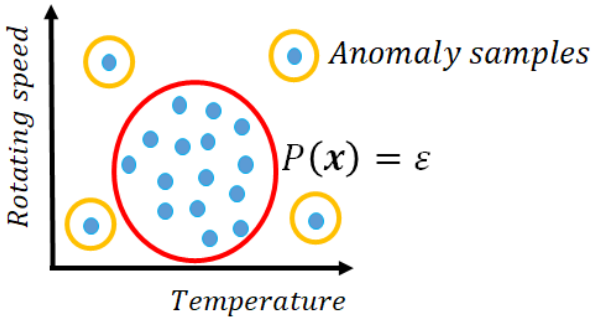


Figure 5.4: Defective detection

5.3 Experiment for Confidence Estimation

In this section, we use a pretrained handwriting classifier to do the confidence estimation. We want to judge that the input is handwriting or not. Hence, we collect 18 pictures of cat and dog as input and hope classifier can predicted them are not handwriting with confidence estimation. Figure 5.5 shows that the example of cat and dog pictures.

(a) Picture of cat (b) Picture of dog

Figure 5.5: Example of cat and dog pictures

DOI:10.6814/NCCU201901175

Our pretrained model use CNN and have vary high accuracy on training set and testing set with 99.98% and 99.35%. After we put the pictures of cat and dog to the handwriting classifier model, we get the confidence score of cat and dog pictures. We set $\lambda = 0.6$ as threshold and table 5.1 shows the confidence score of cat and dog pictures.

	Confidence Score		Confidence Score
Cat 1	0.2014	Dog 1	0.3286
Cat 2	0.1873	Dog 2	0.3873
Cat 3	0.1689	Dog 3	0.4854
Cat 4	0.5715	Dog 4	0.2744
Cat 5	0.5491	Dog 5	0.2345
Cat 6	0.4481	Dog 6	0.178
Cat 7	0.3339	Dog 7	0.4345
Cat 8	0.3436	Dog 8	0.4574
Cat 9	0.3723	Dog 9	0.3637

Table 5.1: Confidence score of cat and dog pictures

From table 5.1, we can see that all of confidence score of cat and dog pictures are not greater than 0.6 so that all of cat and dog pictures are not handwriting picture by our classifier.

After our classifier successfully recognizing the cat and dog pictures are not handwriting pictures, we want to know the classifier can recognize the handwriting pictures or not. Hence, we use $0 - 9$ handwriting number pictures but missing 5 and 6 as input to train our model and get 99.3% and 80.96% accuracy on training set and testing set, respectively. Then we do confidence estimation by using cat and dog pictures and 5 and 6 handwriting number pictures as inputs. Table 5.2 shows the confidence score of inputs.

	Confidence Score		Confidence Score
Cat 1	0.4773	Dog 1	0.9333
Cat 2	0.4771	Dog 2	0.8598
Cat 3	0.3947	Dog 3	0.8056
Cat 4	0.3414	Dog 4	0.9103
Cat 5	0.4708	Dog 5	0.858
Cat 6	0.546	Dog 6	0.7704
Cat 7	0.7042	Dog 7	0.7726
Cat 8	0.7597	Dog 8	0.6686
Cat 9	0.7722	Dog 9	0.5838
Class 5	0.9987	Class 6	0.9993

Table 5.2: Confidence score of inputs

We set $\lambda = 0.8$ and find that our classifier can recognize class 5 and 6 as handwriting

DOI:10.6814/NCCU201901175

number pictures successfully and recognize most cat and dog pictures are not handwriting number pictures.

DOI:10.6814/NCCU201901175

Chapter 6

Methods for Imbalanced Data Problem

In this chapter will introduce many different methods to address the problem of imbalanced data. We are roughly divided into two methods, data-level methods and algorithm-level methods [18] some of them are fit to the binary classification task and the others are suitable to the multi-classification task.

6.1 Data-level Methods

In this section, we will adjust our number of sample from different classes until all classes are balanced. We introduce 3 main methods to handle our original imbalanced data sets. These 3 methods can use on both binary classification task and multi-classification task.

6.1.1 Random-oversampling(ROS)

Random oversampling(ROS) is a popular solution of imbalanced data problem since it is easy and have good performance. [12,30] ROS choose samples randomly from minority classes and copy the samples until the sample number of minority classes is equal to the sample number of majority class. For example, we have A, B, C three classes and the sample number of A, B, C are $10, 4, 5$, respectively. As shown in figure 6.1, A is majority class and minority classes are B and C. After ROS, we can see that the sample number of B and C are equal to A.

DOI:10.6814/NCCU201901175

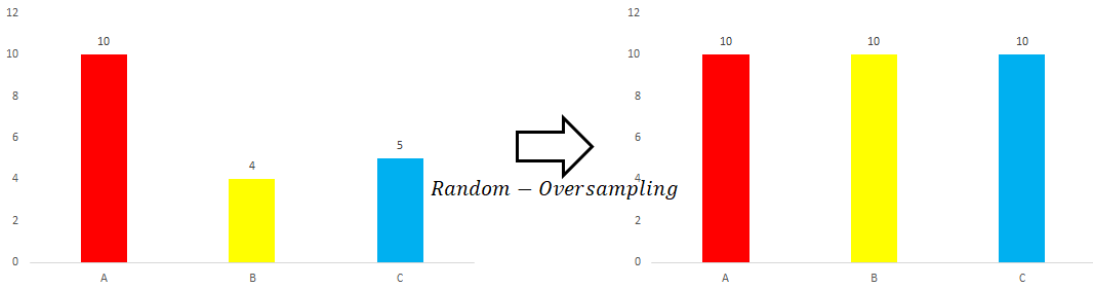


Figure 6.1: Random oversampling(ROS)

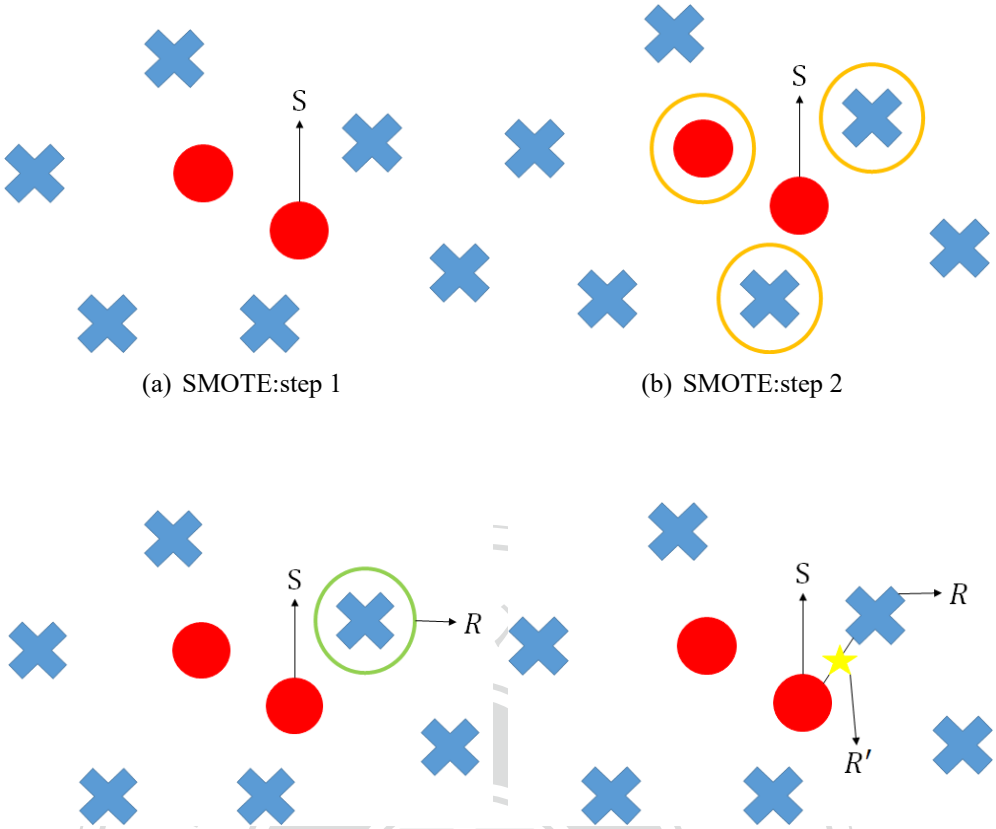
6.1.2 Synthetic minority over-sampling technique(SMOTE)

Although people usually use ROS as a solution of imbalanced data problem, but it may cause overfitting since it has too many repeated samples in a new balanced data set. To avoid this situation, Chawla, Nitesh V., et al. create a advanced method, Synthetic Minority Over-sampling Technique(SMOTE) [5] which produce new minority samples. SMOTE not just copy the sample of minority classes like ROS, it develop an algorithm to create a new sample of minority classes to prevent overfitting. SMOTE choose a sample in minority class randomly, find its k-nearest neighbors and create a new minority sample between them. The following is the algorithm of SMOTE to create new minority samples.

1. Select a minority sample A randomly.
2. Find k-nearest neighbors.
3. Choose 1 sample from k-nearest neighbors randomly and call it B.
4. Create a new sample C which $C = \lambda S + (1 - \lambda)R$, where λ is a random number between 0 and 1.

The k-nearest neighbors which are top k nearest samples to A in all classes, but we usually choose them in minority class. Actually, the new sample C is on the line between A and B. Continue this way, we can get a lot of artificial minority samples until the sample number of all classes are equal. Figure 6.2 shows the algorithm detailed.

DOI:10.6814/NCCU201901175



(a) SMOTE:step 1 (b) SMOTE:step 2

(c) SMOTE:step 3 (d) SMOTE:step 4

Figure 6.2: Algorithm of SMOTE

SMOTE usually have good performance on classification of imbalanced data set, but since several samples are synthetic, the interpretability of the model is greatly reduced.

6.1.3 Random-undersampling(RUS)

Undersampling is another common solution of imbalanced data [12], similar as oversampling, it also let all classes have same sample number. Different from ROS, random undersampling(RUS) remove the data in majority class until all classes have same sample number. Some article use transfer learning with RUS to classify imbalanced data sets of plankton images [25] and some research show that in some situations RUS have better performance than ROS [10]. But actually, RUS usually performance much badly than ROS because it may deleted some important data in majority class. Figure 6.3 shows the example that how random undersampling works.

DOI:10.6814/NCCU201901175

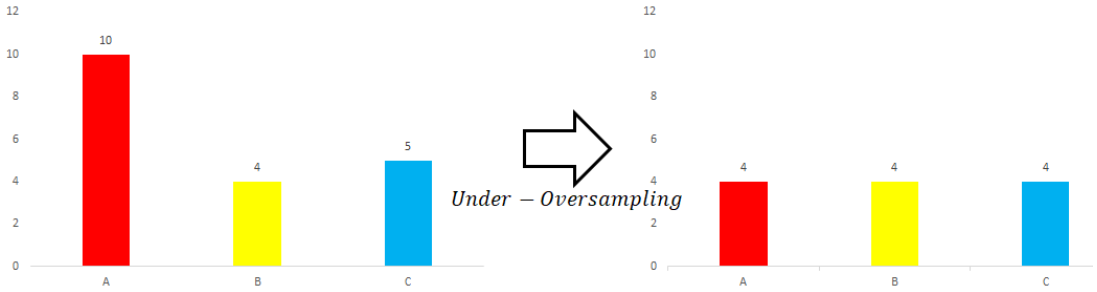


Figure 6.3: Random undersampling(RUS)

In data-level methods, we usually use ROS or SMOTE to eliminate class imbalance and have good effect on small data sets. But if our data set is much bigger or have extreme class imbalance i.e. ρ is large, performance of oversampling is not good any more because it will produce too many repeated samples led to overfitting and increase the training time. On the other hand, we believe that RUS can obtain better results on big imbalanced data set since it can reduce the training time and the number of sample in minority class is enough for model training.

6.2 Algorithm-level Methods

In this section, we will introduce 3 different loss function and cost sensitive learning. MFE and focal loss are two loss function which can be used on both binary classification and multi-classification. Due to the property of MFSE, this loss function only can be used on binary classification task. At last, cost sensitive learning can be used on both two classification task.

6.2.1 Mean false error(MFE)

Since MFE is inspired by the concepts of false positive rate and false negative rate [36], then we introduce confusion matrix with these two concepts first. Confusion matrix is a common indicator to judge a model is good or not. As shown in table 6.1, the index of vertical axis is actual condition, on other hand, the index of horizontal axis is predicted condition.

The meaning of true positive, false positive, false negative and true negative are correct prediction of positive samples, incorrect prediction of negative samples, incorrect prediction of positive samples and correct prediction of negative samples, respectively. Take an example, we want to classify the pictures of dog and cat, pictures of dog are positive samples and

DOI:10.6814/NCCU201901175

	Condition Positive	Condition Negative
Predicted Condition Positive	True Positive	False Positive
Predicted Condition Negative	False Negative	True Negative

Table 6.1: Confusion matrix

negative samples are cat pictures. If we predict dog pictures as dog and cat pictures as cat, then these condition are called true positive and true negative, respectively. But if we have wrong prediction like take dog pictures as cat and cat pictures as dog, then these condition are called false negative and false positive which they are also called Type I error and Type II error.

The loss function MFE is focus on the false condition and is much more sensitive than MSE. MFE is combined with two error, mean false positive error(FPE) and mean false negative error(FNE), the following are formula of them,

$$FPE = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{y_i} - F_\theta(\boldsymbol{x_i})||^2$$

$$FNE = \frac{1}{P}\sum_{i=1}^{P} ||\boldsymbol{y_i} - F_\theta(\boldsymbol{x_i})||^2$$

$$MFE = FPE + FNE$$

where N and P are the number of negative samples and positive samples, respectively.

In this paper, we apply MFE in multi-classification task so we modify the MFE loss function as following

$$MFE = \sum_{c=1}^{C} FcE = \sum_{c=1}^{C} \frac{1}{N_c}\sum_{i=1}^{N_c} ||\boldsymbol{y_i} - F_\theta(\boldsymbol{x_i})||^2$$

, where N_c is the number of samples in c class and $c = 1, 2, ..., C$ where C is the number of class.

6.2.2 Mean squared false error(MSFE)

Since we want to get the high classification accuracy on positive class, then the false negative error must quite low [36]. Hence, Wang et al. design an improve loss function, mean squared false error(MFSE) which is more sensitive than MFE on the error of positive class. In MFE, we only ensure that the minimize the sum of FPE and FNE, but it is not enough to get the high classification accuracy on positive class. Usually, FPE contribute more loss than FNE in

DOI:10.6814/NCCU201901175

MFE since in imbalanced data set there are much more negative samples than positive samples. Hence, MFE is not sensitive enough to the error of positive class. To solve this problem, MFSE is presented and the following are the formula of MSFE.

$$\begin{aligned} MFSE &= FPE^2 + FNE^2 \\ &= \frac{1}{2}((FPE + FNE)^2 + (FPE - FNE)^2) \end{aligned}$$

As above, MFSE minimizes $(FPE + FNE)^2$ and $(FPE - FNE)^2$ at the same time, then the error of positive class and negative class will also minimize at the same time. As a result, we can have the same effect as MFE and guarantee the positive class accuracy simultaneously. However, the property that MFSE minimizes both $(FPE + FNE)^2$ and $(FPE - FNE)^2$ is not exist in multi-classification task, so we only use MFSE on binary classification task.

6.2.3 Focal loss

Lin et al. propose a new loss function: focal loss which is modified from traditional cross-entropy(CE) [26]. Focal loss can reduce the weight of samples which is easy to classify and let the model focus on the samples which is hard to classify. Here, we will follow Lin et al. to introduce loss function form CE to focal loss. In binary classification, the following is the formula of binary cross-entropy.

$$L(\theta) = -\frac{1}{k}\sum_{i=1}^{k}[y_i \log(F_\theta(x_i)) + (1 - y_i)\log(1 - F_\theta(x_i))]$$

For convenience, we let

$$L(\theta) = L(p_i) = -\frac{1}{k}\sum_{i=1}^{k}\log(p_i), \text{ where } p_i = \begin{cases} F_\theta(x_i), \text{ if } y_i = 1 \\ 1 - F_\theta(x_i), \text{ if } y_i = 0 \end{cases}$$

Then in order to control the weight of the positive and negative samples to the loss, we add a new hyper parameter α_i, where $\alpha_i = \alpha$ when $y_i = 1$ and $\alpha_i = 1 - \alpha$ when $y_i = 0$. Usually, negative samples are much more than positive samples i.e. the number of i which y_i is 1 is more than the number of i that y_i is 0, then we will set α between 0.5 to 1 such that it can increase the

DOI:10.6814/NCCU201901175

weight of positive samples to loss. Hence, we have

$$L_{\alpha_i}(p_i) = -\frac{1}{k}\sum_{i=1}^{k}\alpha_i \log(p_i), \text{ where } p_i = \begin{cases} F_\theta(x_i), & \text{if } y_i = 1 \\ 1 - F_\theta(x_i), & \text{if } y_i = 0 \end{cases}$$

Although $L_{\alpha_i}(p_i)$ can control the weight of the positive and negative samples to the loss, it can not control the weight of samples which are easy to classify and hard to classify, so Lin et al. proposed the focal loss

$$FL(p_i) = -\frac{1}{k}\sum_{i=1}^{k}(1-p_i)^\gamma \log(p_i), \text{ where } p_i = \begin{cases} F_\theta(x_i), & \text{if } y_i = 1 \\ 1 - F_\theta(x_i), & \text{if } y_i = 0 \end{cases}$$

We call γ as focusing parameter which $\gamma \geq 0$ and $(1-p_i)^\gamma$ is modulating factor. Clearly, we can see that when a sample has wrong classification then p_i is small and $(1-p_i)^\gamma$ approaches to 1. For example, if $y_j = 0$ and wrong classification happened, then p_i must less than 0.5, so p_i is small and $(1-p_i)^\gamma$ is close to 1. Therefore, the loss will not change intensely compared with original binary cross-entropy. Furthermore, when p_i approaches to 1, then modulating factor is small so the loss it contributed is small too. Focal loss restrict the loss of samples which are easy to classify, that is, it can raise the contribution of samples that are hard to classify. Figure 6.4 shows that the performance with different focusing parameter γ and note that focal loss is traditional cross-entropy when $\gamma = 0$.

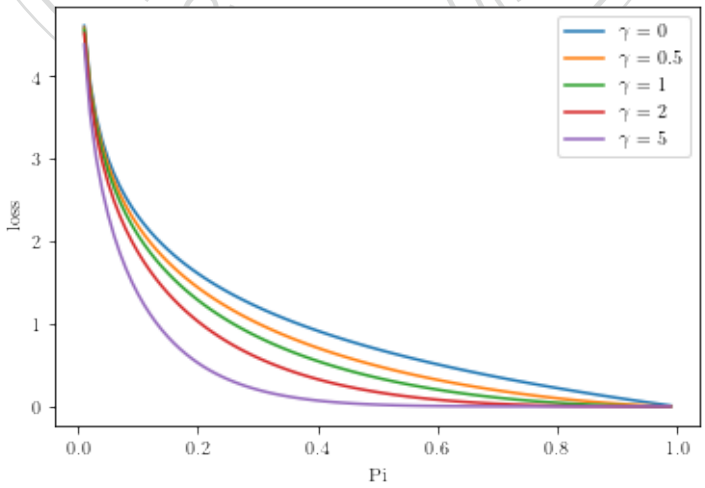


Figure 6.4: Focal loss

DOI:10.6814/NCCU201901175

Finally, Lin et al. find that if we combined $L_{\alpha_i}(p_i)$ and $FL(p_i)$, then it can adjust the weight of positive and negative samples to loss and control the loss of samples which are easy to classify. So we have,

$$FL_{\alpha_i}(p_i) = -\frac{1}{k}\sum_{i=1}^{k}\alpha_i(1-p_i)^{\gamma}\log(p_i), \text{ where } p_i = \begin{cases} F_\theta(x_i), \text{ if } y_i = 1 \\ 1 - F_\theta(x_i), \text{ if } y_i = 0 \end{cases}$$

In this paper, we use focal loss on multi-classification task and do not set α_i as above. We set $\alpha = (\alpha_1, \alpha_2, ..., \alpha_C)$ where C is the number of class so the focal loss function on multi-classification task is as below.

$$MFL_\alpha(F_\theta(\boldsymbol{x_i})) = -\frac{1}{k}\sum_{i=1}^{k}\sum_{c=1}^{C}\alpha_j y_{ic}(1 - F_\theta(x_{ic}))^{\gamma}\log(F_\theta(x_{ic}))$$

6.2.4 Cost sensitive learning

In binary classification, we often treat the loss of classification errors as the same, but it is wrong in real world. For example, we have a classifier to distinguish someone has a cancer or not, positive sample means get a cancer and negative sample is not. If we predict someone does not have cancer and he has cancer in real, then it may not be a big deal. But if someone has cancer and the prediction is not, then he may die because our wrong classification. Hence, the cost of misclassification may not be the same and it will change depends on different situations [11,13].

Cost sensitive learning give different cost to different misclassification as shown in table 6.2 which is cost matrix in binary classification [27]. In cost matrix, $C(c, j)$ means the cost of misclassifying that misclassifis a sample of c th class to j th, note that $C(c, j) = 0$ when $c = j$.

	Predicted Condition Negative	Predicted Condition Positive
Condition Negative	$C(0,0)$	$C(0,1)$
Condition Positive	$C(1,0)$	$C(1,1)$

Table 6.2: Cost matrix

There are many ways to apply cost sensitive learning to deep learning [13]. In this paper, we will use threshold moving apply in testing stage after the classifier is already trained [11,22, 40]. In general, we will set a threshold t for our classifier, if output is greater than t, then it is considered positive sample, otherwise, negative sample, usually $t = 0.5$. So threshold moving adds the concept of cost of misclassification and moves the threshold t to let the minority class

DOI:10.6814/NCCU201901175

can easy to classify. In the case of cancer patient, threshold moving moves the threshold toward inexpensive misclassification i.e. someone does not have cancer and predict he has cancer, so that the samples with high costs become hard to misclassify. Threshold moving will use original data set to train and add concept of cost sensitive learning in testing stage.

The output of our neural network is $F_\theta(x_{i,c})$ for all $c = 1, 2, ..., C$, which is an probability and C is the number of classes, after threshold moving we will get a new output $F^*_\theta(x_{i,c})$ for all $c = 1, 2, ..., C$ and it belongs to c^* class by calculated

$$\arg\max_c F^*_\theta(x_{i,c})$$

Clearly, the sum of all possibility is equal to 1 and we have

$$\sum_{c=1}^{C} F_\theta(x_{i,c}) = \sum_{c=1}^{C} F^*_\theta(x_{i,c}) = 1$$

The concept of cost sensitive learning is added in threshold moving, so we consider the effect of $C(c, j)$ and get the new output by following equation

$$F^*_\theta(x_{i,c}) = \beta \sum_{j=1}^{C} F_\theta(x_{i,c}) C(c, j),$$

where β is a parameter to let $F^*_\theta(x_{i,c})$ fit the probability equation $\sum_{c=1}^{C} F^*_\theta(x_{i,c}) = 1$.

Cost sensitive learning can also use on sampling [16], changing learning rate [22], modify the output [13] and create a new loss function [13,27]. Through the ratio of different $C(c, j)$, we can decide how many sample we need to duplicate or delete in new data set. Change learning rate or modify the output will led neural network to notice the point of cost in training stage. On the other hand, different from traditional loss function, we can minimize the total cost of misclassification to train our neural network. There are still many spaces where can work hard, we can combine cost sensitive learning with other methods to address imbalanced data and compare the performance of each different model in future.

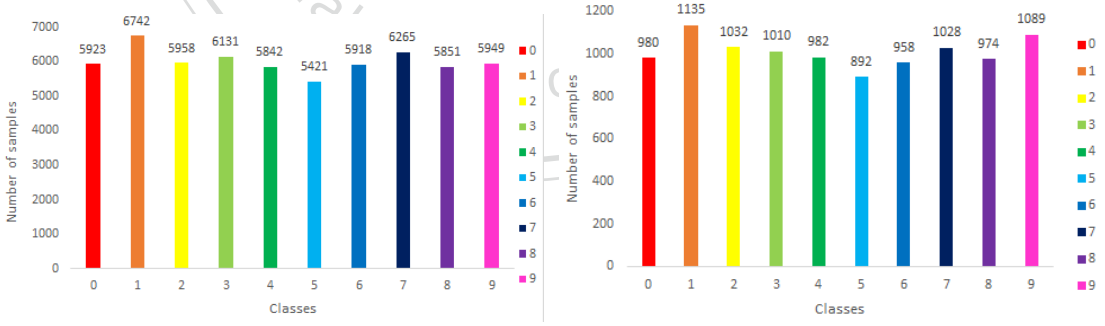
DOI:10.6814/NCCU201901175

Chapter 7

Experiment for Multi-classification Task

In this chapter, we will introduce the structure of our models. We design an imbalanced data set from MNIST which have imbalance rate $\rho = 2500$ and use CNN for training. This model M_b is our baseline model which have bad performance because of imbalanced data and we will use 7 different methods to create other 7 models to prove that our solution is work and compare their performance.

MNIST is a data set which contains 10 classes about handwriting images from 0 to 9. There are 60000 and 10000 images in training set and testing set, respectively. MNIST is a balance model on both training set and testing set, and all classes have more than 5000 samples. Figure 7.1 shows that the samples number of each class on training set and testing set.



(a) Training set (b) Testing set

Figure 7.1: Sample number of MNIST

Each images have size 28×28 and its label represents its number as shown in figure 7.2.

DOI:10.6814/NCCU201901175

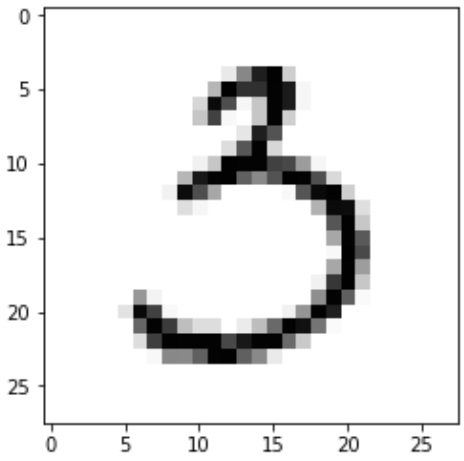


Figure 7.2: Sample from MNIST

7.1 Baseline Model

We create an imbalanced data set from MNIST by choose 5 minority class which is $0, 1, 4, 6, 7$ randomly with imbalance rate $\rho = 2500$ in this section. Figure 7.3 shows that the number of samples in imbalanced MNIST.

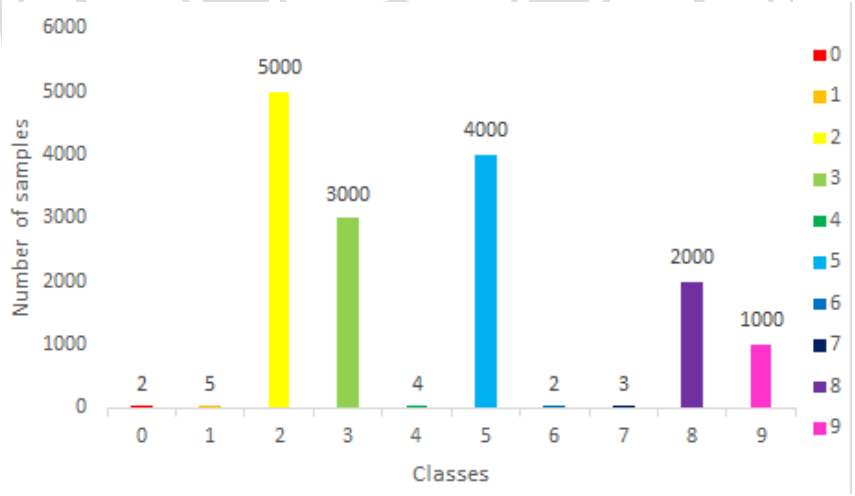


Figure 7.3: Imbalanced MNIST

After create an imbalanced data set, we use CNN to train our imbalanced data set and hope to get bad performance so that we can use other method to adjust it. In this baseline model, we have three convolutional layers and they all connect max pooling layers. After the input images pass through layers as above, it will be flatten and through two dense layers, finally get the output which represents the probability that the input belongs to each classes. We set the size of

DOI:10.6814/NCCU201901175

filters in all convolutional layers is 3×3, pooling size is 2×2, dropout rate is 0.2 and each layer contains $32, 64, 128, 200, 10$ neurons, respectively. We choose Relu as our activation function in every convolutional layer and first dense layer, second dense layer we use softmax due to we want to get the probability output, figure 7.4 shows the structure of CNN model clearly.

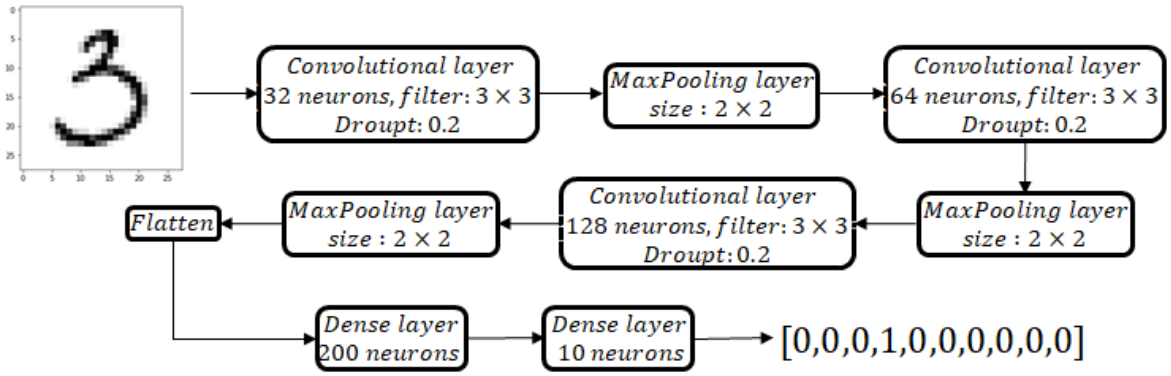


Figure 7.4: Structure of baseline CNN model

We use stochastic gradient decent as our optimizer, set learning rate $\eta = 0.05$ and loss function is mean square error(MSE). The number of input is 15016, batch size is 100 and trained 200 epochs. M_b is training by training set which we created is imbalanced, but test with the original testing set which is balanced. After 5 trainings, we find that M_b have great performance and get 99.04% average accuracy on training set, but we guess this is just an illusion because the average accuracy on testing set is worse i.e. 48.63%. Table 7.1 shows the performance of M_b which contains the average accuracy of training set, testing set, minority class, respectively and in figure 7.5 we can find that at class $0, 1, 4, 6, 7$ have bad performance in confusion matrix due to the number of samples is small.

	Training	Testing	0	1	4	6	7
M_b	99.04%	48.63%	0%	0%	0%	0%	0%

Table 7.1: Average accuracy of M_b

DOI:10.6814/NCCU201901175

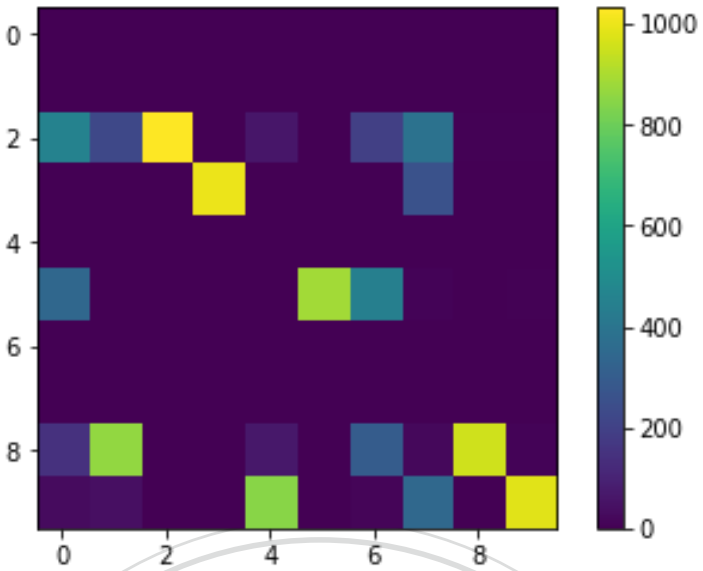


Figure 7.5: Confusion matrix of M_b

To improve the M_b, we will use 7 different methods to adjust the model and compare their effect.

7.2 Random-Oversampling Model

In this section, we adjust our training data set and let it become a balanced data set. As we introduce in section 6.1.1 we will choose samples randomly from minority classes and copy the samples until the sample number of minority class is equal to the sample number of majority class. After ROS, the dataset is balanced and we train it as the same structure of M_b and call this new model M_{os}. The model M_{os} also have good performance on training set which get 99.76% average accuracy, and the performance on testing set is better than M_b, that is, 79.44% average accuracy, we surmise that the overfitting happened so that the accuracy is not good enough. Table 7.2 shows that the comparison of average accuracy between M_{os} and M_b, and M_{os} is better than baseline model M_b.

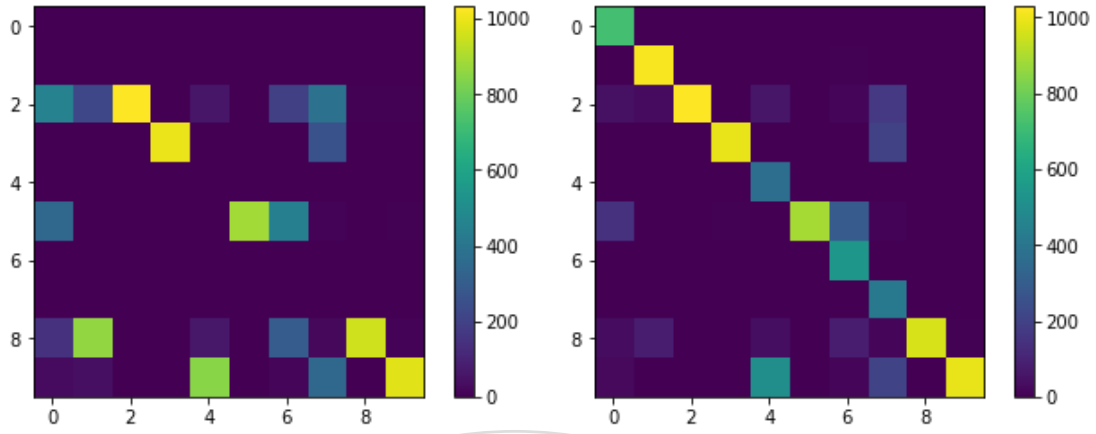
	Training	Testing	0	1	4	6	7
M_b	99.04%	48.63%	0%	0%	0%	0%	0%
M_{os}	99.76%	79.44%	74.11%	89.56%	37.47%	56.8%	40.44%

Table 7.2: Average accuracy of M_b and M_{os}

We discover that ROS adjust the accuracy on minority classes $0, 1, 4, 6, 7$ in confusion

DOI:10.6814/NCCU201901175

matrix shown in figure 7.6.



(a) M_b (b) M_{os}

Figure 7.6: Confusion matrix of M_b and M_{os}

7.3 Synthetic Minority Over-sampling Technique Model

Different from ROS, we will create an artificial minority samples to prevent overfitting. We follow the algorithm on section 6.1.2 and produce many samples of minority so that the dataset is balanced. We can see that the picture of new sample of class 1 produced by SMOTE in figure 7.7, looks like two pictures of class 1 overlap.

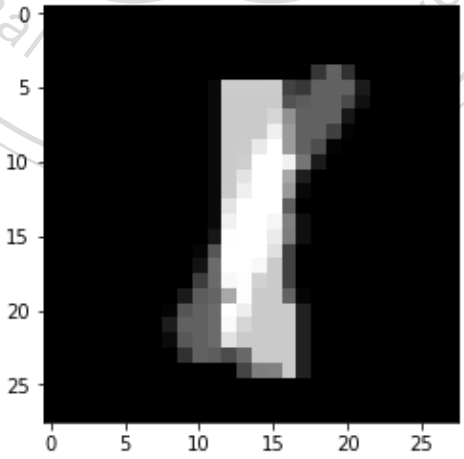


Figure 7.7: Average accuracy of M_b and M_{os}

Similarly, we use the same structure as M_b to train our model M_{sm} and get 99.73% average accuracy on training set and testing set performs like ROS which have 77.54% average accuracy,

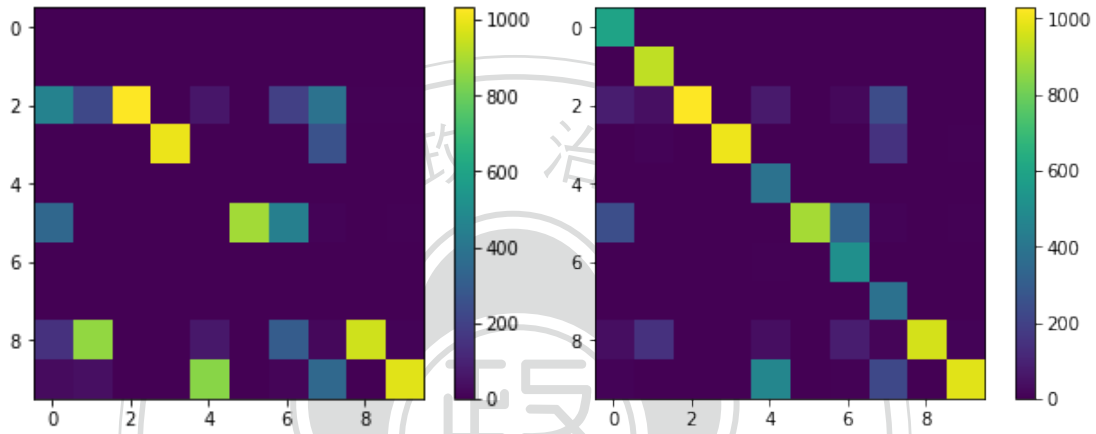
DOI:10.6814/NCCU201901175

and the comparison between M_b and M_{sm} shown in table 7.3.

	Training	Testing	0	1	4	6	7
M_b	99.04%	48.63%	0%	0%	0%	0%	0%
M_{sm}	99.73%	76.8%	60.85%	82.07%	39.5%	53.94%	37.33%

Table 7.3: Average accuracy of M_b and M_{sm}

Figure 7.8 shows the comparison of confusion matrix between our model M_{sm} and M_b, we can find that it also improve the accuracy of minority class.



(a) M_b (b) M_{sm}

Figure 7.8: Confusion matrix of M_b and M_{sm}

Although the performance of M_{sm} is better than M_b, but since it create fake samples and some of them are not look like original number, the interpretability of the model is greatly reduced and the performance is also not perfect.

7.4 Random-Undersampling Model

Section $7.1 - 7.3$ is the method of oversampling, this section we will use undersampling to try to address the bad performance of model which caused by imbalanced data set. As in section 6.1.3, we will delete the sample of majority class until the data set is balanced. Similarly, we use same structure of CNN to train our model M_{us} but since it is hard to converge, so we change the training epochs as 500. M_{us} get very low accuracy in many classes on testing set as shown in table 7.4 and its confusion matrix also shows the bad predicted result in figure 7.9.

DOI:10.6814/NCCU201901175

	0	1	2	3	4	5	6	7	8	9
M_{us}	56.5%	95.42%	59.49%	0%	0%	0%	0%	0%	0%	0%

Table 7.4: Average accuracy of every class in M_{us}

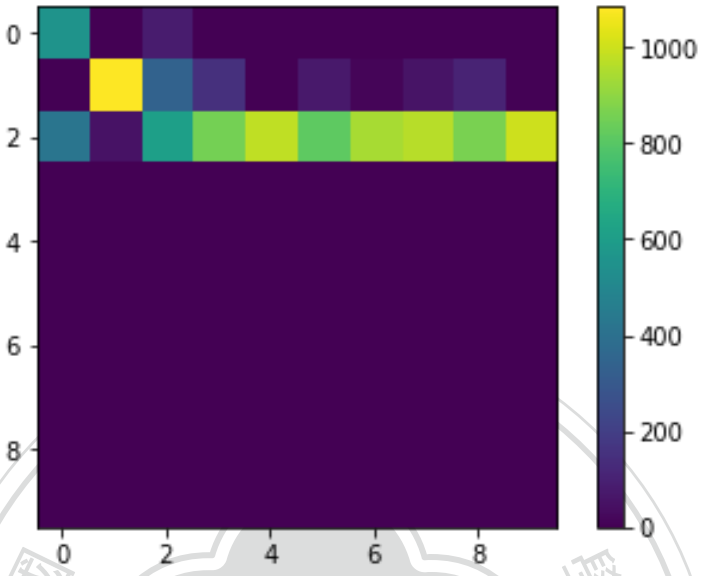


Figure 7.9: Confusion matrix of M_{us}

We believe that the bad performance of RUS due to the sample number of training set is too small and our model can not capture the feature of picture. Hence, we consider RUS is not a suitable method of our problem, we think it is suitable on a larger data set, which after RUS it also have enough samples to train. But RUS is not all have disadvantages, it save our training time and is quicker than M_b for 180 times.

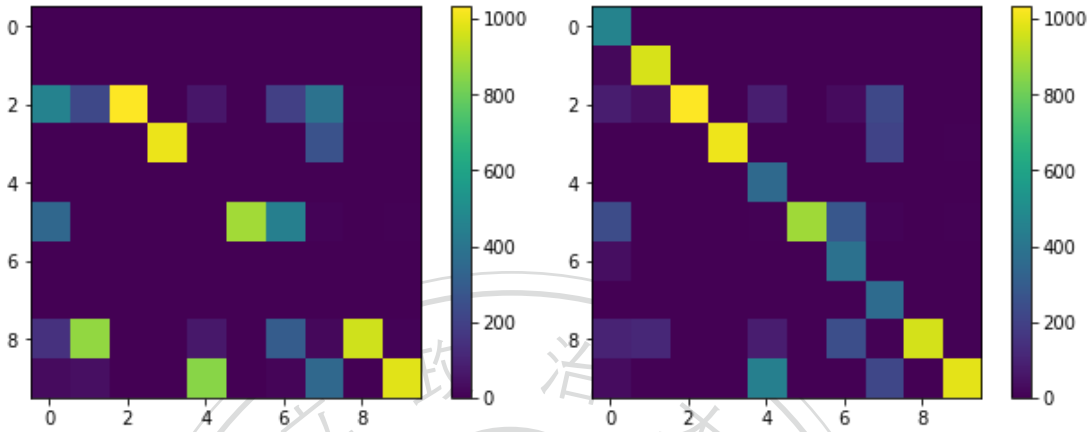
7.5 Mean False Error Model

Different from section $7.1 - 7.4$ is data-level method, in this section we use new loss function MFE as we introduced in section 6.2.1. MFE is more sensitive to the loss that contributed by minority class. We use the same structure as M_b to train our model M_{fe} and it also has high average accuracy 99.92 on training set and the performance on testing is better than M_b too which get 75.55 average accuracy. The performance of comparison between M_b and our mean false error model M_{fe} shown in Table 7.5 and we can find it increases the accuracy of minority in confusion matrix shown in figure 7.10.

DOI:10.6814/NCCU201901175

	Training	Testing	0	1	4	6	7
M_b	99.04%	48.63%	0%	0%	0%	0%	0%
M_{fe}	99.92%	75.55%	58.3%	81.13%	36.47%	44.58%	38.59%

Table 7.5: Average accuracy of M_b and M_{fe}



(a) M_b (b) M_{fe}

Figure 7.10: Confusion matrix of M_b and M_{fe}

7.6 Focal Loss Model

Same as section 7.5, we use new loss function, focal loss let the model be more sensitive to minority class. Another advantage for focal loss is that it can reduce the contribution of sample which is easy to classify as we mentioned in section 6.2.3. In this section we will try different number of focusing parameter γ and α. Different from Lin et al. α in our model is not between 0 and 1, we set $\alpha = (a, a, 1, 1, a, 1, a, a, 1, 1)$ which we give weight a for minority class and 1 for majority class where $a = 1, 5, 10, 50, 100$. On the other hand, we set $\gamma = 0, 0.5, 1, 2, 5$. Note that when $a = 1$ and $\gamma = 0$ the focal loss is categorical cross-entropy which we are familiar. The following are the performance of M_{fl} with different parameters.

DOI:10.6814/NCCU201901175

1. $\gamma = 0$:

	Training	Testing	0	1	4	6	7
$a = 1$	99.99%	74.46%	55.13%	78.26%	37.04%	41.87%	36.12%
$a = 5$	99.99%	77.88%	65.04%	85.9%	44.84%	46.53%	39.49%
$a = 10$	99.99%	78.29%	65.99%	87.8%	44.1%	44.25%	43.46%
$a = 50$	99.99%	79.31%	64.18%	86.42%	55%	47.53%	43.65%
$a = 100$	99.97%	79.73%	68.13%	88.52%	48.18%	53.87%	42.27%

Table 7.6: Average accuracy of M_{fl} with $\gamma = 0$

2. $\gamma = 0.5$:

	Training	Testing	0	1	4	6	7
$a = 1$	99.98%	74.83%	54.68%	80.93%	38.42%	41.39%	36.04%
$a = 5$	99.98%	77.96%	63.19%	86.88%	41.99%	51.66%	39.21%
$a = 10$	99.98%	77.64%	68.56%	88.22%	40.5%	45%	37.13%
$a = 50$	99.99%	78.05%	60.58%	86.14%	50.26%	46.61%	41.12%
$a = 100$	99.98%	77.84%	64.56%	88.04%	40.62%	49.99%	38.86%

Table 7.7: Average accuracy of M_{fl} with $\gamma = 0.5$

3. $\gamma = 1$:

	Training	Testing	0	1	4	6	7
$a = 1$	99.98%	74.07%	55.22%	81.06%	36.96%	35.8%	32.35%
$a = 5$	99.98%	75.91%	57.58%	85.33%	38.28%	41.89%	38.84%
$a = 10$	99.97%	77.65%	63.48%	88.1%	38.97%	50.47%	38.34%
$a = 50$	99.98%	78.24%	61.4%	86.25%	51.29%	46.17%	40.77%
$a = 100$	99.97%	77.23%	63.4%	84.89%	43.33%	47.92%	36.82%

Table 7.8: Average accuracy of M_{fl} with $\gamma = 1$

4. $\gamma = 2$:

	Training	Testing	0	1	4	6	7
$a = 1$	99.95%	72.85%	55.77%	80.18%	36.22%	27.15%	32.05%
$a = 5$	99.97%	75.34%	59.41%	82.77%	38.04%	38.72%	37.48%
$a = 10$	99.98%	74.65%	61.18%	83.16%	33.35%	32.54%	38.86%
$a = 50$	99.98%	76.38%	58.36%	84.18%	42.31%	42.41%	40.3%
$a = 100$	99.96%	77.15%	63.89%	85.84%	43%	44.54%	38.4%

Table 7.9: Average accuracy of M_{fl} with $\gamma = 2$

DOI:10.6814/NCCU201901175

5. $\gamma = 5$:

	Training	Testing	0	1	4	6	7
$a = 1$	99.83%	69.21%	41.07%	75.34%	24.8%	25.36%	28.73%
$a = 5$	99.85%	71.61%	48.46%	78.69%	28.99%	27.59%	35.15%
$a = 10$	99.88%	72.23%	53.32%	79.48%	31.76%	27.86%	33.22%
$a = 50$	99.81%	73.22%	57.66%	78.3%	38.97%	29.93%	31.95%
$a = 100$	99.83%	72.58%	49.62%	85.31%	34.84%	31.54%	28.43%

Table 7.10: Average accuracy of M_{fl} with $\gamma = 5$

Table 7.11 shows that the average accuracy of testing set with different parameter a and $gamma$ in model M_{fl}. We find that when $a = 100$ and $\gamma = 0$, the performance of model is best.

	$a = 1$	$a = 5$	$a = 10$	$a = 50$	$a = 100$
$\gamma = 0$	74.46%	77.88%	78.29%	79.31%	79.73%
$\gamma = 0.5$	74.83%	77.96%	77.64%	78.05%	77.84%
$\gamma = 1$	74.07%	75.91%	77.65%	78.24%	77.23%
$\gamma = 2$	72.85%	75.34%	74.65%	76.38%	77.15%
$\gamma = 5$	69.21%	71.68%	72.23%	73.22%	72.58%

Table 7.11: Average accuracy of M_{fl} with different parameters

Furthermore, we find that almost every model has higher accuracy when a is bigger and γ is smaller. We can see detailed in figure 7.11.

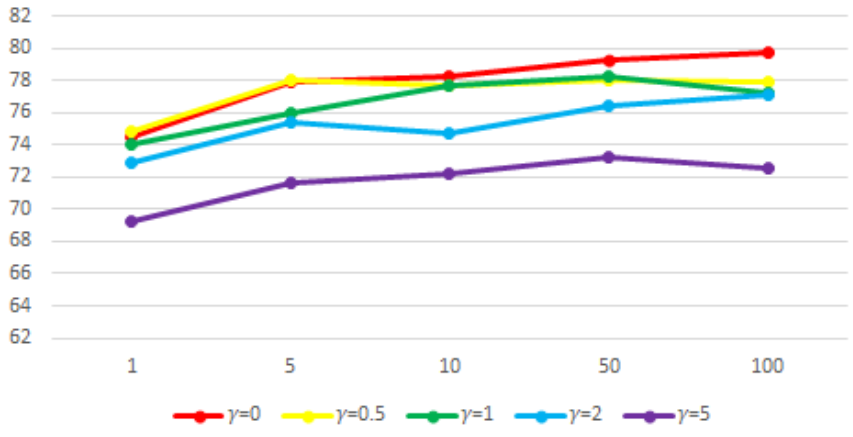


Figure 7.11: Average accuracy of M_{fl} with different parameters

DOI:10.6814/NCCU201901175

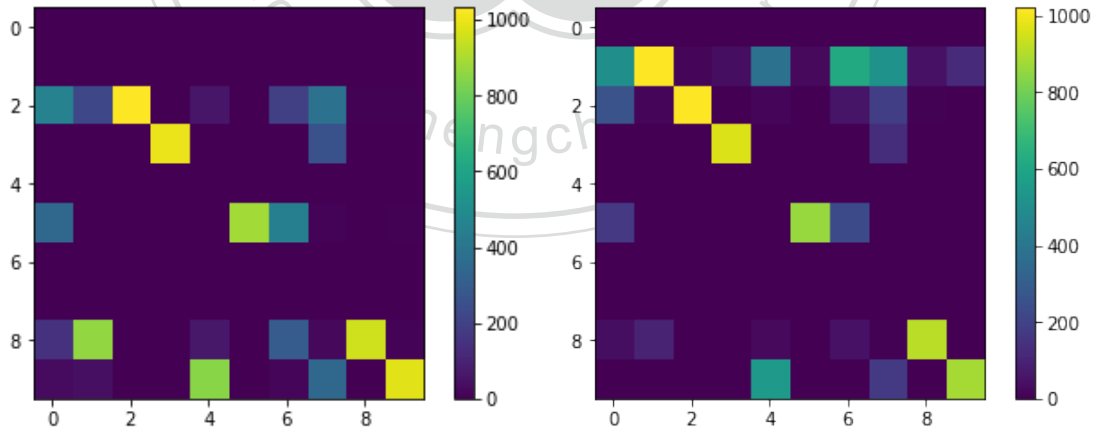
7.7 Cost Sensitive Learning Model

In this section, we add the concept of cost sensitive learning in threshold moving as we mentioned in section 6.2.4. According to the confusion matrix of baseline model in figure 7.5, we can find that where the misclassification happened. For example, the sample of class 0 almost classified to class $2, 4$ and 8. Hence, we can increase the cost of misclassification that 0 misclassified to $2, 4$ and 8 to arise the probability that classified to 0. But after experiment, we find that is hard to create a cost matrix to let the model be sensitive to all minority class. Because if we increase the predicted probability of some minority class, it will effect the probability of other minority class even cause worse situation. Therefore, after try and error many times, we just set $C(1,2) = 1000, C(1,8) = 2000, C(i,j) = 1$ when $i \neq j$ and $C(i,j) = 0$ when $i = j$ as we mentioned in section 6.2.4.

As in table 7.12, we can see that the cost sensitive learning model M_c improve the accuracy of class 1 but the accuracy of other minority class still get 0%.

	0	1	4	6	7
M_b	0%	0%	0%	0%	0%
M_c	0%	89.94%	0%	0%	0%

Table 7.12: Average accuracy of M_b and M_c



(a) M_b (b) M_c

Figure 7.12: Confusion matrix of M_b and M_c

Figure 7.12 shows the comparison of confusion matrix between M_b and M_c that M_c improves the performance of prediction on class 1. We believe that cost sensitive learning model

DOI:10.6814/NCCU201901175

can improve the performance of prediction well in binary classification task.

7.8 Result for Multi-classification Task

In multi-classification task, we use 6 methods to adjust our model, some of them have a good effect and the other may not suit for our data set. Table 7.13 shows that the average accuracy of different methods on minority class $0, 1, 4, 6, 7$.

	Class 0	Class 1	Class 4	Class 6	Class 7
M_b	0%	0%	0%	0%	0%
M_{os}	74.11%	89.56%	37.47%	56.8%	40.44%
M_{sm}	60.85%	82.07%	39.5%	53.94%	37.33%
M_{us}	56.5%	95.42%	0%	0%	0%
M_{fe}	58.3%	81.13%	36.47%	44.58%	38.59%
M_{fl}	68.13%	88.52%	48.18%	53.87%	42.27%
M_c	0%	89.94%	0%	0%	0%

Table 7.13: Average accuracy of minority class with different models

According to table 7.13, we can find that M_{os}, M_{sm}, M_{fe} and M_{fl} have great performance, they increase the average accuracy of minority classes successfully. M_{us} have bad performance may due to it delete to many important samples on majority class. Because of multi-classification task, if we change the cost on M_c, it will cause a lot of effect on other class, so cost sensitive learning is difficult to achieve on our task.

Focal loss is a suitable loss function for imbalanced data set, we believe that M_{fe} have the best performance on our imbalanced data set, we can see the comparison of average accuracy in figure 7.13 in detail.

DOI:10.6814/NCCU201901175

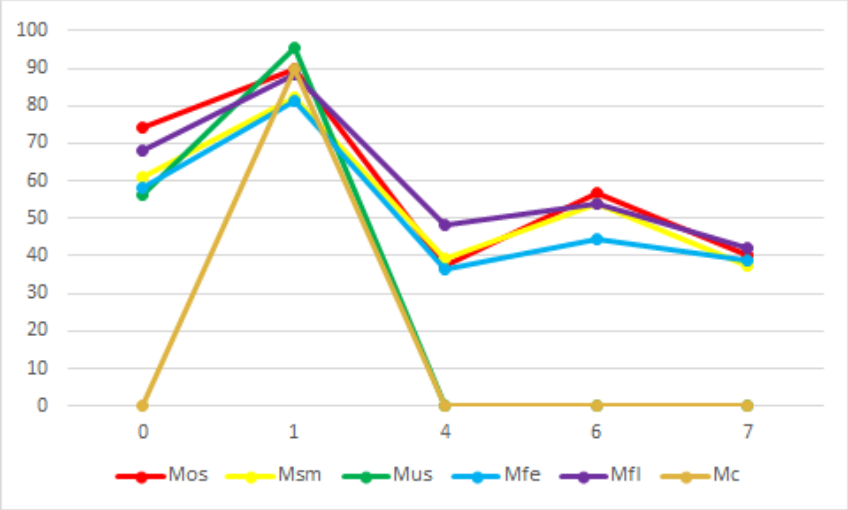


Figure 7.13: Comparison of average accuracy of minority classes with different methods

DOI:10.6814/NCCU201901175

Chapter 8

Experiment for Binary Classification Task

As in chapter 7, we have the same structure CNN model but only have two outputs 0 and 1. In this chapter, we also have baseline model M_{b2} and we will adjust it by 7 different methods. We believe that some methods performs bad in multi-classification task will performs well in binary classification.

8.1 Baseline Model

We create a binary imbalanced data set form MNIST by choose class 0 as minority class and class 1 as majority class with imbalance rate $\rho = 2500$, we can see the sample number of different class in figure 8.1.

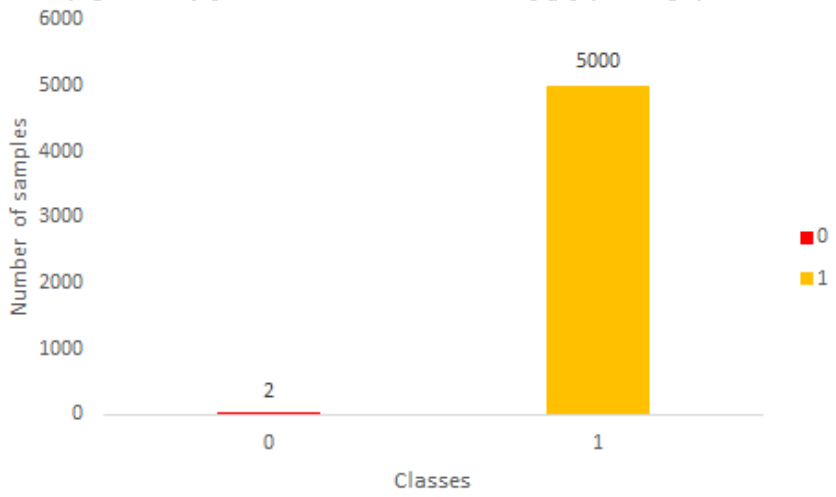


Figure 8.1: Binary imbalanced MNIST

We use same structure as section 7.1 and M_{b2} has great performance on training set with

DOI:10.6814/NCCU201901175

99.96% average accuracy but only has 53.66% on testing set. Table 8.1 shows the performance of M_b with average accuracy of training set,testing set, class 0 and class 1, respectively.

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%

Table 8.1: Average accuracy of M_{b2}

We can find that the samples of class 0 are all predicted to class 1 in confusion matrix in figure 8.2. Hence, we need to adjust our model, let it be more sensitive to minority class.

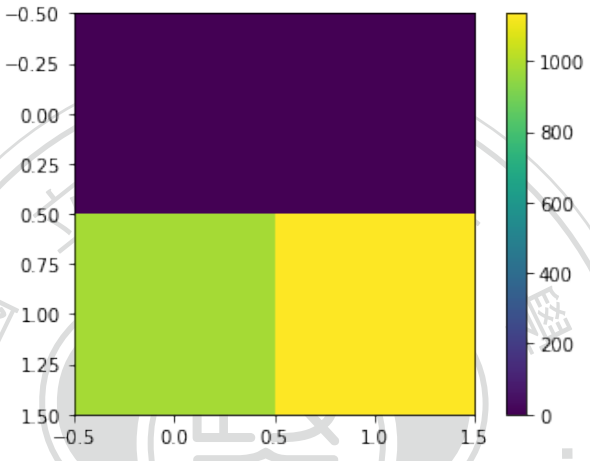


Figure 8.2: Confusion matrix of M_{b2}

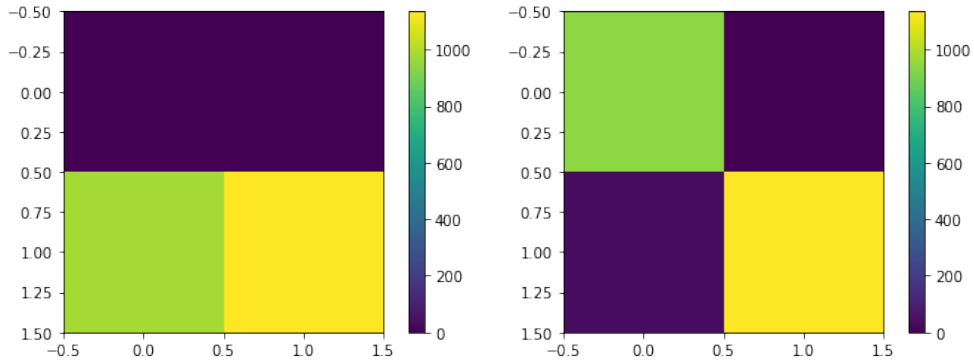
8.2 Random-Oversampling Model

In this section, we use ROS as we mentioned in section 6.1.1 to copy the sample in class 0 until the data set is balanced. M_{os2} get good performance on training set and testing set which have 100% and 98.25% average accuracy, respectively. Table 8.2 and figure 8.3 shows the comparison of M_{b2} and M_{os2} on average accuracy and confusion matrix. As result, ROS improve the average accuracy of minority class 0 and get the great result.

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%
M_{os2}	100%	98.25%	96.22%	100%

Table 8.2: Average accuracy of M_{b2} and M_{os2}

DOI:10.6814/NCCU201901175



(a) M_b (b) M_{os2}

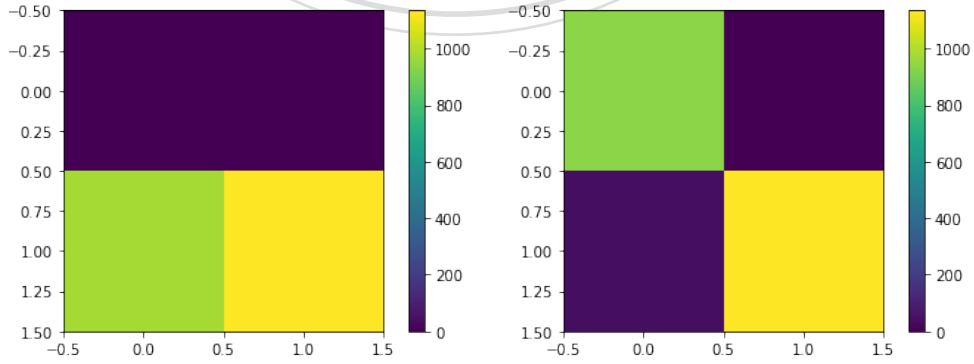
Figure 8.3: Confusion matrix of M_{b2} and M_{os2}

8.3 Synthetic Minority Over-sampling Technique Model

Same as ROS, SMOTE will balance the data set but it does not just copy the sample, it create an artificial sample to prevent overfitting as we mentioned in section 6.1.2. The model M_{sm2} also get high average accuracy 100% and 98.01% on training set and testing set. The comparison of average accuracy between M_{b2} and M_{sm2} shows on table 8.3 and we can see that it improve the accuracy of class 0 in confusion matrix shows on figure 8.4.

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%
M_{sm2}	100%	98.01%	95.71%	100%

Table 8.3: Average accuracy of M_{b2} and M_{sm2}



(a) M_b (b) M_{sm2}

Figure 8.4: Confusion matrix of M_{b2} and M_{sm2}

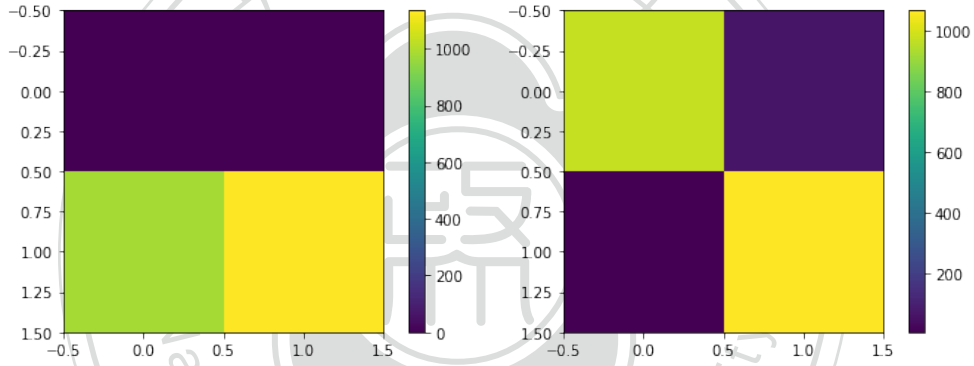
DOI:10.6814/NCCU201901175

8.4 Random-undersampling(RUS)

Different from section $8.2 - 8.3$ which use oversampling to let the model balanced, we use undersampling in this section. As we mentioned in section 6.1.3, we delete the sample in majority until the data set is balanced. Unlike in multi-classification task, the model M_{us2} which use RUS has great performance on binary classification task. Table 8.4 and figure 8.5 show that the effect of M_{us2} which is better than M_{b2}.

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%
M_{us2}	100%	96.69%	99.59%	94.18%

Table 8.4: Average accuracy of M_{b2} and M_{us2}



(a) M_b (b) M_{us2}

Figure 8.5: Confusion matrix of M_{b2} and M_{us2}

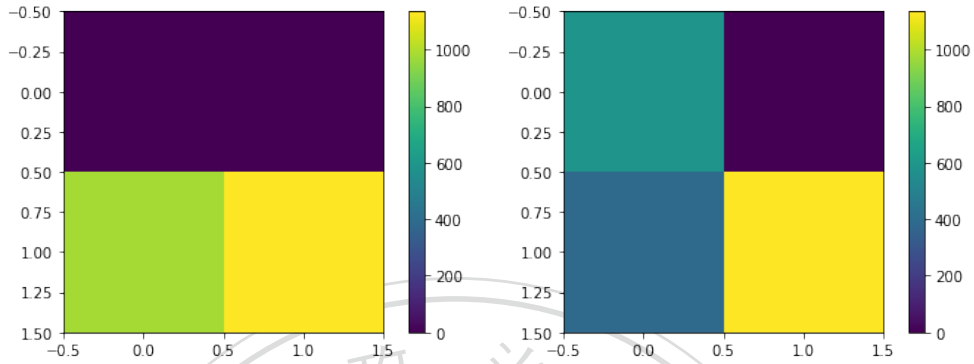
8.5 Mean False Error Model

In this section, we will use new loss function to let the model be more sensitive to minority class 0. As in section 6.2.1, mean false error calculate the loss of every class and increase the contribution of minority class loss to let the model M_{fe2} improve the average accuracy of minority class. In order to let the model converge, we adjust our epochs to 1000. We find that M_{fe2} have great performance on training set and testing set with 100% and 81.34%, respectively. But actually, it just get 59.75% average accuracy on minority class 0. The comparison of performance between M_{b2} and M_{fe2} shown on table 8.5 and figure 8.6.

DOI:10.6814/NCCU201901175

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%
M_{fe2}	100%	81.34%	59.75%	100%

Table 8.5: Average accuracy of M_{b2} and M_{fe2}



(a) M_{b2} (b) M_{fe2}

Figure 8.6: Confusion matrix of M_{b2} and M_{fe2}

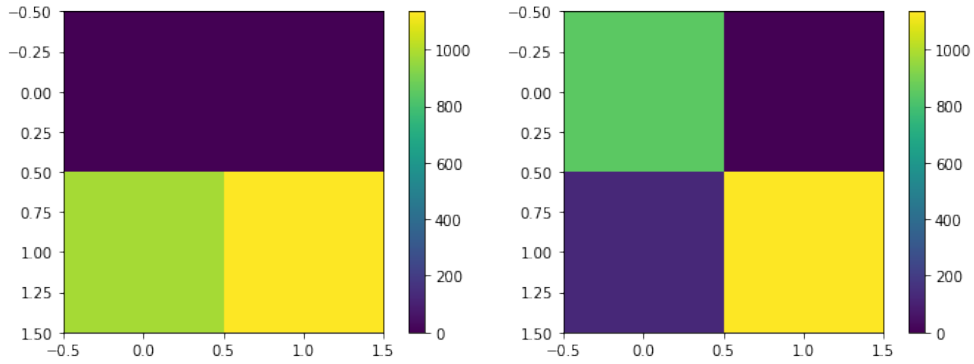
8.6 Mean Squared False Error Model

Mean squared false error is improved version of MFE, but it only can be used on binary classification task. M_{fse} can be more sensitive to minority class and improve the accuracy of minority class at the same time as we mentioned in section 6.2.2. In order to let the model converge, we adjust the learning rate $\eta = 0.001$ and training 40 epochs. The comparison of performance between M_{b2} and M_{fse} is shown on the table 8.6 and figure 8.7. We can see that M_{fse} increase the average accuracy of class 0.

	Training	Testing	0	1
M_{b2}	99.96%	53.66%	0%	100%
M_{fse}	99.98%	93.29%	85.52%	100%

Table 8.6: Average accuracy of M_{b2} and M_{fse}

DOI:10.6814/NCCU201901175



(a) M_{b2} (b) M_{fse}

Figure 8.7: Confusion matrix of M_{b2} and M_{fse}

8.7 Focal Loss Model

In this section, we apply a new loss function focal loss on binary classification task. As in section 6.2.3, focal loss can reduce the contribution of sample which is easy to classify and let the model to be more sensitive to minority class. We also set $a = 1, 5, 10, 50, 100$ and $\gamma = 0, 0.5, 1, 2, 5$ as section 7.6. Note that when $a = 1$ and $\gamma = 0$ is categorical cross-entropy which is common loss function for classification task. Table 8.7 shows that the performance of the model M_{fl2} with average accuracy of testing set about different a and γ. The following are the performance of M_{fl2} with different parameters.

1. $\gamma = 0$:

	Training	Testing	0	1
$a = 1$	100%	93.69%	86.38%	100%
$a = 5$	100%	95.85%	91.05%	100%
$a = 10$	100%	95.13%	89.5%	100%
$a = 50$	100%	96.95%	93.42%	100%
$a = 100$	100%	96.81%	93.13%	100%

Table 8.7: Average accuracy of M_{fl2} with $\gamma = 0$

DOI:10.6814/NCCU201901175

2. $\gamma = 0.5$:

	Training	Testing	0	1
$a = 1$	100%	92.71%	84.28%	100%
$a = 5$	100%	94.42%	87.97%	100%
$a = 10$	100%	94.95%	89.12%	100%
$a = 50$	100%	96.72%	92.93%	100%
$a = 100$	100%	96.38%	92.2%	100%

Table 8.8: Average accuracy of M_{fl2} with $\gamma = 0.5$

3. $\gamma = 1$:

	Training	Testing	0	1
$a = 1$	100%	91.93%	82.38%	100%
$a = 5$	100%	94.8%	88.79%	100%
$a = 10$	100%	95.22%	89.68%	100%
$a = 50$	100%	96.64%	92.75%	100%
$a = 100$	100%	96.61%	92.69%	100%

Table 8.9: Average accuracy of M_{fl2} with $\gamma = 1$

4. $\gamma = 2$:

	Training	Testing	0	1
$a = 1$	100%	90.44%	79.38%	100%
$a = 5$	100%	94.06%	87.2%	100%
$a = 10$	100%	95.49%	90.28%	100%
$a = 50$	100%	96.8%	93.09%	100%
$a = 100$	100%	96.6%	92.67%	100%

Table 8.10: Average accuracy of M_{fl2} with $\gamma = 2$

5. $\gamma = 5$:

	Training	Testing	0	1
$a = 1$	100%	70.07%	35.42%	100%
$a = 5$	100%	89.8%	77.99%	100%
$a = 10$	100%	92.61%	84.07%	100%
$a = 50$	100%	95.63%	90.59%	100%
$a = 100$	100%	96.72%	93.93%	100%

Table 8.11: Average accuracy of M_{fl2} with $\gamma = 5$

DOI:10.6814/NCCU201901175

We find that when $a = 50$ and $\gamma = 0$, the average accuracy of testing set is highest in table 8.12.

	$a = 1$	$a = 5$	$a = 10$	$a = 50$	$a = 100$
$\gamma = 0$	93.69%	95.85%	95.13%	96.95%	96.81%
$\gamma = 0.5$	92.71%	94.42%	94.95%	96.72%	96.38%
$\gamma = 1$	91.83%	94.8%	95.55%	96.64%	96.61%
$\gamma = 2$	90.44%	94.06%	95.49%	96.8%	96.6%
$\gamma = 5$	70.07%	89.8%	92.61%	95.63%	96.72%

Table 8.12: Average accuracy of M_{fl2} with different parameters

Figure 8.8 shows that when a is bigger the average accuracy of testing is higher and the effect of the number of γ is not obvious except when $\gamma = 5$ and $a = 1$ the accuracy is very low.

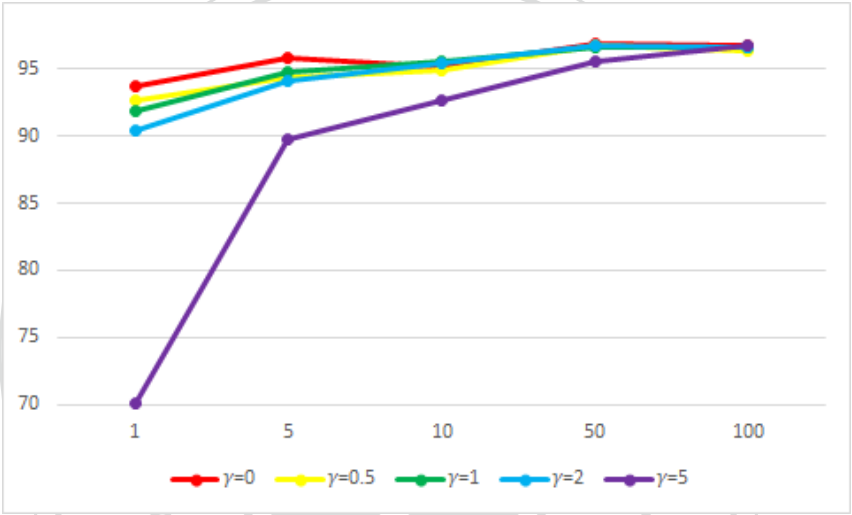
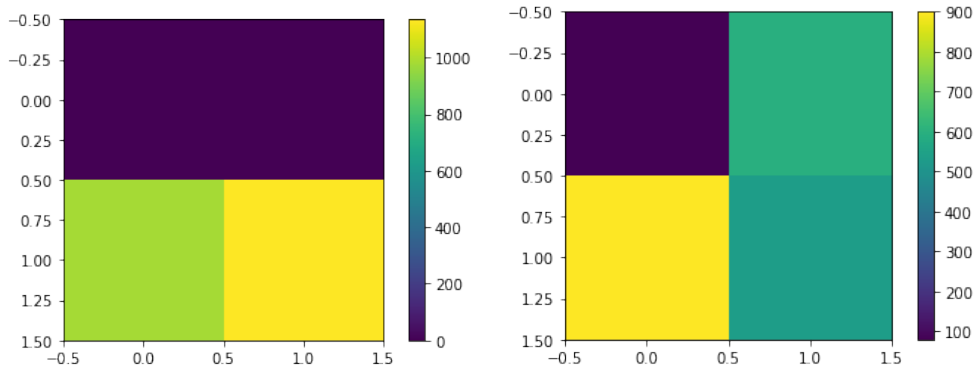


Figure 8.8: Average accuracy of M_{fl2} with different parameters

8.8 Cost Sensitive Learning Model

As in section 7.7, we add the concept of cost sensitive learning in threshold moving. According to the confusion matrix of M_{b2}, we can find that all sample of class 0 is predicted to class 1. Hence, we increase the cost of misclassification that misclassifies 0 to 1. We set $C(0,1) = 100$, $C(i,j) = 1$ when $i \neq j$ and $C(i,j) = 0$ when $i = j$. But after threshold moving, many sample of class 1 are predicted to class 0 and our purpose which let the model M_{c2} to predict the sample of class 0 correctly is not effect as shown in figure 8.9.

DOI:10.6814/NCCU201901175



(a) M_{b2} (b) M_{c2}

Figure 8.9: Confusion matrix of M_{b2} and M_{c2}

We try and error many times but still can not address this problem. Therefore, we calculate the average probability which M_{b2} predicted and shown in table 8.13.

	Sample of class 0	Sample of class 1
Predicted to class 0	0.51%	1.2%
Predicted to class 1	99.48%	98.79%

Table 8.13: Average probability predicted from M_{b2}

According to table 8.13, we can find that the sample of class 0 has higher probability to predict to class 1 than sample of class 1. So if we increase the probability of sample predicted to class 0, the sample from class 1 will much easier to predict to class 0. The model M_{c2} is fail in our data set, but we believe it will work on other problem.

8.9 Result for Binary Classification Task

In binary classification task, we use 7 different methods to adjust our baseline model M_{b2}. Some methods which have bad performance in multi-classification task are good in binary classification task. Table 8.14 shows that the average accuracy of different methods.

We can find that all methods have positive effect on our imbalanced data set except M_{c2}. Since the probability of sample of class 0 that be predicted to class 1 by baseline model M_{b2} is higher than the probability of sample of class 1 which is predicted to class 1, then if we increase the cost of misclassification that misclassify class 0 to class 1, then there are much more sample of class 1 will be predicted to class 0. Hence, our purpose that increase the accuracy of class 0

DOI:10.6814/NCCU201901175

	Class 0	Class 1
M_{b2}	0%	100%
M_{os2}	96.22%	100%
M_{sm2}	95.71%	100%
M_{us2}	99.59%	94.18%
M_{fe2}	59.75%	100%
M_{fse}	85.52%	100%
M_{fl2}	93.46%	100%
M_{c2}	8.16%	47.31%

Table 8.14: Average accuracy of class 0 and 1 with different models

can not be achieve. We believe that the cost sensitive learning is suitable for the model that can distinguish the difference of sample slightly.

According to table 8.14, we find that the data-level method is more suitable than algorithm-level method in binary classification of our imbalanced data set. In data-level method, the best model is M_{os2} and we can see the comparison of different model shown in figure 8.10 in detail.

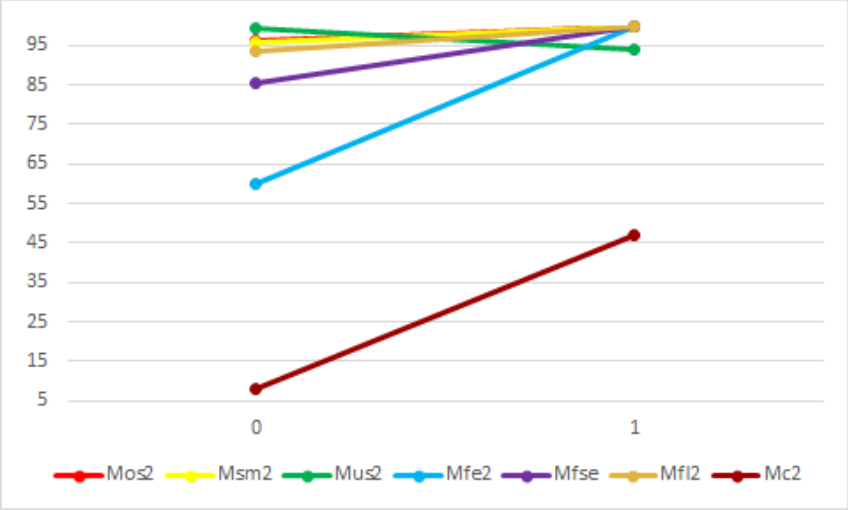


Figure 8.10: Comparison of average accuracy of class 0 and 1 with different methods

DOI:10.6814/NCCU201901175

Chapter 9

Conclusion

9.1 Contribution

In this paper, we have two main contribution. First, we review many different methods for imbalanced data problem and anomaly detection some of them are expended from binary classification to multi-classification. Second, we do the experiment on imbalanced MNIST with multi-classification task and binary classification.

We find that the new loss function, focal loss has best performance on multi-classification task. On the other hand, the best methods for binary classification task is ROS.

9.2 Future Work

We believe that the sample of MNIST is too easy to classifier so the overfitting is not happen and most method works effectively on our imbalanced MNIST. Try to use these methods on other much more complicated data set which has more classes and higher imbalanced rate may have different result.

Imbalanced data set not just happen in image classification but also happen on time-based data set. Test the performance of different methods on RNN models may have different effect.

DOI:10.6814/NCCU201901175

Appendix A

Python Code

In this section, we add the python code for our model in multi-classification task and binary classification task, some of code of binary classification task are similar to multi-classification task so we just put the multi-classification version.

A.1 Baseline Model

```
# coding: utf-8

# # 套件

# In[1]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
```


DOI:10.6814/NCCU201901175

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout


# # 生成不平衡資料&處理

# In[2]:


# 將訓練集分類
x0 = []
x1 = []
x2 = []
x3 = []
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
```


DOI:10.6814/NCCU201901175

```
y9 = []

for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
        x4.append(x_train[i])
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])
```


DOI:10.6814/NCCU201901175

```
# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
lx.append(len(x8))
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:
```


DOI:10.6814/NCCU201901175

```
lx #60000


# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5, replace = False)
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
```


DOI:10.6814/NCCU201901175

```
rx2 = []
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
for i in ri2:
    rx2.append(x2[i])
for i in ri3:
    rx3.append(x3[i])
for i in ri4:
    rx4.append(x4[i])
for i in ri5:
    rx5.append(x5[i])
for i in ri6:
    rx6.append(x6[i])
for i in ri7:
    rx7.append(x7[i])
for i in ri8:
    rx8.append(x8[i])
for i in ri9:
    rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
    y6[:2] + y7[:3] + y8[:2000] + y9[:1000]
```


DOI:10.6814/NCCU201901175

```
# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(10),lxr)


# In[10]:


# 訓練數據index
np.random.seed(10)
rxi = np.random.choice(range(15016), 15016, replace = False)


# In[11]:


# 生成訓練數據
x_trainib = []
y_trainib = []
for j in rxi:
    x_trainib.append((rx[j]/255)) #normalization
    y_trainib.append(np.utils.to_categorical(ry[j],10))
x_train_ib = np.array(x_trainib).reshape(15016,28,28,1)
```


DOI:10.6814/NCCU201901175

```
y_train_ib = np.array(y_trainib)


# # 神經網路建模&訓練

# In[12]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = "mean_squared_error", optimizer = SGD(lr =
    0.05), metrics = ["accuracy"])
```


DOI:10.6814/NCCU201901175

```
# In[13]:


model.summary()


# In[14]:


model.fit(x_train_ib, y_train_ib, batch_size = 100, epochs = 200)


# # 測試成果

# In[15]:


# 將測試集分類
xx0 = []
xx1 = []
xx2 = []
xx3 = []
xx4 = []
xx5 = []
xx6 = []
xx7 = []
xx8 = []
xx9 = []

yy0 = []
yy1 = []
```


DOI:10.6814/NCCU201901175

```
yy2 = []
yy3 = []
yy4 = []
yy5 = []
yy6 = []
yy7 = []
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
    if y_test[i] == 4:
        xx4.append(x_test[i])
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
```


DOI:10.6814/NCCU201901175

```
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
    if y_test[i] == 9:
        xx9.append(x_test[i])
        yy9.append(y_test[i])


# In[16]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
lxx.append(len(xx7))
lxx.append(len(xx8))
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
```


DOI:10.6814/NCCU201901175

```
lyy.append(len(yy8))
lyy.append(len(yy9))


# In[17]:


lxx


# In[18]:


plt.bar(range(10),lxx)


# In[19]:


# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)

score = model.evaluate(x_test1, y_test1)
print('loss:', score[0])
print('Accuracy:', score[1])


# In[20]:


model_json = model.to_json()
open('MNIST_baseline.json', 'w').write(model_json)
```


DOI:10.6814/NCCU201901175

```
model.save_weights('MNIST_baseline_wights.h5')
```

A.2 Random-Oversampling Model

```
# coding: utf-8

# # 套件

# In[1]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout


# # 生成不平衡資料&處理

# In[2]:
```


DOI:10.6814/NCCU201901175

```
# 將訓練集分類
x0 = []
x1 = []
x2 = []
x3 = []
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
y9 = []

for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
```


DOI:10.6814/NCCU201901175

```
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
        x4.append(x_train[i])
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])


# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
```


DOI:10.6814/NCCU201901175

```
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
lx.append(len(x8))
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:


lx #60000


# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:
```


DOI:10.6814/NCCU201901175

```
# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5, replace = False)
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
rx2 = []
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
```


DOI:10.6814/NCCU201901175

```
for i in ri2:
    rx2.append(x2[i])
for i in ri3:
    rx3.append(x3[i])
for i in ri4:
    rx4.append(x4[i])
for i in ri5:
    rx5.append(x5[i])
for i in ri6:
    rx6.append(x6[i])
for i in ri7:
    rx7.append(x7[i])
for i in ri8:
    rx8.append(x8[i])
for i in ri9:
    rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
    y6[:2] + y7[:3] + y8[:2000] + y9[:1000]


# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
```


DOI:10.6814/NCCU201901175

```
plt.bar(range(10),lxr)


# # ROS

# In[21]:


# 生成隨機index
np.random.seed(10)
oi0 = np.random.choice(range(2), 5000, replace = True)
oi1 = np.random.choice(range(5), 5000, replace = True)
oi3 = np.random.choice(range(3000), 5000, replace = True)
oi4 = np.random.choice(range(4), 5000, replace = True)
oi5 = np.random.choice(range(4000), 5000, replace = True)
oi6 = np.random.choice(range(2), 5000, replace = True)
oi7 = np.random.choice(range(3), 5000, replace = True)
oi8 = np.random.choice(range(2000), 5000, replace = True)
oi9 = np.random.choice(range(1000), 5000, replace = True)


# In[25]:


ox0 = []
ox1 = []
ox2 = rx2
ox3 = []
ox4 = []
ox5 = []
ox6 = []
ox7 = []
ox8 = []
```


DOI:10.6814/NCCU201901175

```
ox9 = []

for i in oi0:
   ox0.append(rx0[i])
for i in oi1:
   ox1.append(rx1[i])
for i in oi3:
   ox3.append(rx3[i])
for i in oi4:
   ox4.append(rx4[i])
for i in oi5:
   ox5.append(rx5[i])
for i in oi6:
   ox6.append(rx6[i])
for i in oi7:
   ox7.append(rx7[i])
for i in oi8:
   ox8.append(rx8[i])
for i in oi9:
   ox9.append(rx9[i])

ox = ox0 + ox1 + ox2 + ox3 + ox4 + ox5 + ox6 + ox7 + ox8 + ox9
oy = y0[:5000] + y1[:5000] + y2[:5000] + y3[:5000] + y4[:5000] +
   y5[:5000] + y6[:5000] + y7[:5000] + y8[:5000] + y9[:5000]


# In[ ]:


# 訓練數據index
np.random.seed(10)
oxi = np.random.choice(range(50000), 50000, replace = False)
```


DOI:10.6814/NCCU201901175

```
# In[ ]:


# 生成訓練數據
x_trainos = []
y_trainos = []
for j in oxi:
    x_trainos.append((ox[j]/255)) #normalization
    y_trainos.append(np_utils.to_categorical(oy[j],10))
x_train_os = np.array(x_trainos).reshape(50000,28,28,1)
y_train_os = np.array(y_trainos)


# # 神經網路建模&訓練

# In[ ]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
```


DOI:10.6814/NCCU201901175

```
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = "mean_squared_error", optimizer = SGD(lr =
    0.05), metrics = ["accuracy"])


# In[ ]:


model.summary()


# In[ ]:


model.fit(x_train_os, y_train_os, batch_size = 100, epochs = 200)


# # 測試成果

# In[ ]:


# 將測試集分類
xx0 = []
xx1 = []
```


DOI:10.6814/NCCU201901175

```
xx2 = []
xx3 = []
xx4 = []
xx5 = []
xx6 = []
xx7 = []
xx8 = []
xx9 = []

yy0 = []
yy1 = []
yy2 = []
yy3 = []
yy4 = []
yy5 = []
yy6 = []
yy7 = []
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
```


DOI:10.6814/NCCU201901175

```
    if y_test[i] == 4:
        xx4.append(x_test[i])
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
    if y_test[i] == 9:
        xx9.append(x_test[i])
        yy9.append(y_test[i])


# In[ ]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
```


DOI:10.6814/NCCU201901175

```
lxx.append(len(xx7))
lxx.append(len(xx8))
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
lyy.append(len(yy8))
lyy.append(len(yy9))
```

```
# In[ ]:
```

```
lxx
```

```
# In[ ]:
```

```
plt.bar(range(10),lxx)
```

```
# In[ ]:
```

```
# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)
```


DOI:10.6814/NCCU201901175

```
score = model.evaluate(x_test1, y_test1)
print('loss:', score[0])
print('Accuracy:', score[1])


# In[ ]:


model_json = model.to_json()
open('MNIST_ROS.json', 'w').write(model_json)
model.save_weights('MNIST_ROS_wights.h5')
```

A.3 Synthetic Minority Over-sampling Technique Model

```
# coding: utf-8

# # 套件

# In[1]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
```


DOI:10.6814/NCCU201901175

```
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout


# # 生成不平衡資料&處理

# In[2]:


# 將訓練集分類
x0 = []
x1 = []
x2 = []
x3 = []
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
y9 = []
```


DOI:10.6814/NCCU201901175

```
for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
        x4.append(x_train[i])
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])
```


DOI:10.6814/NCCU201901175

```
# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
lx.append(len(x8))
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:


lx #60000
```


DOI:10.6814/NCCU201901175

```
# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5, replace = False)
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
rx2 = []
```


DOI:10.6814/NCCU201901175

```
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
for i in ri2:
    rx2.append(x2[i])
for i in ri3:
    rx3.append(x3[i])
for i in ri4:
    rx4.append(x4[i])
for i in ri5:
    rx5.append(x5[i])
for i in ri6:
    rx6.append(x6[i])
for i in ri7:
    rx7.append(x7[i])
for i in ri8:
    rx8.append(x8[i])
for i in ri9:
    rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
    y6[:2] + y7[:3] + y8[:2000] + y9[:1000]
```


DOI:10.6814/NCCU201901175

```
# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(10),lxr)


# # SMOTE

# In[10]:


# 計算每個類別中的樣本跟其他同類別樣本的距離
rx0d = [[] for i in range(2)]
for i in range(2):
    for j in range(2):
        rx0d[i].append(np.linalg.norm(rx0[i] - rx0[j]))
rx1d = [[] for i in range(5)]
for i in range(5):
    for j in range(5):
        rx1d[i].append(np.linalg.norm(rx1[i] - rx1[j]))
rx3d = [[] for i in range(3000)]
for i in range(3000):
    for j in range(3000):
        rx3d[i].append(np.linalg.norm(rx3[i] - rx3[j]))
rx4d = [[] for i in range(4)]
```


DOI:10.6814/NCCU201901175

```
for i in range(4):
    for j in range(4):
        rx4d[i].append(np.linalg.norm(rx4[i] - rx4[j]))
rx5d = [[] for i in range(4000)]
for i in range(4000):
    for j in range(4000):
        rx5d[i].append(np.linalg.norm(rx5[i] - rx5[j]))
rx6d = [[] for i in range(2)]
for i in range(2):
    for j in range(2):
        rx6d[i].append(np.linalg.norm(rx6[i] - rx6[j]))
rx7d = [[] for i in range(3)]
for i in range(3):
    for j in range(3):
        rx7d[i].append(np.linalg.norm(rx7[i] - rx7[j]))
rx8d = [[] for i in range(2000)]
for i in range(2000):
    for j in range(2000):
        rx8d[i].append(np.linalg.norm(rx8[i] - rx8[j]))
rx9d = [[] for i in range(1000)]
for i in range(1000):
    for j in range(1000):
        rx9d[i].append(np.linalg.norm(rx9[i] - rx9[j]))


# In[11]:


# 將距離排序
rx0ds = []
for i in rx0d:
    rx0ds.append(sorted(i))
rx1ds = []
```


DOI:10.6814/NCCU201901175

```
for i in rx1d:
    rx1ds.append(sorted(i))
rx3ds = []
for i in rx3d:
    rx3ds.append(sorted(i))
rx4ds = []
for i in rx4d:
    rx4ds.append(sorted(i))
rx5ds = []
for i in rx5d:
    rx5ds.append(sorted(i))
rx6ds = []
for i in rx6d:
    rx6ds.append(sorted(i))
rx7ds = []
for i in rx7d:
    rx7ds.append(sorted(i))
rx8ds = []
for i in rx8d:
    rx8ds.append(sorted(i))
rx9ds = []
for i in rx9d:
    rx9ds.append(sorted(i))


# In[12]:


rx7ds[0]


# In[13]:
```


DOI:10.6814/NCCU201901175

```
# index of SMOTE
np.random.seed(10)
rx0i = np.random.choice(range(2), 4998, replace = True)
rx1i = np.random.choice(range(5), 4995, replace = True)
rx3i = np.random.choice(range(3000), 2000, replace = True)
rx4i = np.random.choice(range(4), 4996, replace = True)
rx5i = np.random.choice(range(4000), 1000, replace = True)
rx6i = np.random.choice(range(2), 4998, replace = True)
rx7i = np.random.choice(range(3), 4997, replace = True)
rx8i = np.random.choice(range(2000), 3000, replace = True)
rx9i = np.random.choice(range(1000), 4000, replace = True)

rx0dsi = np.random.choice(range(1,2), 4998, replace = True)
rx1dsi = np.random.choice(range(1,4), 4995, replace = True)
rx3dsi = np.random.choice(range(1,4), 2000, replace = True)
rx4dsi = np.random.choice(range(1,4), 4996, replace = True)
rx5dsi = np.random.choice(range(1,4), 1000, replace = True)
rx6dsi = np.random.choice(range(1,2), 4998, replace = True)
rx7dsi = np.random.choice(range(1,3), 4997, replace = True)
rx8dsi = np.random.choice(range(1,4), 3000, replace = True)
rx9dsi = np.random.choice(range(1,4), 4000, replace = True)


# In[14]:


# 找出R
rx0dR = []
for i in range(4998):
   for j in range(2):
      if ((rx0ds[rx0i[i]][rx0dsi[i]])==(np.linalg.norm(rx0[rx0i[i]]
         - rx0[j]))):
```


DOI:10.6814/NCCU201901175

```
            rx0dR.append(j)
rx1dR = []
for i in range(4995):
   for j in range(5):
      if ((rx1ds[rx1i[i]][rx1dsi[i]])==(np.linalg.norm(rx1[rx1i[i]]
         - rx1[j]))):
         rx1dR.append(j)
rx3dR = []
for i in range(2000):
   for j in range(3000):
      if ((rx3ds[rx3i[i]][rx3dsi[i]])==(np.linalg.norm(rx3[rx3i[i]]
         - rx3[j]))):
         rx3dR.append(j)
rx4dR = []
for i in range(4996):
   for j in range(4):
      if ((rx4ds[rx4i[i]][rx4dsi[i]])==(np.linalg.norm(rx4[rx4i[i]]
         - rx4[j]))):
         rx4dR.append(j)
rx5dR = []
for i in range(1000):
   for j in range(4000):
      if ((rx5ds[rx5i[i]][rx5dsi[i]])==(np.linalg.norm(rx5[rx5i[i]]
         - rx5[j]))):
         rx5dR.append(j)
rx6dR = []
for i in range(4998):
   for j in range(2):
      if ((rx6ds[rx6i[i]][rx6dsi[i]])==(np.linalg.norm(rx6[rx6i[i]]
         - rx6[j]))):
         rx6dR.append(j)
rx7dR = []
for i in range(4997):
```


DOI:10.6814/NCCU201901175

```
    for j in range(3):
        if ((rx7ds[rx7i[i]][rx7dsi[i]])==(np.linalg.norm(rx7[rx7i[i]]
            - rx7[j]))):
            rx7dR.append(j)
rx8dR = []
for i in range(3000):
    for j in range(2000):
        if ((rx8ds[rx8i[i]][rx8dsi[i]])==(np.linalg.norm(rx8[rx8i[i]]
            - rx8[j]))):
            rx8dR.append(j)
rx9dR = []
for i in range(4000):
    for j in range(1000):
        if ((rx9ds[rx9i[i]][rx9dsi[i]])==(np.linalg.norm(rx9[rx9i[i]]
            - rx9[j]))):
            rx9dR.append(j)


# In[15]:


# 製造R'
np.random.seed(10)
rx0R = []
for i in range(4998):
    l0 = np.random.random()
    rx0R.append(l0*rx0[rx0i[i]]+(1-l0)*rx0[rx0dR[i]])
rx1R = []
for i in range(4995):
    l1 = np.random.random()
    rx1R.append(l1*rx1[rx1i[i]]+(1-l1)*rx1[rx1dR[i]])
rx3R = []
for i in range(2000):
```


DOI:10.6814/NCCU201901175

```
    l3 = np.random.random()
    rx3R.append(l3*rx3[rx3i[i]]+(1-l3)*rx3[rx3dR[i]])
rx4R = []
for i in range(4996):
    l4 = np.random.random()
    rx4R.append(l4*rx4[rx4i[i]]+(1-l4)*rx4[rx4dR[i]])
rx5R = []
for i in range(1000):
    l5 = np.random.random()
    rx5R.append(l5*rx5[rx5i[i]]+(1-l5)*rx5[rx5dR[i]])
rx6R = []
for i in range(4998):
    l6 = np.random.random()
    rx6R.append(l6*rx6[rx6i[i]]+(1-l6)*rx6[rx6dR[i]])
rx7R = []
for i in range(4997):
    l7 = np.random.random()
    rx7R.append(l7*rx7[rx7i[i]]+(1-l7)*rx7[rx7dR[i]])
rx8R = []
for i in range(3000):
    l8 = np.random.random()
    rx8R.append(l8*rx8[rx8i[i]]+(1-l8)*rx8[rx8dR[i]])
rx9R = []
for i in range(4000):
    l9 = np.random.random()
    rx9R.append(l9*rx9[rx9i[i]]+(1-l9)*rx9[rx9dR[i]])


# In[31]:


plt.imshow(rx1R[1], cmap = 'Greys_r')
```


DOI:10.6814/NCCU201901175

```
# In[17]:


# SMOTE後的數據集
sx = rx0 + rx0R + rx1 + rx1R + rx2 + rx3 + rx3R + rx4 + rx4R + rx5 +
   rx5R + rx6 + rx6R + rx7 + rx7R + rx8 + rx8R + rx9 + rx9R
sy = y0[:5000] + y1[:5000] + y2[:5000] + y3[:5000] + y4[:5000] +
   y5[:5000] + y6[:5000] + y7[:5000] + y8[:5000] + y9[:5000]


# In[18]:


# 訓練數據index
np.random.seed(10)
sxi = np.random.choice(range(50000), 50000 , replace = False)


# In[19]:


# 生成訓練數據
x_trainsm = []
y_trainsm = []
for j in sxi:
   x_trainsm.append((sx[j]/255)) #normalization
   y_trainsm.append(np_utils.to_categorical(sy[j],10))
x_train_sm = np.array(x_trainsm).reshape(50000,28,28,1)
y_train_sm = np.array(y_trainsm)


# # 神經網路建模&訓練
```


DOI:10.6814/NCCU201901175

```
# In[20]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = "mean_squared_error", optimizer = SGD(lr =
    0.05), metrics = ["accuracy"])


# In[21]:
```


DOI:10.6814/NCCU201901175

```
model.summary()


# In[22]:


model.fit(x_train_sm, y_train_sm, batch_size = 100, epochs = 200)


# # 測試成果

# In[23]:


# 將測試集分類
xx0 = []
xx1 = []
xx2 = []
xx3 = []
xx4 = []
xx5 = []
xx6 = []
xx7 = []
xx8 = []
xx9 = []

yy0 = []
yy1 = []
yy2 = []
yy3 = []
yy4 = []
yy5 = []
```


DOI:10.6814/NCCU201901175

```
yy6 = []
yy7 = []
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
    if y_test[i] == 4:
        xx4.append(x_test[i])
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
```


DOI:10.6814/NCCU201901175

```
    if y_test[i] == 9:
        xx9.append(x_test[i])
        yy9.append(y_test[i])


# In[24]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
lxx.append(len(xx7))
lxx.append(len(xx8))
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
lyy.append(len(yy8))
lyy.append(len(yy9))
```


DOI:10.6814/NCCU201901175

```
# In[25]:


lxx


# In[26]:


plt.bar(range(10),lxx)


# In[27]:


# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)

score = model.evaluate(x_test1, y_test1)
print('loss:', score[0])
print('Accuracy:', score[1])


# In[28]:


model_json = model.to_json()
open('MNIST_SMOTE.json', 'w').write(model_json)
model.save_weights('MNIST_SMOTE_wights.h5')
```


DOI:10.6814/NCCU201901175

A.4 Random-Undersampling Model

```
# coding: utf-8

# # 套件

# In[1]:


import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout


# # 生成不平衡資料&處理

# In[2]:


# 將訓練集分類
x0 = []
x1 = []
x2 = []
x3 = []
```


DOI:10.6814/NCCU201901175

```
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
y9 = []

for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
        x4.append(x_train[i])
```


DOI:10.6814/NCCU201901175

```
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])


# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
lx.append(len(x8))
```


DOI:10.6814/NCCU201901175

```
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:


lx #60000


# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5, replace = False)
```


DOI:10.6814/NCCU201901175

```
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
rx2 = []
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
for i in ri2:
    rx2.append(x2[i])
for i in ri3:
    rx3.append(x3[i])
for i in ri4:
```


DOI:10.6814/NCCU201901175

```
    rx4.append(x4[i])
for i in ri5:
    rx5.append(x5[i])
for i in ri6:
    rx6.append(x6[i])
for i in ri7:
    rx7.append(x7[i])
for i in ri8:
    rx8.append(x8[i])
for i in ri9:
    rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
    y6[:2] + y7[:3] + y8[:2000] + y9[:1000]


# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(10),lxr)


# # RUS
```


DOI:10.6814/NCCU201901175

```
# In[10]:


# 隨機挑選index(2)
np.random.seed(10)
ui0 = np.random.choice(range(2),2, replace = False)
ui1 = np.random.choice(range(5),2, replace = False)
ui2 = np.random.choice(range(5000),2, replace = False)
ui3 = np.random.choice(range(3000),2, replace = False)
ui4 = np.random.choice(range(4),2, replace = False)
ui5 = np.random.choice(range(4000),2, replace = False)
ui6 = np.random.choice(range(2),2, replace = False)
ui7 = np.random.choice(range(3),2, replace = False)
ui8 = np.random.choice(range(2000),2, replace = False)
ui9 = np.random.choice(range(1000),2, replace = False)


# In[11]:


# 生成RUS數據
ux0 = []
ux1 = []
ux2 = []
ux3 = []
ux4 = []
ux5 = []
ux6 = []
ux7 = []
ux8 = []
ux9 = []
for i in ui0:
    ux0.append(rx0[i])
```


DOI:10.6814/NCCU201901175

```
for i in ui1:
   ux1.append(rx1[i])
for i in ui2:
   ux2.append(rx2[i])
for i in ui3:
   ux3.append(rx3[i])
for i in ui4:
   ux4.append(rx4[i])
for i in ui5:
   ux5.append(rx5[i])
for i in ui6:
   ux6.append(rx6[i])
for i in ui7:
   ux7.append(rx7[i])
for i in ui8:
   ux8.append(rx8[i])
for i in ui9:
   ux9.append(rx9[i])

ux = ux0 + ux1 + ux2 + ux3 + ux4 + ux5 + ux6 + ux7 + ux8 + ux9
uy = y0[:2] + y1[:2] + y2[:2] + y3[:2] + y4[:2] + y5[:2] + y6[:2] +
   y7[:2] + y8[:2] + y9[:2]


# In[12]:


# 訓練數據index
np.random.seed(10)
uxi = np.random.choice(range(20), 20, replace = False)


# In[13]:
```


DOI:10.6814/NCCU201901175

```
# 生成訓練數據
x_trainus = []
y_trainus = []
for j in uxi:
    x_trainus.append((ux[j]/255)) #normalization
    y_trainus.append(np_utils.to_categorical(ry[j],10))
x_train_us = np.array(x_trainus).reshape(20,28,28,1)
y_train_us = np.array(y_trainus)


# # 神經網路建模&訓練

# In[14]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
```


DOI:10.6814/NCCU201901175

```
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = "mean_squared_error", optimizer = SGD(lr =
    0.05), metrics = ["accuracy"])


# In[15]:


model.summary()


# In[16]:


model.fit(x_train_us, y_train_us, batch_size = 100, epochs = 500)


# # 測試成果

# In[17]:


# 將測試集分類
xx0 = []
xx1 = []
xx2 = []
xx3 = []
```


DOI:10.6814/NCCU201901175

```
xx4 = []
xx5 = []
xx6 = []
xx7 = []
xx8 = []
xx9 = []

yy0 = []
yy1 = []
yy2 = []
yy3 = []
yy4 = []
yy5 = []
yy6 = []
yy7 = []
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
    if y_test[i] == 4:
        xx4.append(x_test[i])
```


DOI:10.6814/NCCU201901175

```
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
    if y_test[i] == 9:
        xx9.append(x_test[i])
        yy9.append(y_test[i])


# In[18]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
lxx.append(len(xx7))
lxx.append(len(xx8))
```


DOI:10.6814/NCCU201901175

```
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
lyy.append(len(yy8))
lyy.append(len(yy9))


# In[19]:


lxx


# In[20]:


plt.bar(range(10),lxx)


# In[21]:


# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)

score = model.evaluate(x_test1, y_test1)
```


DOI:10.6814/NCCU201901175

```
print('loss:', score[0])
print('Accuracy:', score[1])


# In[22]:


model_json = model.to_json()
open('MNIST_RUS.json', 'w').write(model_json)
model.save_weights('MNIST_RUS_wights.h5')
```

A.5 Mean False Error Model

```
# coding: utf-8

# # 套件

# In[1]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
```


DOI:10.6814/NCCU201901175

```
from keras.optimizers import SGD
from keras.layers import Dropout
from keras import backend as K
import tensorflow as tf


# # 生成不平衡資料&處理

# In[2]:


# 將訓練集分類
x0 = []
x1 = []
x2 = []
x3 = []
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
y9 = []
```


DOI:10.6814/NCCU201901175

```
for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
        x4.append(x_train[i])
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])
```


DOI:10.6814/NCCU201901175

```
# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
lx.append(len(x8))
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:


lx #60000
```


DOI:10.6814/NCCU201901175

```
# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5, replace = False)
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
rx2 = []
```


DOI:10.6814/NCCU201901175

```
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
   rx0.append(x0[i])
for i in ri1:
   rx1.append(x1[i])
for i in ri2:
   rx2.append(x2[i])
for i in ri3:
   rx3.append(x3[i])
for i in ri4:
   rx4.append(x4[i])
for i in ri5:
   rx5.append(x5[i])
for i in ri6:
   rx6.append(x6[i])
for i in ri7:
   rx7.append(x7[i])
for i in ri8:
   rx8.append(x8[i])
for i in ri9:
   rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
   y6[:2] + y7[:3] + y8[:2000] + y9[:1000]
```


DOI:10.6814/NCCU201901175

```
# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(10),lxr)


# In[10]:


# 訓練數據index
np.random.seed(10)
rxi = np.random.choice(range(15016),15016, replace = False)


# In[11]:


# 生成訓練數據
x_trainib = []
y_trainib = []
for j in rxi:
    x_trainib.append((rx[j]/255)) #normalization
    y_trainib.append(np.utils.to_categorical(ry[j],10))
x_train_ib = np.array(x_trainib).reshape(15016,28,28,1)
y_train_ib = np.array(y_trainib)
```


DOI:10.6814/NCCU201901175

```
# # MFE

# In[17]:


def mfe(y_true, y_pred):
    y_true0 = y_true[:,0:1]
    y_true1 = y_true[:,1:2]
    y_true2 = y_true[:,2:3]
    y_true3 = y_true[:,3:4]
    y_true4 = y_true[:,4:5]
    y_true5 = y_true[:,5:6]
    y_true6 = y_true[:,6:7]
    y_true7 = y_true[:,7:8]
    y_true8 = y_true[:,8:9]
    y_true9 = y_true[:,9:10]
    return (y_true0 * K.mean(K.square(y_true - y_pred))/2+y_true1 *
       K.mean(K.square(y_true - y_pred))/5
          +y_true2 * K.mean(K.square(y_true - y_pred))/5000+y_true3 *
             K.mean(K.square(y_true - y_pred))/3000
          +y_true4 * K.mean(K.square(y_true - y_pred))/4+y_true5 *
             K.mean(K.square(y_true - y_pred))/4000
          +y_true6 * K.mean(K.square(y_true - y_pred))/2 + y_true7 *
             K.mean(K.square(y_true - y_pred))/3
          +y_true8 * K.mean(K.square(y_true - y_pred))/2000+y_true9 *
             K.mean(K.square(y_true - y_pred))/1000)*15016


# # 神經網路建模&訓練

# In[21]:
```


DOI:10.6814/NCCU201901175

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = mfe, optimizer = SGD(lr = 0.05), metrics =
    ["accuracy"])


# In[22]:


model.summary()
```


DOI:10.6814/NCCU201901175

```
# In[ ]:


model.fit(x_train_ib, y_train_ib, batch_size = 100, epochs = 200)


# # 測試成果

# In[ ]:


# 將測試集分類
xx0 = []
xx1 = []
xx2 = []
xx3 = []
xx4 = []
xx5 = []
xx6 = []
xx7 = []
xx8 = []
xx9 = []

yy0 = []
yy1 = []
yy2 = []
yy3 = []
yy4 = []
yy5 = []
yy6 = []
yy7 = []
```


DOI:10.6814/NCCU201901175

```
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
    if y_test[i] == 4:
        xx4.append(x_test[i])
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
    if y_test[i] == 9:
        xx9.append(x_test[i])
```


DOI:10.6814/NCCU201901175

```
        yy9.append(y_test[i])


# In[ ]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
lxx.append(len(xx7))
lxx.append(len(xx8))
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
lyy.append(len(yy8))
lyy.append(len(yy9))


# In[ ]:
```


DOI:10.6814/NCCU201901175

```
lxx


# In[ ]:


plt.bar(range(10),lxx)


# In[ ]:


# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)

score = model.evaluate(x_test1, y_test1)
print('loss:', score[0])
print('Accuracy:', score[1])


# In[ ]:


model_json = model.to_json()
open('MNIST_MFE.json', 'w').write(model_json)
model.save_weights('MNIST_MFE_wights.h5')
```

A.6 Focal Loss Model

DOI:10.6814/NCCU201901175

```
# coding: utf-8

# # 套件

# In[1]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout
from keras import backend as K
import tensorflow as tf


# # 生成不平衡資料&處理

# In[2]:


# 將訓練集分類
x0 = []
x1 = []
x2 = []
```


DOI:10.6814/NCCU201901175

```
x3 = []
x4 = []
x5 = []
x6 = []
x7 = []
x8 = []
x9 = []

y0 = []
y1 = []
y2 = []
y3 = []
y4 = []
y5 = []
y6 = []
y7 = []
y8 = []
y9 = []

for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])
    if y_train[i] == 2:
        x2.append(x_train[i])
        y2.append(y_train[i])
    if y_train[i] == 3:
        x3.append(x_train[i])
        y3.append(y_train[i])
    if y_train[i] == 4:
```


DOI:10.6814/NCCU201901175

```
        x4.append(x_train[i])
        y4.append(y_train[i])
    if y_train[i] == 5:
        x5.append(x_train[i])
        y5.append(y_train[i])
    if y_train[i] == 6:
        x6.append(x_train[i])
        y6.append(y_train[i])
    if y_train[i] == 7:
        x7.append(x_train[i])
        y7.append(y_train[i])
    if y_train[i] == 8:
        x8.append(x_train[i])
        y8.append(y_train[i])
    if y_train[i] == 9:
        x9.append(x_train[i])
        y9.append(y_train[i])


# In[3]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))
lx.append(len(x2))
lx.append(len(x3))
lx.append(len(x4))
lx.append(len(x5))
lx.append(len(x6))
lx.append(len(x7))
```


DOI:10.6814/NCCU201901175

```
lx.append(len(x8))
lx.append(len(x9))
ly.append(len(y0))
ly.append(len(y1))
ly.append(len(y2))
ly.append(len(y3))
ly.append(len(y4))
ly.append(len(y5))
ly.append(len(y6))
ly.append(len(y7))
ly.append(len(y8))
ly.append(len(y9))


# In[4]:


lx #60000


# In[5]:


# 訓練集各類數量長條圖
plt.bar(range(10),lx)


# In[6]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
```


DOI:10.6814/NCCU201901175

```
ri1 = np.random.choice(range(6742),5, replace = False)
ri2 = np.random.choice(range(5958),5000, replace = False)
ri3 = np.random.choice(range(6131),3000, replace = False)
ri4 = np.random.choice(range(5842),4, replace = False)
ri5 = np.random.choice(range(5421),4000, replace = False)
ri6 = np.random.choice(range(5918),2, replace = False)
ri7 = np.random.choice(range(6265),3, replace = False)
ri8 = np.random.choice(range(5851),2000, replace = False)
ri9 = np.random.choice(range(5949),1000, replace = False)


# In[7]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
rx2 = []
rx3 = []
rx4 = []
rx5 = []
rx6 = []
rx7 = []
rx8 = []
rx9 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
for i in ri2:
    rx2.append(x2[i])
for i in ri3:
    rx3.append(x3[i])
```


DOI:10.6814/NCCU201901175

```
for i in ri4:
    rx4.append(x4[i])
for i in ri5:
    rx5.append(x5[i])
for i in ri6:
    rx6.append(x6[i])
for i in ri7:
    rx7.append(x7[i])
for i in ri8:
    rx8.append(x8[i])
for i in ri9:
    rx9.append(x9[i])

rx = rx0 + rx1 + rx2 + rx3 + rx4 + rx5 + rx6 + rx7 + rx8 + rx9
ry = y0[:2] + y1[:5] + y2[:5000] + y3[:3000] + y4[:4] + y5[:4000] +
    y6[:2] + y7[:3] + y8[:2000] + y9[:1000]


# In[8]:


# 不平衡資料各類樣本數
lxr = [2, 5, 5000, 3000, 4, 4000, 2, 3, 2000, 1000] #15015


# In[9]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(10),lxr)


# In[10]:
```


DOI:10.6814/NCCU201901175

```
# 訓練數據index
np.random.seed(10)
rxi = np.random.choice(range(15016), 15016, replace = False)


# In[11]:


# 生成訓練數據
x_trainib = []
y_trainib = []
for j in rxi:
    x_trainib.append((rx[j]/255)) #normalization
    y_trainib.append(np_utils.to_categorical(ry[j],10))
x_train_ib = np.array(x_trainib).reshape(15016,28,28,1)
y_train_ib = np.array(y_trainib)


# # Focal loss

# In[13]:


def focalloss(y_true, y_pred):

    g = 2.0
    a = tf.constant([[5],[5],[1],[1],[5],[1],[5],[5],[1],[1]], dtype
       = tf.float32)
    y_pred = tf.clip_by_value(y_pred, 1.e-7, 1. - 1.e-7)

    ce = tf.multiply(y_true, -tf.log(y_pred))
```


DOI:10.6814/NCCU201901175

```
    m = tf.pow(tf.subtract(1., y_pred), g)
    f = tf.matmul(tf.multiply(m, ce), a)
    loss = tf.reduce_mean(f)
    return loss


# # 神經網路建模&訓練

# In[16]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))                                    #output
```


DOI:10.6814/NCCU201901175

```
    10個結果
model.add(Activation('softmax'))

model.compile(loss = focalloss, optimizer = SGD(lr = 0.05), metrics
    = ["accuracy"])


# In[17]:


model.summary()


# In[18]:


model.fit(x_train_ib, y_train_ib, batch_size = 100, epochs = 200)


# # 測試成果

# In[ ]:


# 將測試集分類
xx0 = []
xx1 = []
xx2 = []
xx3 = []
xx4 = []
xx5 = []
xx6 = []
xx7 = []
```


DOI:10.6814/NCCU201901175

```
xx8 = []
xx9 = []

yy0 = []
yy1 = []
yy2 = []
yy3 = []
yy4 = []
yy5 = []
yy6 = []
yy7 = []
yy8 = []
yy9 = []

for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])
    if y_test[i] == 2:
        xx2.append(x_test[i])
        yy2.append(y_test[i])
    if y_test[i] == 3:
        xx3.append(x_test[i])
        yy3.append(y_test[i])
    if y_test[i] == 4:
        xx4.append(x_test[i])
        yy4.append(y_test[i])
    if y_test[i] == 5:
        xx5.append(x_test[i])
        yy5.append(y_test[i])
```


DOI:10.6814/NCCU201901175

```
    if y_test[i] == 6:
        xx6.append(x_test[i])
        yy6.append(y_test[i])
    if y_test[i] == 7:
        xx7.append(x_test[i])
        yy7.append(y_test[i])
    if y_test[i] == 8:
        xx8.append(x_test[i])
        yy8.append(y_test[i])
    if y_test[i] == 9:
        xx9.append(x_test[i])
        yy9.append(y_test[i])


# In[ ]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))
lxx.append(len(xx2))
lxx.append(len(xx3))
lxx.append(len(xx4))
lxx.append(len(xx5))
lxx.append(len(xx6))
lxx.append(len(xx7))
lxx.append(len(xx8))
lxx.append(len(xx9))
lyy.append(len(yy0))
lyy.append(len(yy1))
lyy.append(len(yy2))
```


DOI:10.6814/NCCU201901175

```
lyy.append(len(yy3))
lyy.append(len(yy4))
lyy.append(len(yy5))
lyy.append(len(yy6))
lyy.append(len(yy7))
lyy.append(len(yy8))
lyy.append(len(yy9))


# In[ ]:


lxx


# In[ ]:


plt.bar(range(10),lxx)


# In[ ]:


# 測試平衡資料
x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_test1 = np_utils.to_categorical(y_test, 10)

score = model.evaluate(x_test1, y_test1)
print('loss:', score[0])
print('Accuracy:', score[1])
```


DOI:10.6814/NCCU201901175

```
# In[ ]:


model_json = model.to_json()
open('MNIST_focalloss25.json', 'w').write(model_json)
model.save_weights('MNIST_focalloss25_wights.h5')
```

A.7 Cost Sensitive Learning Model

```
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')


# In[2]:


from keras.models import model_from_json
from keras.optimizers import SGD


# In[3]:


model = model_from_json(open('MNIST_baseline.json').read())
```


DOI:10.6814/NCCU201901175

```
model.load_weights('MNIST_baseline_wights.h5')


# In[4]:


model.compile(loss = "mean_squared_error", optimizer = SGD(lr =
    0.05), metrics = ["accuracy"])


# In[5]:


from keras.datasets import mnist
from keras.utils import np_utils
(x_train, y_train), (x_test, y_test) = mnist.load_data()


# In[6]:


x_test1 = (x_test/255).reshape(10000, 28, 28, 1)
y_true = y_test
y_pred = model.predict_classes(x_test1)


# In[7]:


from sklearn.metrics import confusion_matrix


# In[8]:
```


DOI:10.6814/NCCU201901175

```
C=confusion_matrix(y_pred, y_true)


# In[9]:


plt.imshow(C)
plt.colorbar()


# In[10]:


C


# In[11]:


TP = []
for i in range(10):
    tp = 0
    for j in range(10):
        if i == j :
            tp = tp + C[i][j]
            TP.append(tp)


# In[12]:
```


DOI:10.6814/NCCU201901175

```
N = [980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009]
TPR = []
for i in range(10):
   TPR.append(TP[i]/N[i]*100)
TPR


# In[13]:


TN = []
tn = 0
for i in range(10):
   tn = tn + TP[i]
for i in range(10):
   TN.append(tn-TP[i])


# In[14]:


FP = [0 for i in range(10)]
for i in range(10):
   for j in range(10):
      FP[i] = FP[i] + C[i][j]
for i in range(10):
   FP[i] = FP[i] - TP[i]


# In[15]:


FN = [0 for i in range(10)]
```


DOI:10.6814/NCCU201901175

```
for i in range(10):
    for j in range(10):
        FN[i] = FN[i] + C[j][i]
for i in range(10):
    FN[i] = FN[i] - TP[i]


# In[16]:


Sensitivity = [0 for i in range(10)]
Specificity = [0 for i in range(10)]
for i in range(10):
    Sensitivity[i] = TP[i]/(TP[i]+FN[i])
    Specificity[i] = TN[i]/(FP[i]+TN[i])


# In[17]:


Specificity_bar = [1 for i in range(10)]
for i in range(10):
    Specificity_bar[i] = Specificity_bar[i] - Specificity[i]


# In[18]:


a = np.mean(Sensitivity)
b = np.mean(Specificity_bar)


# In[19]:
```


DOI:10.6814/NCCU201901175

```
plt.plot(Specificity_bar, Sensitivity, 'o')
plt.plot([0,1],[0,1])
plt.plot(b,a, 'o')


# In[20]:


a


# In[21]:


b


# # Cost sensitive learning

# In[22]:


y_prob = model.predict_proba(x_test1)


# In[149]:


cm = np.array([[0,1,1,1,1,1,1,1,1,1], [1,0,1000,1,1,1,0,0,2000,1],
            [1,1,0,1,1,1,1,1,1,1], [1,1,1,0,1,1,1,1,1,1],
            [1,1,1,1,0,1,1,1,1,1], [1,1,1,1,1,0,1,1,1,1],
```


DOI:10.6814/NCCU201901175

```
            [1,1,1,1,1,1,0,1,1,1], [1,1,1,1,1,1,1,0,1,1],
            [1,1,1,1,1,1,1,1,0,1], [1,1,1,1,1,1,1,1,1,0],])


# In[150]:


y_prob_new = [[] for i in range(10000)]
for i in range(10000):
    for j in range(10):
        y_prob_new[i].append(np.sum(y_prob[i][j]*cm[j]))


# In[151]:


y_prob_softmax = [[] for i in range(10000)]
for i in range(10000):
    y_prob_softmax[i].append(y_prob_new[i]/np.sum(y_prob_new[i]))


# In[152]:


y_pred_new = []
for i in y_prob_softmax:
    y_pred_new.append(np.argmax(i))


# In[153]:


C_new = confusion_matrix(y_pred_new, y_true)
```


DOI:10.6814/NCCU201901175

```
plt.imshow(C_new)
plt.colorbar()


# In[154]:


C_new


# In[155]:


TP_new = []
for i in range(10):
    tp = 0
    for j in range(10):
        if i == j :
            tp = tp + C_new[i][j]
            TP_new.append(tp)


# In[156]:


TPR_new = []
for i in range(10):
    TPR_new.append(TP_new[i]/N[i]*100)
TPR_new
```

A.8 Mean Squared False Error Model

DOI:10.6814/NCCU201901175

```
# coding: utf-8

# # 套件

# In[14]:


get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import random
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import Dropout
from keras import backend as K
import tensorflow as tf


# # 生成不平衡資料&處理

# In[15]:


# 將訓練集分類
x0 = []
x1 = []
```


DOI:10.6814/NCCU201901175

```
y0 = []
y1 = []

for i in range(60000):
    if y_train[i] == 0:
        x0.append(x_train[i])
        y0.append(y_train[i])
    if y_train[i] == 1:
        x1.append(x_train[i])
        y1.append(y_train[i])


# In[16]:


# 訓練集分類後的長度
lx = []
ly = []
lx.append(len(x0))
lx.append(len(x1))


# In[17]:


lx #12665


# In[18]:


# 訓練集各類數量長條圖
```


DOI:10.6814/NCCU201901175

```
plt.bar(range(2),lx)


# In[19]:


# 隨機index
np.random.seed(10)
ri0 = np.random.choice(range(5923),2, replace = False)
ri1 = np.random.choice(range(6742),5000, replace = False)


# In[20]:


# 生成隨機不平衡數據
rx0 = []
rx1 = []
for i in ri0:
    rx0.append(x0[i])
for i in ri1:
    rx1.append(x1[i])
rx = rx0 + rx1
ry = y0[:2] + y1[:5000]


# In[21]:


# 不平衡資料各類樣本數
lxr = [2, 5000] #5004
```


DOI:10.6814/NCCU201901175

```
# In[22]:


# 不平衡資料各類樣本數長條圖
plt.bar(range(2),lxr)


# In[23]:


# 訓練數據index
np.random.seed(10)
rxi = np.random.choice(range(5002), 5002, replace = False)


# In[24]:


# 生成訓練數據
x_trainib = []
y_trainib = []
for j in rxi:
    x_trainib.append((rx[j]/255)) #normalization
    y_trainib.append(np_utils.to_categorical(ry[j],2))
x_train_ib = np.array(x_trainib).reshape(5002,28,28,1)
y_train_ib = np.array(y_trainib)


# # MFSE

# In[103]:
```


DOI:10.6814/NCCU201901175

```
def mfse(y_true, y_pred):

    y_true0 = y_true[:,0:1]
    y_true1 = y_true[:,1:2]

    return (y_true0 * (K.square(K.mean(K.square(y_true - y_pred))/2))
          + y_true1 * (K.square(K.mean(K.square(y_true -
             y_pred))/5000)))*5002


# # 神經網路建模&訓練

# In[12]:


model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', input_shape = (28,
    28, 1))) #32個大小為3x3的filters，出來的結果會補0變成一樣大小的矩陣
model.add(Activation('relu'))
model.add(Dropout(0.2))
    #丟棄20%的output避免overfitting
model.add(MaxPooling2D(pool_size = (2,2)))
    #2x2當中選最大的值
model.add(Conv2D(64, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3, 3), padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())                                    #矩陣拉平
model.add(Dense(200))
```


DOI:10.6814/NCCU201901175

```
model.add(Activation('relu'))
model.add(Dense(2))                                    #output
    10個結果
model.add(Activation('softmax'))

model.compile(loss = mfse, optimizer = SGD(lr = 0.001), metrics =
    ["accuracy"])


# In[13]:


model.summary()


# In[14]:


model.fit(x_train_ib, y_train_ib, batch_size = 100, epochs = 40)


# # 測試成果

# In[15]:


# 將測試集分類
xx0 = []
xx1 = []

yy0 = []
yy1 = []
```


DOI:10.6814/NCCU201901175

```
for i in range(10000):
    if y_test[i] == 0:
        xx0.append(x_test[i])
        yy0.append(y_test[i])
    if y_test[i] == 1:
        xx1.append(x_test[i])
        yy1.append(y_test[i])

rxx = xx0 + xx1
ryy = yy0 + yy1


# In[16]:


# 隨機index for testing set
np.random.seed(10)
rxxi = np.random.choice(range(2115), 2115, replace = False)


# In[17]:


# 測試集分類後的長度
lxx = []
lyy = []
lxx.append(len(xx0))
lxx.append(len(xx1))


# In[18]:
```


DOI:10.6814/NCCU201901175

```
lxx #2115
```

```
# In[19]:
```

```
plt.bar(range(2),lxx)
```

```
# In[21]:
```

```
# 生成測試數據
x_testib = []
y_testib = []
for j in rxxi:
    x_testib.append((rxx[j]/255)) #normalization
    y_testib.append(np_utils.to_categorical(ryy[j],2))
x_test_ib = np.array(x_testib).reshape(2115,28,28,1)
y_test_ib = np.array(y_testib)
```

```
# In[22]:
```

```
# 測試平衡資料
score = model.evaluate(x_test_ib, y_test_ib)
print('loss:', score[0])
print('Accuracy:', score[1])
```

```
# In[23]:
```


DOI:10.6814/NCCU201901175

```
model_json = model.to_json()
open('MNIST_MFSE2.json', 'w').write(model_json)
model.save_weights('MNIST_MFSE2_wights.h5')
```

A.9 Anomaly Detection Model

```
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('env', 'KERAS_BACKEND=tensorflow')
import numpy as np
import matplotlib.pyplot as plt


# In[2]:


from keras.models import model_from_json
from keras.optimizers import SGD


# In[3]:


model = model_from_json(open('手寫辨識CNN_model_cnn.json').read())
model.load_weights('手寫辨識CNN_model_weights_cnn.h5')
```


DOI:10.6814/NCCU201901175

```
# In[4]:


model.compile(loss = ”categorical_crossentropy”, optimizer = SGD(lr
    = 0.05), metrics = [”accuracy”])


# In[5]:


from keras.datasets import mnist
from keras.utils import np_utils
(x_train, y_train), (x_test, y_test) = mnist.load_data()


# In[6]:


x_test1 = x_test.reshape(10000, 28, 28, 1)
y_true = y_test
y_pred = model.predict_classes(x_test1)


# In[7]:


from sklearn.metrics import confusion_matrix


# In[8]:
```


DOI:10.6814/NCCU201901175

```
C=confusion_matrix(y_pred, y_true)


# In[9]:


plt.imshow(C)


# In[10]:


C


# In[11]:


TP = []
for i in range(10):
    tp = 0
    for j in range(10):
        if i == j :
            tp = tp + C[i][j]
            TP.append(tp)


# In[12]:


TP
```


DOI:10.6814/NCCU201901175

```
# In[13]:


N = [980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009]
TPR = []
for i in range(10):
   TPR.append(TP[i]/N[i]*100)
TPR


# In[14]:


TN = []
tn = 0
for i in range(10):
   tn = tn + TP[i]
for i in range(10):
   TN.append(tn-TP[i])


# In[15]:


FP = [0 for i in range(10)]
for i in range(10):
   for j in range(10):
      FP[i] = FP[i] + C[i][j]
for i in range(10):
   FP[i] = FP[i] - TP[i]


# In[16]:
```


DOI:10.6814/NCCU201901175

```
FN = [0 for i in range(10)]
for i in range(10):
   for j in range(10):
      FN[i] = FN[i] + C[j][i]
for i in range(10):
   FN[i] = FN[i] - TP[i]


# In[17]:


Sensitivity = [0 for i in range(10)]
Specificity = [0 for i in range(10)]
for i in range(10):
   Sensitivity[i] = TP[i]/(TP[i]+FN[i])
   Specificity[i] = TN[i]/(FP[i]+TN[i])


# In[18]:


Specificity_bar = [1 for i in range(10)]
for i in range(10):
   Specificity_bar[i] = Specificity_bar[i] - Specificity[i]


# In[19]:


a = np.mean(Sensitivity)
b = np.mean(Specificity_bar)
```


DOI:10.6814/NCCU201901175

```
# In[20]:


plt.plot(Specificity_bar, Sensitivity, 'o')
plt.plot([0,1],[0,1])
plt.plot(b,a, 'o')


# # Cat

# In[21]:


from PIL import Image
import os


# In[22]:


os.getcwd()


# In[23]:


cimg = np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載.jpg'))
plt.figure('下載')
plt.imshow(cimg)
```


DOI:10.6814/NCCU201901175

```
# In[24]:


cimg1 =
    np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載1.jpg'))
plt.figure('下載1')
plt.imshow(cimg1)


# In[25]:


cimg2 =
    np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載2.jpg'))
plt.figure('下載2')
plt.imshow(cimg2)


# In[26]:


Cimg = []
Cimg.append(cimg)
Cimg.append(cimg1)
Cimg.append(cimg2)


# In[27]:


from skimage import transform
```


DOI:10.6814/NCCU201901175

```
# In[28]:


Cat = []
for i in range(3):
    for j in range(3):
        cat = transform.resize(Cimg[i][:,:,j],(28,28))
        Cat.append(cat)


# In[29]:


plt.imshow(Cat[2])


# In[30]:


Cat = np.array(Cat).reshape(9,28,28,1)


# In[31]:


Cat_prob = model.predict_proba(Cat)


# In[32]:


for i in Cat_prob:
    print(np.max(i))
```


DOI:10.6814/NCCU201901175

```
# # Dog

# In[33]:


dimg = np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載3.jpg'))
plt.figure('下載3')
plt.imshow(dimg)


# In[34]:


dimg1 =
   np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載4.jpg'))
plt.figure('下載4')
plt.imshow(dimg1)


# In[35]:


dimg2 =
   np.array(Image.open('C:\\Users\\Eric\\神經網路學習\\下載5.jpg'))
plt.figure('下載5')
plt.imshow(dimg2)


# In[36]:
```


DOI:10.6814/NCCU201901175

```
Dimg = []
Dimg.append(dimg)
Dimg.append(dimg1)
Dimg.append(dimg2)


# In[37]:


Dog = []
for i in range(3):
    for j in range(3):
        dog = transform.resize(Dimg[i][:,:,j],(28,28))
        Dog.append(dog)


# In[38]:


Dog = np.array(Dog).reshape(9,28,28,1)


# In[39]:


Dog_prob = model.predict_proba(Dog)


# In[40]:


for i in Dog_prob:
    print(np.max(i))
```


DOI:10.6814/NCCU201901175

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[3] MHB Carvalho, ML Brizot, LM Lopes, CH Chiba, S Miyadahira, and M Zugaib. Detection of fetal structural abnormalities at the 11–14 week ultrasound scan. *Prenatal Diagnosis: Published in Affiliation With the International Society for Prenatal Diagnosis*, 22(1):1–4, 2002.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[6] Edward Choi, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association*, 24(2):361–370, 2016.

[7] David A Cieslak, Nitesh V Chawla, and Aaron Striegel. Combating imbalance in network intrusion datasets. In *GrC*, pages 732–737, 2006.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

DOI:10.6814/NCCU201901175

[9] MJ Desforges, PJ Jacob, and JE Cooper. Applications of probability density estimation to the detection of abnormal conditions in engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 212(8):687–703, 1998.

[10] Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11, pages 1–8. Citeseer, 2003.

[11] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

[12] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239, 2017.

[13] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, (9):1263–1284, 2008.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] JB Heaton, Nicholas G Polson, and Jan Hendrik Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

[16] David Hsu, Gildardo Sánchez-Ante, and Zheng Sun. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 3874–3880. IEEE, 2005.

[17] Anil K Jain, Jianchang Mao, and KM Mohiuddin. Artificial neural networks: A tutorial. *Computer*, (3):31–44, 1996.

[18] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

DOI:10.6814/NCCU201901175

[19] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Miroslav Kubat, Robert C Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine learning*, 30(2-3):195–215, 1998.

[22] Matjaz Kukar, Igor Kononenko, et al. Cost-sensitive learning with neural networks. In *ECAI*, pages 445–449, 1998.

[23] Yoji Kukita, Junji Uchida, Shigeyuki Oba, Kazumi Nishino, Toru Kumagai, Kazuya Taniguchi, Takako Okuyama, Fumio Imamura, and Kikuya Kato. Quantitative identification of mutant alleles derived from lung cancer in plasma cell-free dna via anomaly detection using deep sequencing data. *PloS one*, 8(11):e81468, 2013.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436, 2015.

[25] Hansang Lee, Minseok Park, and Junmo Kim. Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning. In *2016 IEEE international conference on image processing (ICIP)*, pages 3713–3717. IEEE, 2016.

[26] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[27] CX Ling and VS Sheng. Cost-sensitive learning and the class imbalance problem. 2011. *Encyclopedia of Machine Learning: Springer*, 24.

[28] Amogh Mahapatra, Nisheeth Srivastava, and Jaideep Srivastava. Contextual anomaly detection in text data. *Algorithms*, 5(4):469–489, 2012.

DOI:10.6814/NCCU201901175

[29] Bomin Mao, Zubair Md Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers*, 66(11):1946–1960, 2017.

[30] David Masko and Paulina Hensman. The impact of imbalanced training data for convolutional neural networks, 2015.

[31] P Rahmawati and Prawito Prajitno. Online vibration monitoring of a water pump machine to detect its malfunction components based on artificial neural network. In *Journal of Physics: Conference Series*, volume 1011, page 012045. IOP Publishing, 2018.

[32] R Bharat Rao, Sriram Krishnan, and Radu Stefan Niculescu. Data mining for improved cardiac care. *ACM SIGKDD Explorations Newsletter*, 8(1):3–10, 2006.

[33] Richard G Stafford, Jacob Beutel, et al. Application of neural networks as an aid in medical diagnosis and general anomaly detection, July 19 1994. US Patent 5,331,550.

[34] David WJ Stein, Scott G Beaven, Lawrence E Hoff, Edwin M Winter, Alan P Schaum, and Alan D Stocker. Anomaly detection from hyperspectral imagery. *IEEE signal processing magazine*, 19(1):58–69, 2002.

[35] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[36] Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J Kennedy. Training deep neural networks on imbalanced data sets. In *2016 international joint conference on neural networks (IJCNN)*, pages 4368–4374. IEEE, 2016.

[37] Wei Wei, Jinjiu Li, Longbing Cao, Yuming Ou, and Jiahang Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4): 449–475, 2013.

[38] Rui Yan, Yiping Song, and Hua Wu. Learning to respond with deep neural networks for retrieval-based human-computer conversation system. In *Proceedings of the 39th*

DOI:10.6814/NCCU201901175

International ACM SIGIR conference on Research and Development in Information Retrieval, pages 55–64. ACM, 2016.

[39] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated it system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1291–1300. IEEE, 2016.

[40] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge & Data Engineering*, (1):63–77, 2006.

DOI:10.6814/NCCU201901175