UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

***TITLE:*** ***Sem1 AY 22/23 – Convolutional Neural Networks (CNNs)***

***MODULE:*** ***CS4287 - NEURAL COMPUTING***

***SUBMISSION DATE:*** ***1/11/2022***

***LECTURER:*** ***J.J. Collins***

**It is hereby declared that this coursework item is entirely my/our own work, unless otherwise stated, and that all sources of information have been properly acknowledged and referenced. It is also declared by me/us that this coursework item has not previously been submitted by me/any member of the group as part fulfilment of any module assessment requirement.**

| **Name** | **Student ID** | **Signature** | **Date** |
|---|---|---|---|
| **MaoLin Wei** | **21217009** | **MaoLin Wei** | **31/10/2022** |
| **GuiYang Fan** | **21147418** | **GuiYang Fan** | **31/10/2022** |
| **YinYong Heng** | **21147469** | **YinYong Heng** | **31/10/2022** |

# Table of Contents

1. Overview ..... 1
2. The Dataset ..... 1
    2.1 Introduction ..... 1
    2.3 Visualisation of the Key Attributes ..... 3
    2.4 Pre-processing Data ..... 5
3. The Network Structure: ResNet152V2 ..... 7
    3.1 Structure and Architecture ..... 7
    3.2 Residual Learning ..... 9
        3.2.1 Residual blocks ..... 9
        3.2.2 ResNet and residual learning ..... 11
4. Hyperparameters and Rubric ..... 12
    4.1 Activation Function ..... 12
    4.2 Loss Function - Crossentropy ..... 13
    4.3 The optimiser - Adam ..... 14
    4.4 Batch Normalisation ..... 14
6. Results and Evaluation of Models ..... 25
    6.1 Confusion Matrix ..... 25
    6.2 F1-Score (Precision and Recall) ..... 26
    6.3 Kappa Coefficient ..... 26
    6.4 Hamming Distance ..... 27
    6.5 Jaccard Coefficient ..... 27
    6.6 Cross Fold Validation ..... 28
6.6.2 two-fold cross-validation ..... 28
7. Conclusion and Further Research ..... 29
8. References ..... 32

## 1. Overview

In this project, we implemented Convolutional Neural Network (CNN) based on ResNet152V2. Our work is to analyze the data set containing four different kinds of weather (Cloudy, Rain, Shine and Sunrise) and ensure a high classification accuracy by training the model. In the preliminary preparation stage of the data set and the model, we adopted methods such as modifying the image parameters and using the optimizer to ensure good training results, and the final prediction rate of the model reached a satisfactory 97%. We used the Confusion Matrix, Precision and Recall, Hamming Distance, etc., to evaluate the model, modified different parameters to reduce the impact brought by overfitting, and finally achieved significant results.

## 2. The Dataset

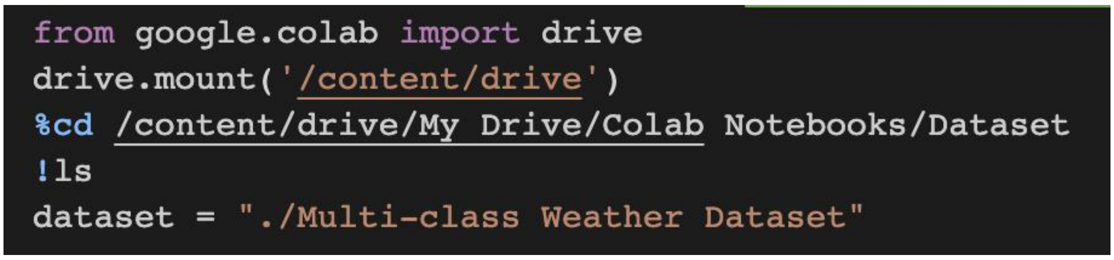
### 2.1 Introduction

We got our dataset from Kaggle and it is named Multi-Weather  Dataset (4-class images). His copyright type is defined as CC0: Public Domain, which means that the author does not retain copyright and ownership of this dataset, it is fully public.

It has 1125 images in four categories, including 300 images of “Cloudy”, 215 images of "Rain", 253 images of "Shine", and 357 images of "Sunrise". Each image is named in the form of category + number.

Our dataset is from
https://www.kaggle.com/datasets/pratik2901/multiclass-weather-dataset

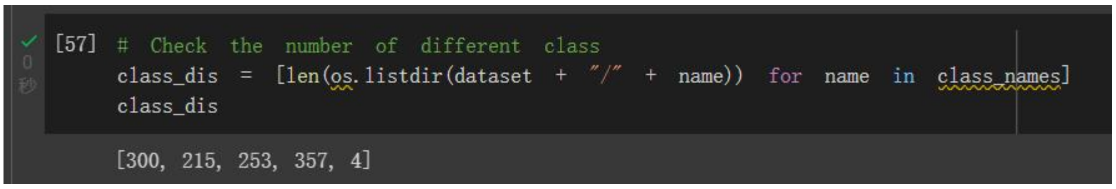
Our project runs on the colab platform, so you need to upload data sets to Google Drive (/content/drive/My Drive/Colab Notebooks/Dataset) and link it to Drive. Furthermore, use variables to save the path for subsequent operations.



```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/My Drive/Colab Notebooks/Dataset
!ls
dataset = "./Multi-class Weather Dataset"
```

*Figure 2.1: Set up the path of dataset*

Data presentation:

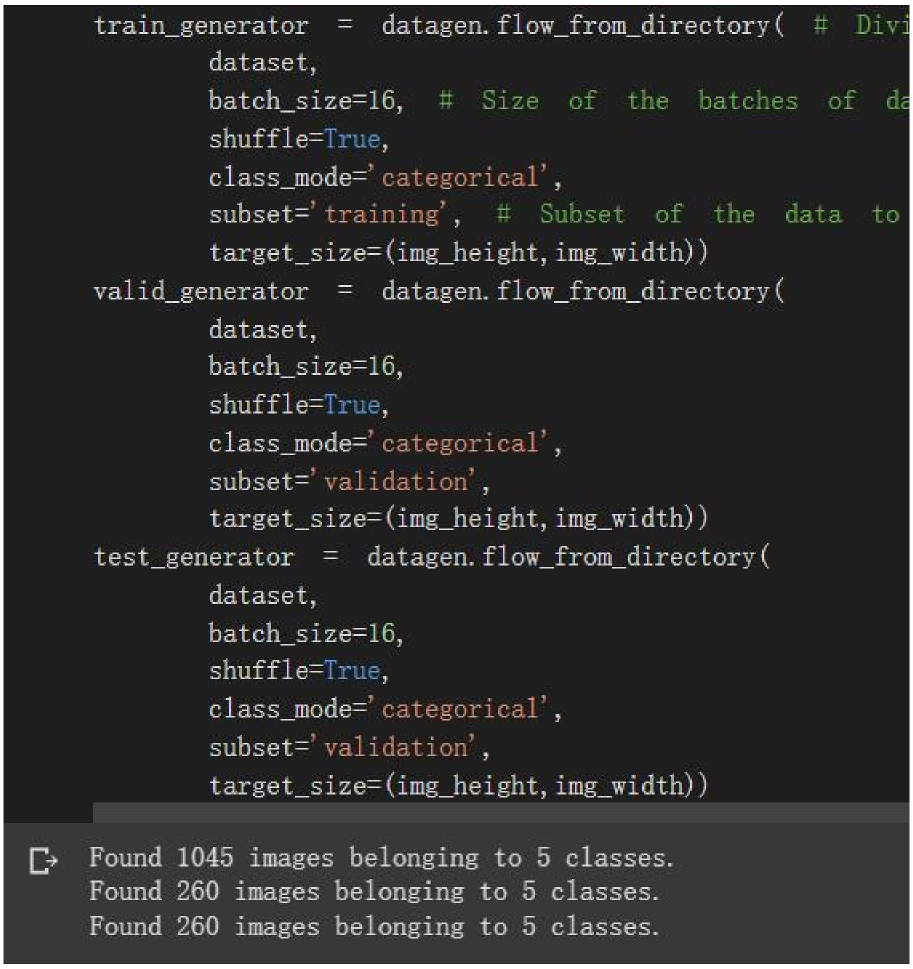


```
[57] # Check the number of different class
     class_dis = [len(os.listdir(dataset + "/" + name)) for name in class_names]
     class_dis

     [300, 215, 253, 357, 4]
```

*Figure 2.2: Number of categories and categories per category*

We divided the data set into three parts, the training set, the validation set, and the test set, for the training and evaluation of the model.



```
train_generator = datagen.flow_from_directory( # Divi
        dataset,
        batch_size=16, # Size of the batches of da
        shuffle=True,
        class_mode='categorical',
        subset='training', # Subset of the data to
        target_size=(img_height, img_width))
valid_generator = datagen.flow_from_directory(
        dataset,
        batch_size=16,
        shuffle=True,
        class_mode='categorical',
        subset='validation',
        target_size=(img_height, img_width))
test_generator = datagen.flow_from_directory(
        dataset,
        batch_size=16,
        shuffle=True,
        class_mode='categorical',
        subset='validation',
        target_size=(img_height, img_width))

Found 1045 images belonging to 5 classes.
Found 260 images belonging to 5 classes.
Found 260 images belonging to 5 classes.
```

*Figure 2.3: Divide the dataset*

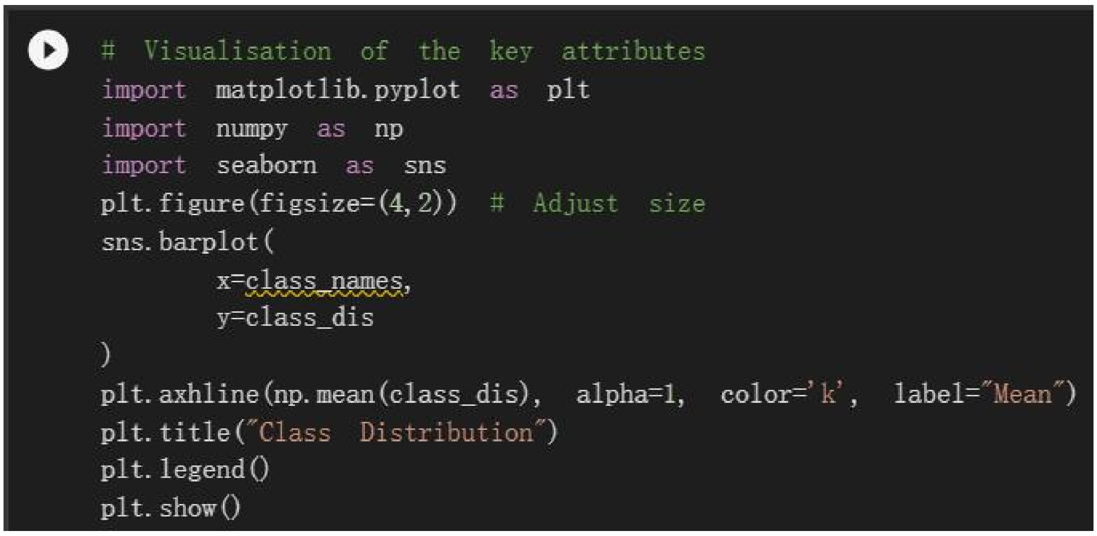
## 2.2 Data Correlation and feature engineering

Each image has 3 characteristics, the image (i.e. the image itself), the image/file name (i.e. the text related to where to find the image and its name), and the label (identifying which category of weather it belongs to). In our implementation. We move all the files from the training and test sets to a new directory with four subfolders, one for each category, and then we use the data source. We use the datagen.flow_from_directory() method to generate the training and test sets. the flow_from_directory() method allows you to read images from a directory and perform data augmentation on the model as it learns from the training data.

## 2.3 Visualisation of the Key Attributes

Since the picture contains much information, to better train and obtain a reliable model, we use image enhancement to show the critical information of the picture. Use the np. shape function to crop the image, and use reshaping to see and implement the cropped effect.

Graphical presentation of data:



```
# Visualisation of the key attributes
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
plt.figure(figsize=(4,2)) # Adjust size
sns.barplot(
        x=class_names,
        y=class_dis
)
plt.axhline(np.mean(class_dis), alpha=1, color='k', label="Mean")
plt.title("Class Distribution")
plt.legend()
plt.show()
```

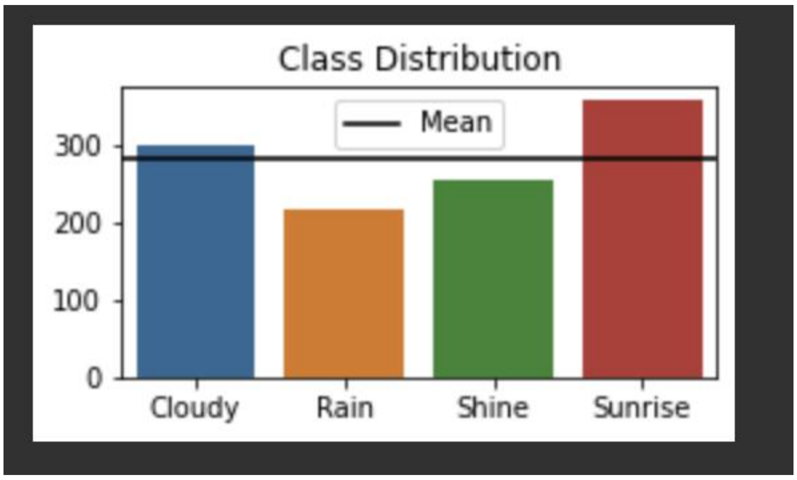
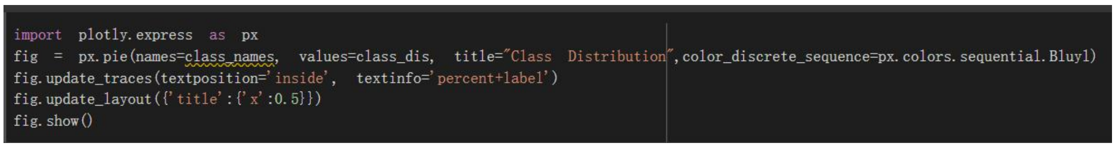


*Figure 2.4: The class distribution of dataset*



```
import plotly.express as px
fig = px.pie(names=class_names, values=class_dis, title="Class Distribution",color_discrete_sequence=px.colors.sequential.Bluyl)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.update_layout({'title':{'x':0.5}})
fig.show()
```

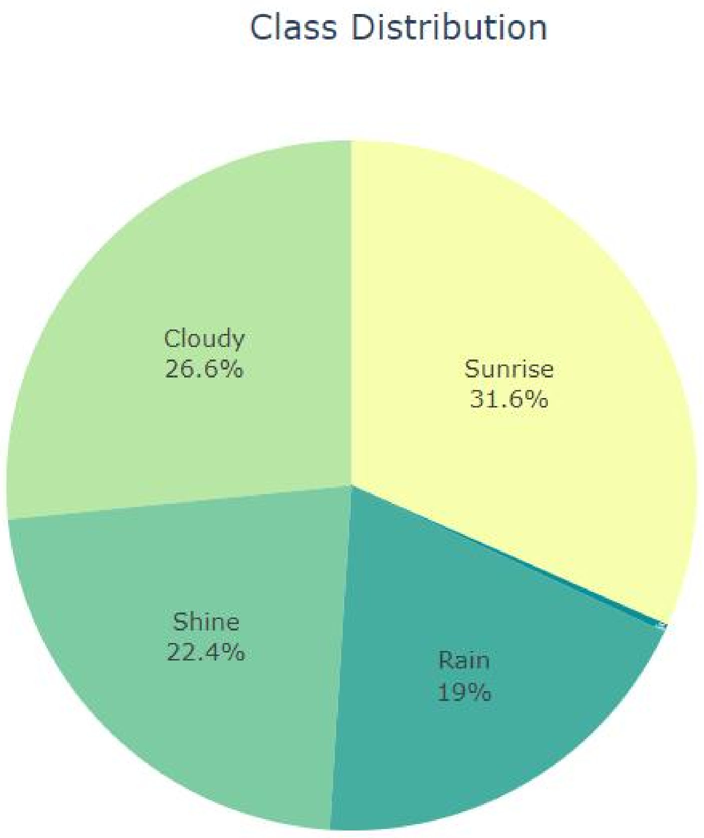
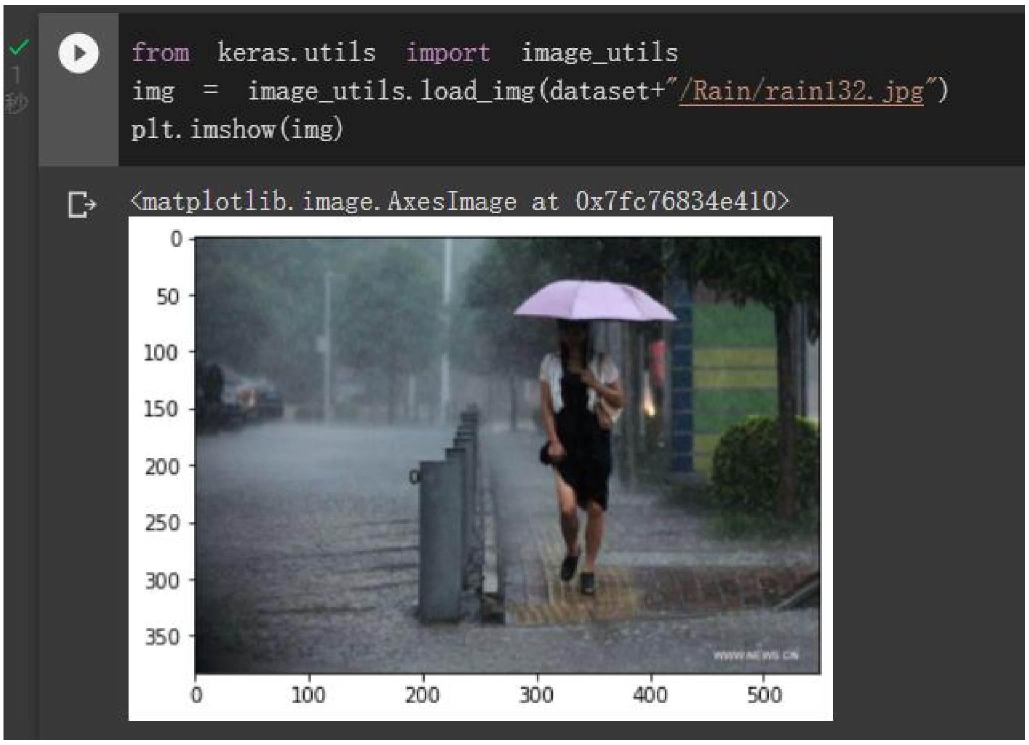


*Figure 2.5: Percentage of each category*

Dataset image display:



```
from  keras.utils  import  image_utils
img  =  image_utils.load_img(dataset+"/Rain/rain132.jpg")
plt.imshow(img)
```

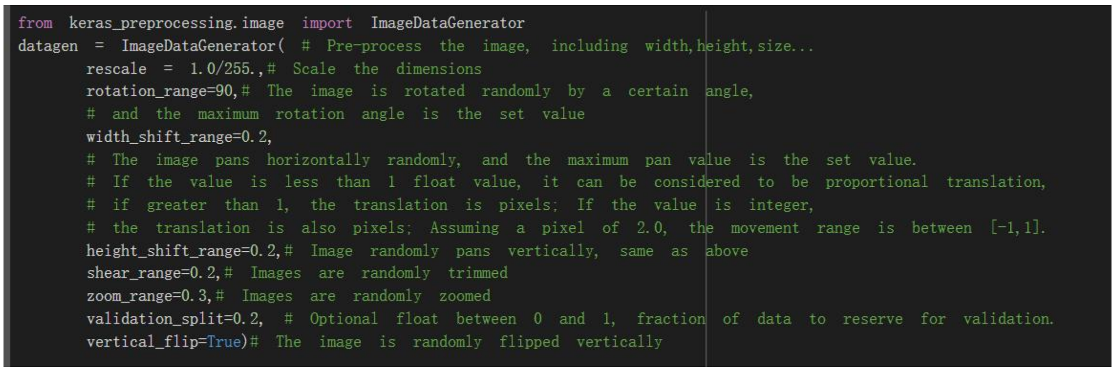
```
<matplotlib.image.AxesImage at 0x7fc76834e410>
```

*Figure 2.6: Percentage of each category*

## 2.4 Pre-processing Data

The primary purpose of image preprocessing is to eliminate irrelevant information, recover accurate, helpful information, enhance the detectability of relevant information, and simplify the data to the greatest extent to improve the reliability of feature extraction, image segmentation, matching and recognition. For preprocessing, we use the ImageDataGenerator class in keras_preprocessing.image, which defines how the image will be enhanced.

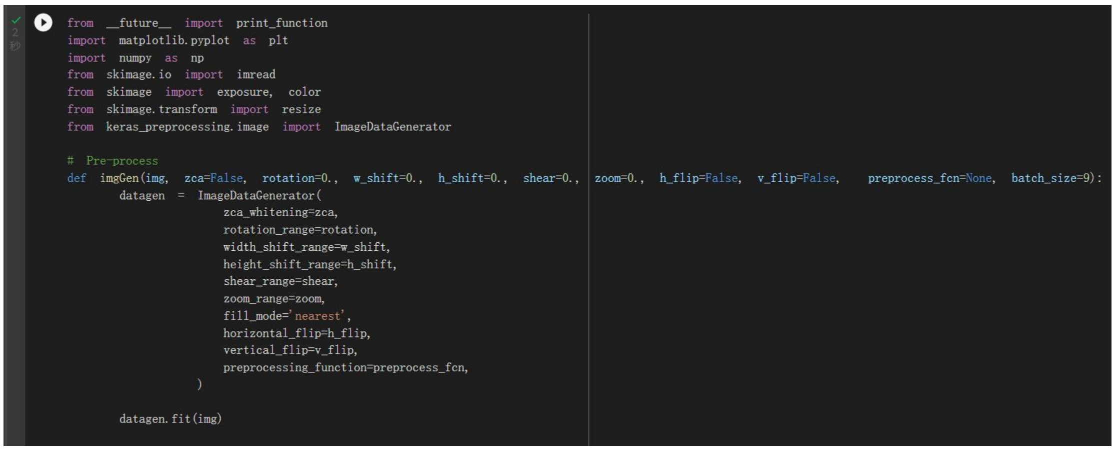
We used the rescale parameter, which will be multiplied over the entire image before performing the rest of the processing. Our images are all integers from 0-255 in the RGB channel, and such an operation could make the image too high or too low, so we set this value to a number between 01. We also use the Rotation_range parameter, which randomly rotates the image, and we set the image to rotate by 90 degrees; zoom_range parameter serves to randomly zoom in on the image; shear_range represents the shear intensity, which is the angle of shearing in the counterclockwise direction by radians, and is used to perform the degree of shear transformation; width_ shift_range is a ratio of the width of the image, representing the magnitude of the random horizontal shift of the image when the data is boosted; height_shift_range is a ratio of the height of the image, representing the magnitude of the random vertical shift of the image when the data is boosted; height_shift_range and width_shift_range are used to specify the the degree of random horizontal and vertical shifts, which are two ratios between 0 and 1. Also, the target_size parameter is used to convert the image to a target size of 90 x 90, which is suitable for our ResNet152V2-based model.



```
from keras_preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( # Pre-process the image, including width,height,size...
        rescale = 1.0/255.,# Scale the dimensions
        rotation_range=90,# The image is rotated randomly by a certain angle,
        # and the maximum rotation angle is the set value
        width_shift_range=0.2,
        # The image pans horizontally randomly, and the maximum pan value is the set value.
        # If the value is less than 1 float value, it can be considered to be proportional translation,
        # if greater than 1, the translation is pixels; If the value is integer,
        # the translation is also pixels; Assuming a pixel of 2.0, the movement range is between [-1,1].
        height_shift_range=0.2,# Image randomly pans vertically, same as above
        shear_range=0.2,# Images are randomly trimmed
        zoom_range=0.3,# Images are randomly zoomed
        validation_split=0.2, # Optional float between 0 and 1, fraction of data to reserve for validation.
        vertical_flip=True)# The image is randomly flipped vertically
```

*Figure 2.7: Code of Data enhancement*

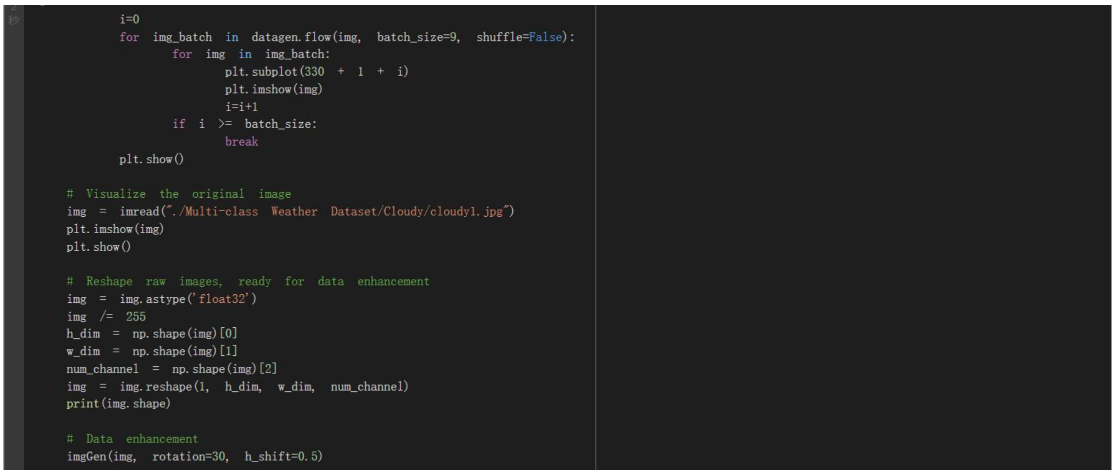
A visual example of our use of data enhancement techniques is shown in the following figure:



```
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
from skimage.io import imread
from skimage import exposure, color
from skimage.transform import resize
from keras_preprocessing.image import ImageDataGenerator

# Pre-process
def imgGen(img, zca=False, rotation=0., w_shift=0., h_shift=0., shear=0., zoom=0., h_flip=False, v_flip=False,  preprocess_fcn=None, batch_size=9):
        datagen = ImageDataGenerator(
                        zca_whitening=zca,
                        rotation_range=rotation,
                        width_shift_range=w_shift,
                        height_shift_range=h_shift,
                        shear_range=shear,
                        zoom_range=zoom,
                        fill_mode='nearest',
                        horizontal_flip=h_flip,
                        vertical_flip=v_flip,
                        preprocessing_function=preprocess_fcn,
                      )

        datagen.fit(img)
```



```
        i=0
        for img_batch in datagen.flow(img, batch_size=9, shuffle=False):
                for img in img_batch:
                        plt.subplot(330 + 1 + i)
                        plt.imshow(img)
                        i=i+1
                if i >= batch_size:
                        break
        plt.show()

# Visualize the original image
img = imread("./Multi-class Weather Dataset/Cloudy/cloudy1.jpg")
plt.imshow(img)
plt.show()

# Reshape raw images, ready for data enhancement
img = img.astype('float32')
img /= 255
h_dim = np.shape(img)[0]
w_dim = np.shape(img)[1]
num_channel = np.shape(img)[2]
img = img.reshape(1, h_dim, w_dim, num_channel)
print(img.shape)

# Data enhancement
imgGen(img, rotation=30, h_shift=0.5)
```

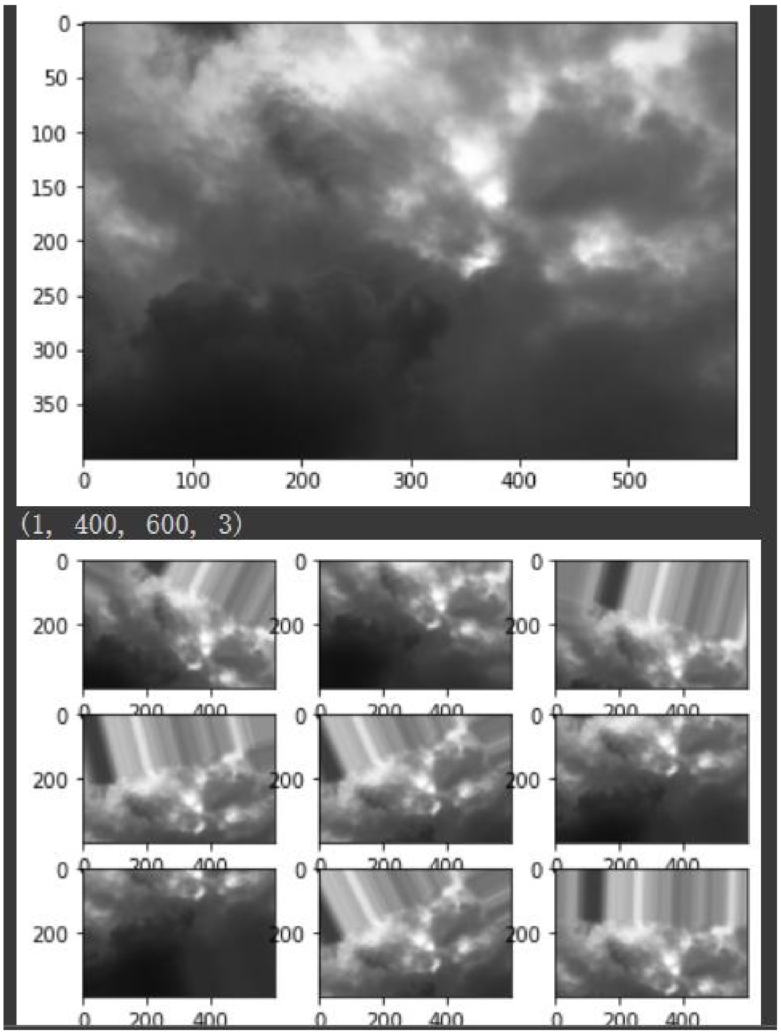
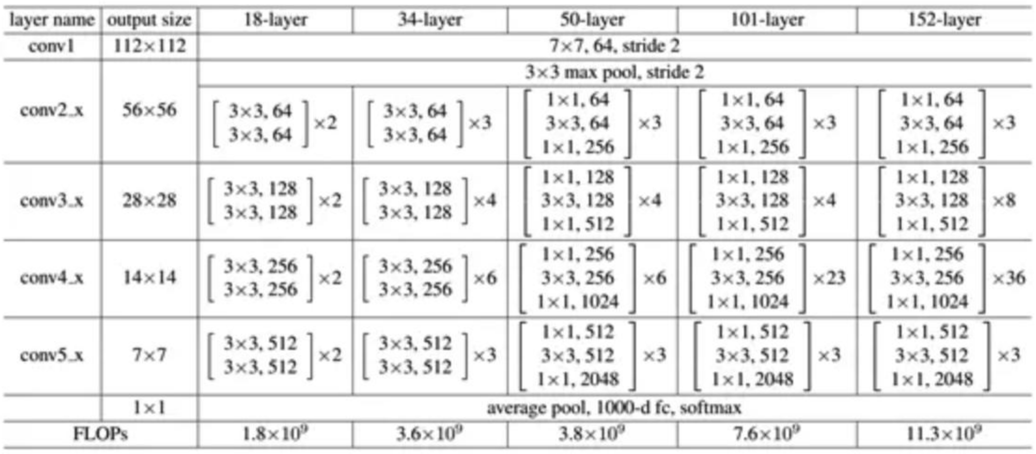

```
(1, 400, 600, 3)
```

*Figure 2.8: An example of data enhancement*

# 3. The Network Structure: ResNet152V2

## 3.1 Structure and Architecture

We used ResNet152V2 as the training data model, which was upgraded from ResNet. Dr. Keming, He invented ResNet (He *et al.* 2016).

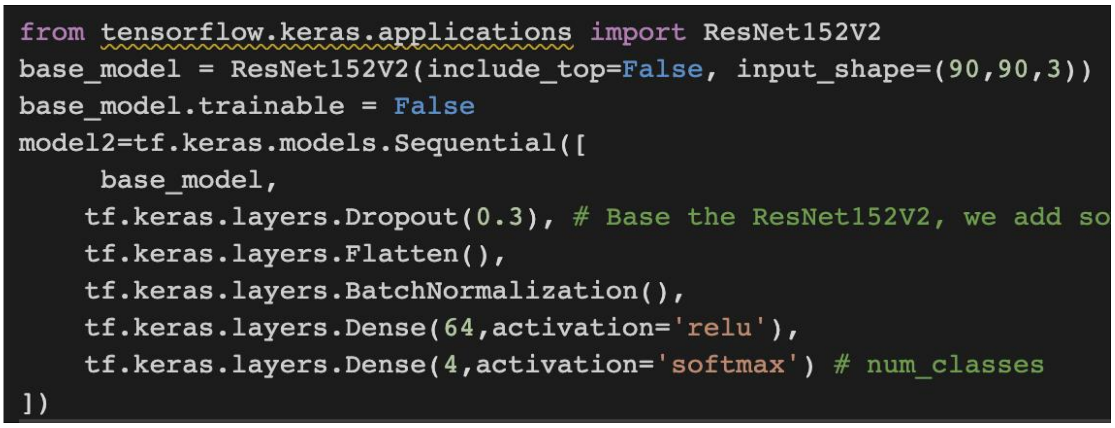


| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

***Figure 3.1: Structure of ResNet at different depths***

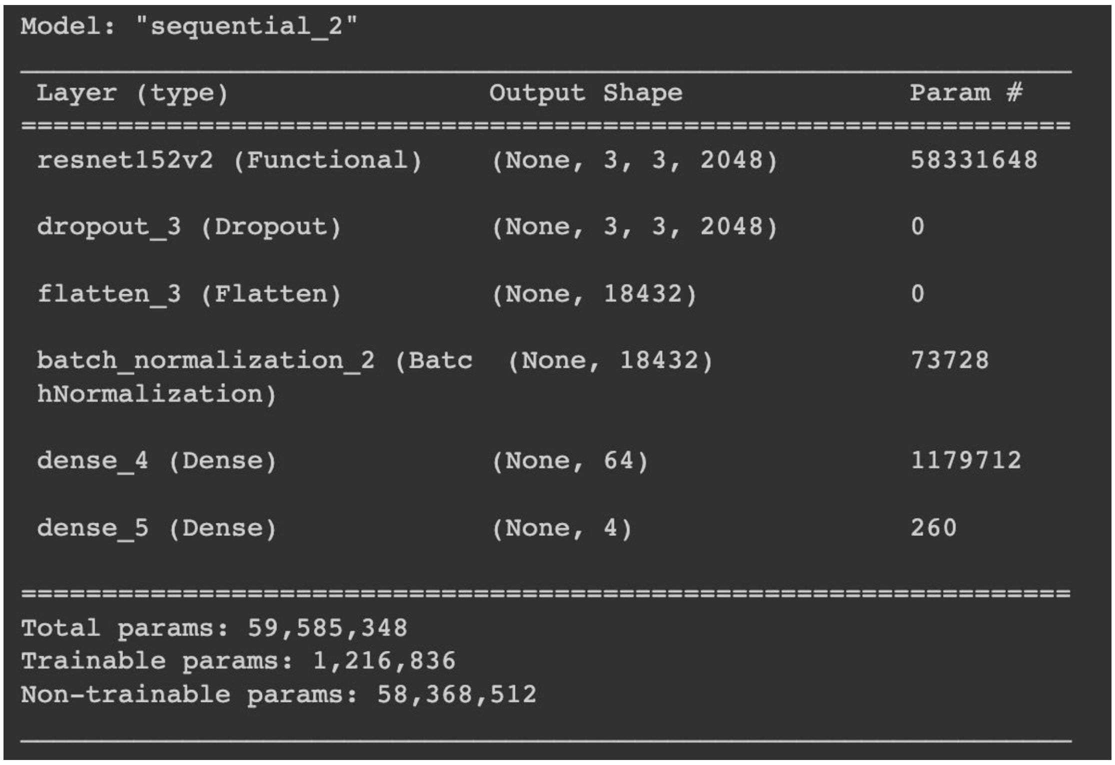
ResNet network is a reference to the VGG19 network, based on which it has been modified and added residual units through the short-circuit mechanism. The changes are mainly reflected in ResNet using convolution with stride=2 for downsampling, replacing the fully connected layer with the global average pool layer. An important design principle of ResNet is that when the feature map size is reduced by half, the number of feature maps is doubled, which maintains the complexity of the network layer. Compared with the ordinary network, ResNet increases the short-circuit mechanism between every two layers, which forms the residual learning (He *et al.* 2016).

We build the required model in the code by importing ResNet152V2 as the base model, adding the dropout layer, the flatten layer, and the activation function. Dropout can be more effective in mitigating the occurrence of overfitting and to some extent regularisation. It acts as an averaging agent and is useful in reducing the complex relationships between neurons. Flatten layers are used to "flatten" the input, i.e. to make a multidimensional input one-dimensional, and are often used in the transition from convolutional to fully-connected layers; Flatten does not affect the size of the batch.



```
from tensorflow.keras.applications import ResNet152V2
base_model = ResNet152V2(include_top=False, input_shape=(90,90,3))
base_model.trainable = False
model2=tf.keras.models.Sequential([
     base_model,
    tf.keras.layers.Dropout(0.3), # Base the ResNet152V2, we add so
    tf.keras.layers.Flatten(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(4,activation='softmax') # num_classes
])
```

*Figure 3.2: Code of Model*


```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet152v2 (Functional)    (None, 3, 3, 2048)        58331648

 dropout_3 (Dropout)         (None, 3, 3, 2048)        0

 flatten_3 (Flatten)         (None, 18432)             0

 batch_normalization_2 (Batc  (None, 18432)            73728
 hNormalization)

 dense_4 (Dense)             (None, 64)                1179712

 dense_5 (Dense)             (None, 4)                 260

=================================================================
Total params: 59,585,348
Trainable params: 1,216,836
Non-trainable params: 58,368,512
_________________________________________________________________
```

*Figure 3.3: Structure of Model with details*

## 3.2 Residual Learning

### *3.2.1 Residual blocks*

A residual network is made up of a series of residual blocks. A residual block can be expressed as:

$$x_{l+1} = x_l + \mathcal{F}(x_l, W_l)$$

The residual block is divided into two parts: the direct mapping part and the residual part. $h(x_l)$ is a direct mapping, the response to which in Figure is the curve on the left. $\mathcal{F}(x_l, W_l)$ is the residual part, which generally consists of two or three convolution operations, i.e. the part of the diagram containing the convolution on the right-hand side.

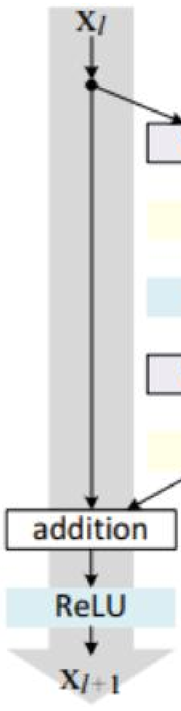


***Figure 3.4: Residual blocks***

Weight in the figure refers to the convolution operation in a convolutional network, and addition refers to the unit-addition operation.

In a convolutional network, the number of Feature Maps for $x_l$ and $x_{l+1}$ are not the same, this is where the convolution is used to raise or lower the dimension. At this point, the residual block is represented as:

$$x_{l+1} = h(x_l) + \mathcal{F}(x_l, W_l)$$

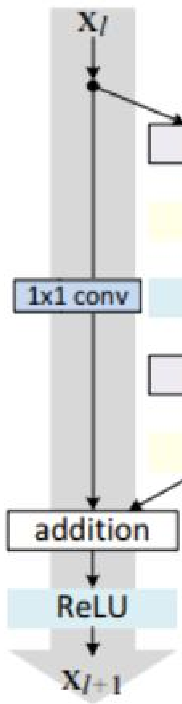


***Figure 3.5: 1*1 Residual blocks***

### *3.2.2 ResNet and residual learning*

ResNet had an excellent effect on model training because it used Residual Learning (He *et al*. 2016) in the convolutional neural network. For example, an ordinary VGG model consists of 19 layers, while ResNet already has 152 layers. From experience, the depth of the network is crucial to the model's performance. When the number of network layers increases, the network can extract more complex feature patterns, so better results can be obtained theoretically when the model is deeper. However, it was found in the experiment that there was a degradation problem in the deep network: when the network depth increased, the network accuracy was saturated or even decreased (Monti *et al.* 2018). The problem of gradient vanishing or explosion exists in the deep network, which makes the deep learning model challenging to train. Further processing was generally carried out through Batch Normalization then (Bjorck *et al.* 2018).

Therefore, Dr. He proposed Residual Learning to solve the above problems. For a stacking structure, when the input is x, the feature it learns is denoted as H(x). Now we hope that it can learn the residual $F(x)=H(x)-x$ so that the original learning feature is $F(x)+x$. This is because learning residuals is more accessible than learning primitive features directly. When the residual is 0, the accumulation layer only does the identity mapping. At least the network performance will not deteriorate. The residual will not be 0, enabling the accumulation layer to learn new features based on the input features to perform better. The structure of residual learning is shown in the figure below. This is like a "short circuit" in a circuit, so it is a shortcut connection.

The residual network still allows the non-linear layer to satisfy $H(x, w_h)$ and then introduces a short connection directly from the input to the output of the non-linear layer, making the whole mapping change into:

$$y = H(x, w_h) + x$$

This is the core formula of the residual network.

A specific residual module is defined in the following diagram:

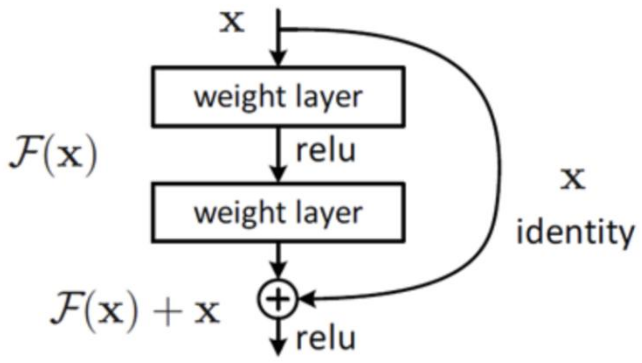


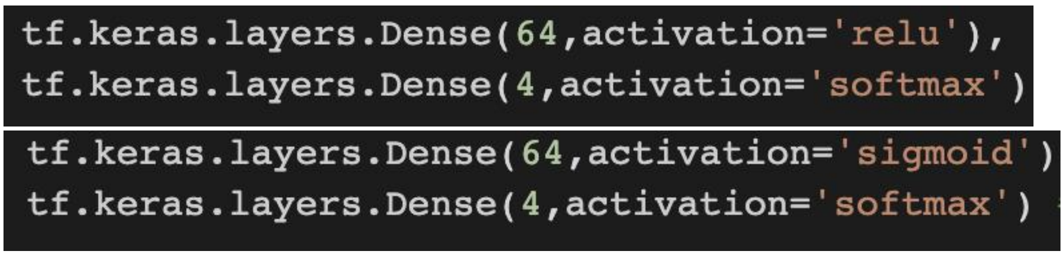
***Figure 3.6: Residuals learning unit***

## 4. Hyperparameters and Rubric

### 4.1 Activation Function

Activation functions are mainly about introducing nonlinear features into our network (Sharma *et al.* 2017). The goal is to convert an input signal from a node in the model into an output signal. This output signal is now used as the input for the next layer in the stack. Therefore, if no activation function is used, our output at each layer is just a linear transformation of the input function at the previous layer. No matter how many layers are in the neural network, the output is a linear combination of inputs. If used, the activation function introduces a nonlinear element to the neuron, allowing the neural network to approximate any nonlinear function so that the neural network can be applied to a nonlinear model.

Three different activation functions (ReLu, softmax, and sigmoid) are used in our model, among which sigmoid as an influencing parameter is explored in the fifth part.



```
tf.keras.layers.Dense(64,activation='relu'),
tf.keras.layers.Dense(4,activation='softmax')
```

```
tf.keras.layers.Dense(64,activation='sigmoid')
tf.keras.layers.Dense(4,activation='softmax')
```

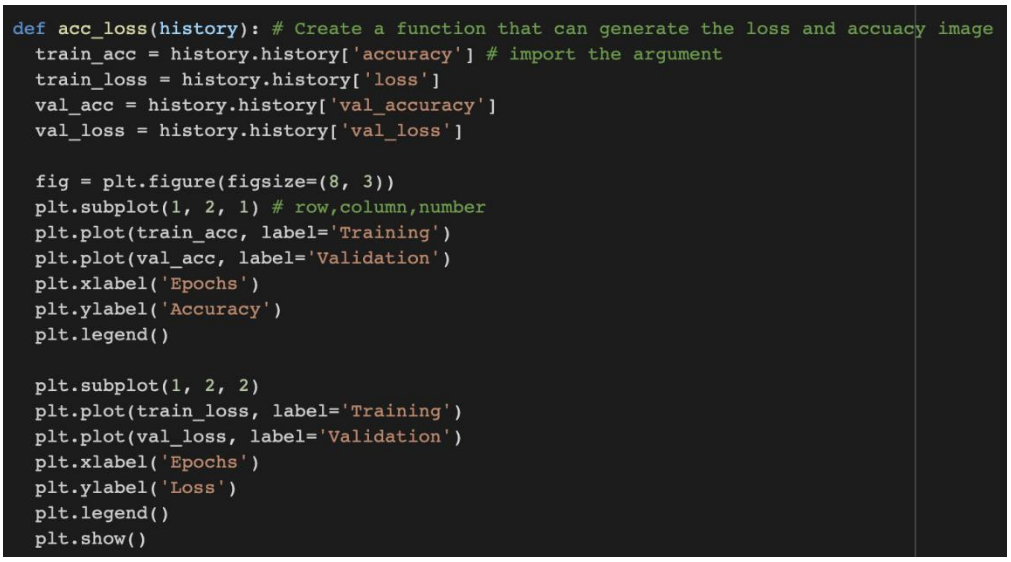
***Figure 4.1: Use different activation function***

## 4.2 Loss Function - Crossentropy

The purpose of the loss function is to express the degree of difference between logit and label. Different loss functions have different expression meanings. In minimizing the loss function, the logit approaches the label differently so that the results may differ (Xu *et al.* 2015).

Since we are a classification problem, we use the cross-entropy loss function in the model. For labels with multiple categories, the One-hot operation is to transform the specific Label space into a probability measure space (set as p), such as [1,0,0] (indicating that it is the first category). This probability can be interpreted as if the scalar output of the label classification is 1 (i.e., the probability is 100%) and the other values are 0 (i.e., the probability is 0%). After processing by the Softmax function, the actual output value of the neural network is a probability vector, such as [0.96, 0.04, 0], whose probability distribution is set as q. Now we want to measure the difference (loss) between p and q, and a better way is to measure the difference in the probability distribution between the two, so we need to measure the difference between the two probability distributions using cross-entropy (Xu *et al.* 2015).

Because we need to use the function many times, we write it in advance so it can be called directly later, improving code reuse.

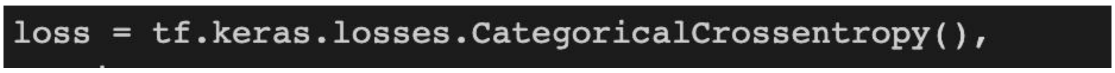


```
def acc_loss(history): # Create a function that can generate the loss and accuacy image
  train_acc = history.history['accuracy'] # import the argument
  train_loss = history.history['loss']
  val_acc = history.history['val_accuracy']
  val_loss = history.history['val_loss']

  fig = plt.figure(figsize=(8, 3))
  plt.subplot(1, 2, 1) # row,column,number
  plt.plot(train_acc, label='Training')
  plt.plot(val_acc, label='Validation')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()

  plt.subplot(1, 2, 2)
  plt.plot(train_loss, label='Training')
  plt.plot(val_loss, label='Validation')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()
```

***Figure 4.2: Create loss function plot***



```
loss = tf.keras.losses.CategoricalCrossentropy(),
```

*Figure 4.3: Use cross-entropy function*

## 4.3 The optimiser - Adam

In the model, we need to use the optimizer, the actual value minus the predicted value, taking the absolute value and summing. In the training process, it can be roughly understood as: adjusting w (weight) and b (deviation) to make the loss, as mentioned earlier, as small as possible to achieve better results.

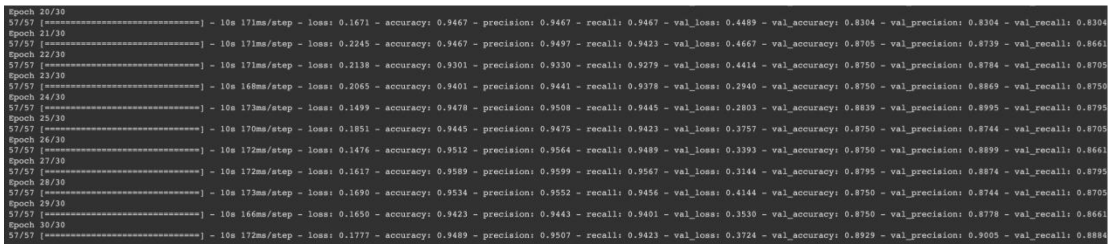
We chose to use the Adam optimizer here because the results of the Adam optimizer are generally superior to all other optimization algorithms, with faster computation times and fewer tuning parameters required. The comparison is also made in the fifth part (Zhang 2018).

## 4.4 Batch Normalisation

As mentioned before, when the depth of the neural network increases, the problem of gradient disappearance will occur. Using Batch Normalisation(BN) can accelerate the convergence speed. At the same time, to improve the model's generalisation ability, the Batch Normalisation scale factor can effectively identify neurons that do not contribute much to the network and automatically weaken or eliminate some neurons after activation function. In addition, due to normalization, the problem of significant parameter changes caused by different data distributions rarely occurs. Due to the current controversy over the placement of Batch Normalisation (Ioffe and Szegedy 2015), we have adopted the common practice of placing Batch Normalisation before the activation function.

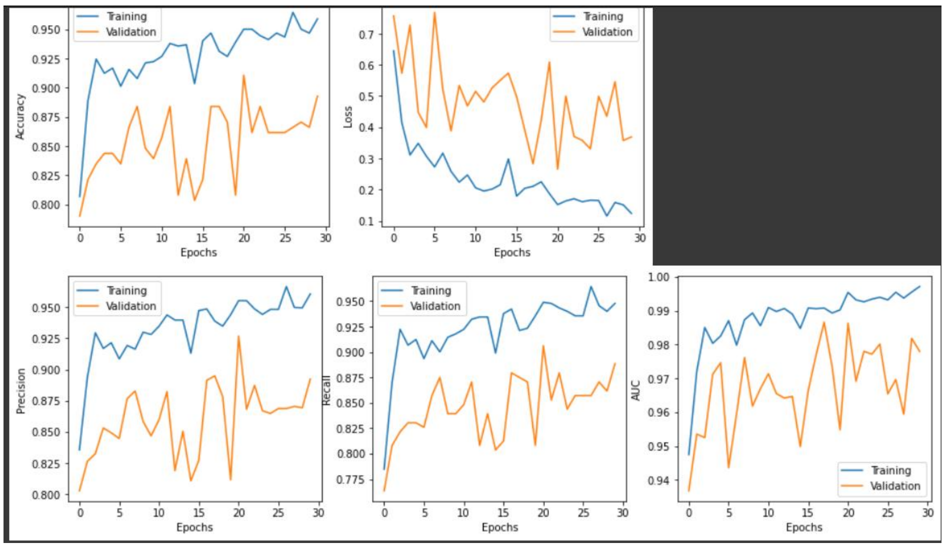
# 5. Impact of varying hyperparameters in ResNet

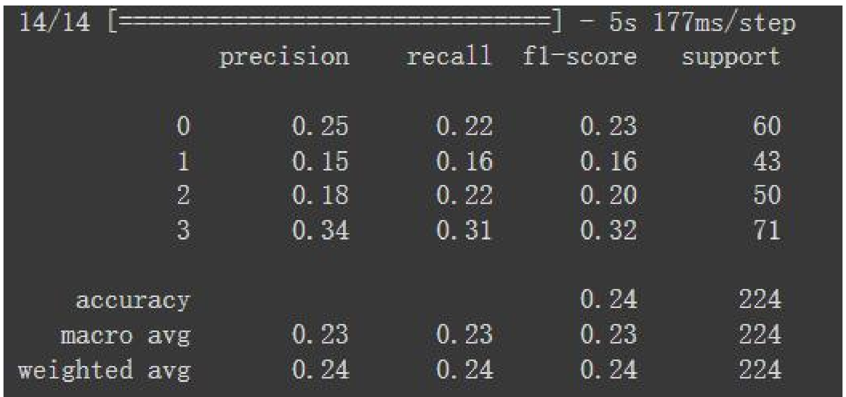
After data acquisition, data preprocessing, modelling and training, we obtained excellent results. However, after the evaluation of the model, it was found that although the training set performed well, there was a 10% difference between the training set and the verification set. Therefore, we designed five experiments to compare the model's performance under different parameter conditions. Only change the corresponding parameter at a time; the rest of the same.



```
Epoch 20/30
57/57 [==============================] - 10s 171ms/step - loss: 0.1671 - accuracy: 0.9467 - precision: 0.9467 - recall: 0.9467 - val_loss: 0.4489 - val_accuracy: 0.8304 - val_precision: 0.8304 - val_recall: 0.8304
Epoch 21/30
57/57 [==============================] - 10s 171ms/step - loss: 0.2245 - accuracy: 0.9467 - precision: 0.9497 - recall: 0.9423 - val_loss: 0.4667 - val_accuracy: 0.8705 - val_precision: 0.8739 - val_recall: 0.8661
Epoch 22/30
57/57 [==============================] - 10s 171ms/step - loss: 0.2138 - accuracy: 0.9301 - precision: 0.9330 - recall: 0.9279 - val_loss: 0.4414 - val_accuracy: 0.8750 - val_precision: 0.8784 - val_recall: 0.8705
Epoch 23/30
57/57 [==============================] - 10s 168ms/step - loss: 0.2065 - accuracy: 0.9401 - precision: 0.9441 - recall: 0.9378 - val_loss: 0.2940 - val_accuracy: 0.8750 - val_precision: 0.8869 - val_recall: 0.8750
Epoch 24/30
57/57 [==============================] - 10s 173ms/step - loss: 0.1499 - accuracy: 0.9478 - precision: 0.9508 - recall: 0.9445 - val_loss: 0.2803 - val_accuracy: 0.8839 - val_precision: 0.8995 - val_recall: 0.8795
Epoch 25/30
57/57 [==============================] - 10s 170ms/step - loss: 0.1851 - accuracy: 0.9445 - precision: 0.9475 - recall: 0.9423 - val_loss: 0.3757 - val_accuracy: 0.8750 - val_precision: 0.8744 - val_recall: 0.8705
Epoch 26/30
57/57 [==============================] - 10s 172ms/step - loss: 0.1476 - accuracy: 0.9512 - precision: 0.9564 - recall: 0.9489 - val_loss: 0.3393 - val_accuracy: 0.8750 - val_precision: 0.8899 - val_recall: 0.8661
Epoch 27/30
57/57 [==============================] - 10s 172ms/step - loss: 0.1617 - accuracy: 0.9589 - precision: 0.9599 - recall: 0.9567 - val_loss: 0.3144 - val_accuracy: 0.8795 - val_precision: 0.8874 - val_recall: 0.8795
Epoch 28/30
57/57 [==============================] - 10s 173ms/step - loss: 0.1690 - accuracy: 0.9534 - precision: 0.9552 - recall: 0.9456 - val_loss: 0.4144 - val_accuracy: 0.8750 - val_precision: 0.8744 - val_recall: 0.8705
Epoch 29/30
57/57 [==============================] - 10s 166ms/step - loss: 0.1650 - accuracy: 0.9423 - precision: 0.9443 - recall: 0.9401 - val_loss: 0.3530 - val_accuracy: 0.8750 - val_precision: 0.8778 - val_recall: 0.8661
Epoch 30/30
57/57 [==============================] - 10s 172ms/step - loss: 0.1777 - accuracy: 0.9489 - precision: 0.9507 - recall: 0.9423 - val_loss: 0.3724 - val_accuracy: 0.8929 - val_precision: 0.9005 - val_recall: 0.8884
```

*Figure 5.1: The model training on the training and validation sets*





```
14/14 [==============================] - 5s 177ms/step
              precision    recall  f1-score   support

           0       0.25      0.22      0.23        60
           1       0.15      0.16      0.16        43
           2       0.18      0.22      0.20        50
           3       0.34      0.31      0.32        71

    accuracy                           0.24       224
   macro avg       0.23      0.23      0.23       224
weighted avg       0.24      0.24      0.24       224
```

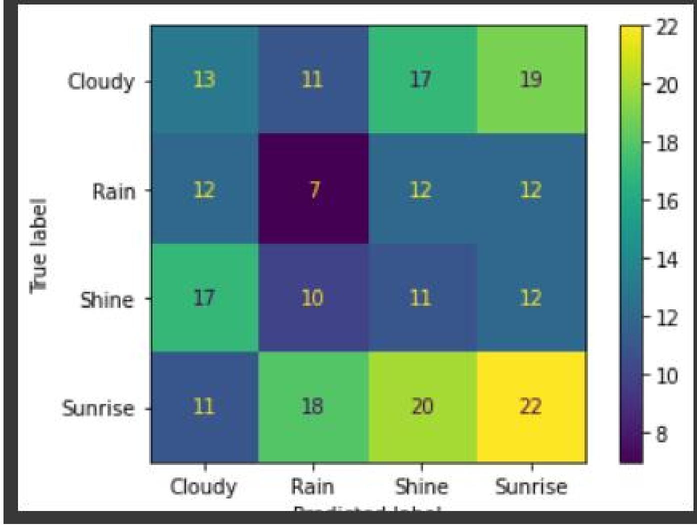
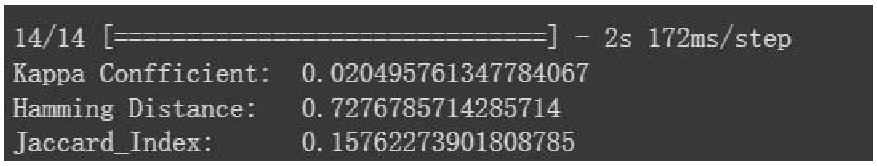


***Figure 5.2: The loss, accuracy, precision, recall, AUC plot, confusion matrix and f1-score (precision and recall) with original model***



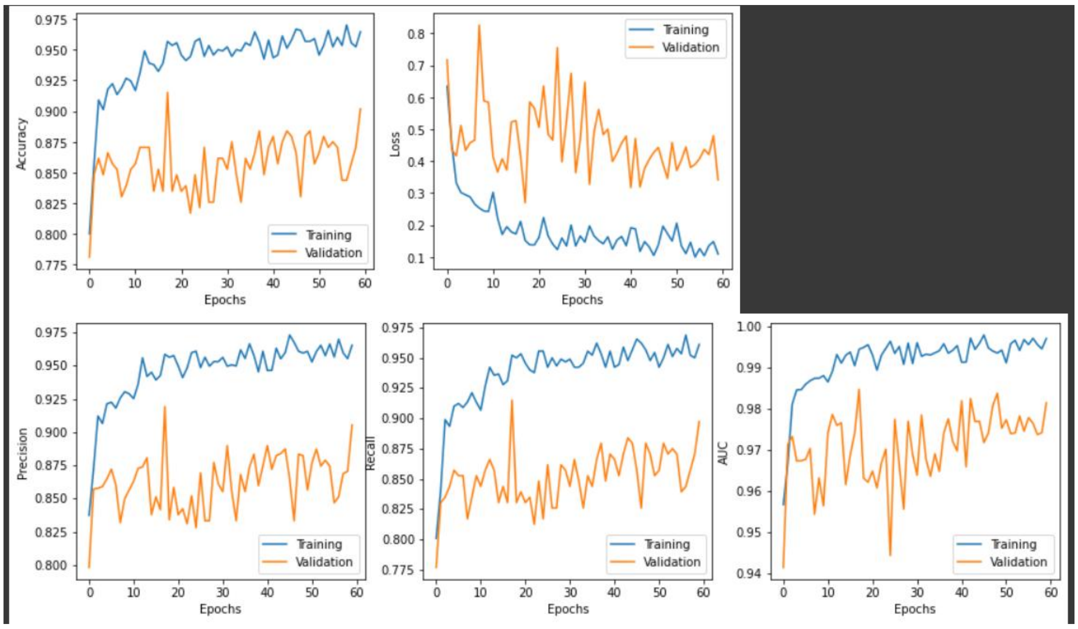
```
14/14 [==============================] - 2s 172ms/step
Kappa Confficient:   0.020495761347784067
Hamming Distance:    0.7276785714285714
Jaccard_Index:       0.15762273901808785
```

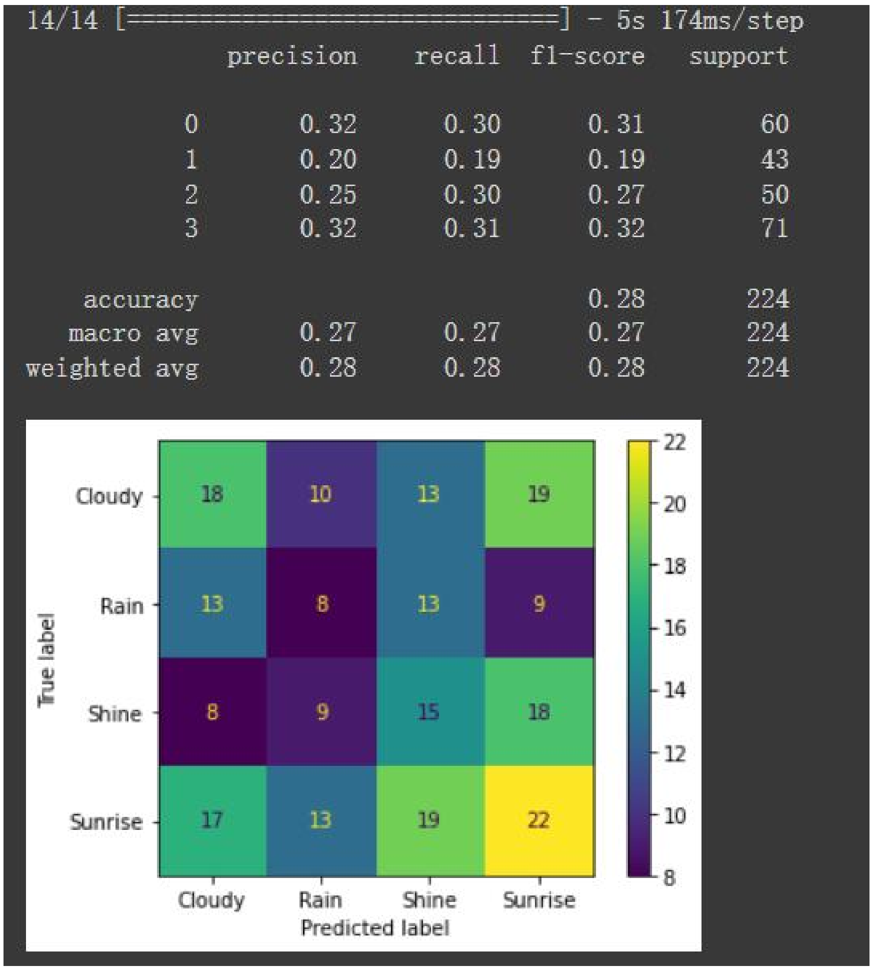
*Figure 5.3: The evaluation result of model*

## 5.1 Experiment 1: Turning the number of epochs during training

First, we changed the epochs number from 30 to 60. In the model, the process is called an epoch when a complete data set passes through the neural network once and returns once.

**The training results are as follows:**



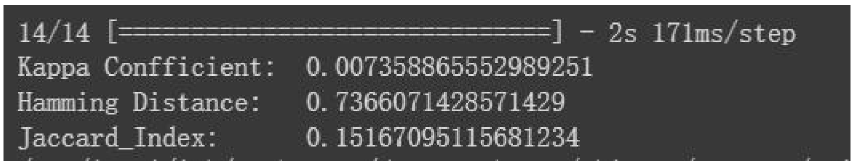


```
14/14 [==============================] - 5s 174ms/step
              precision    recall  f1-score   support

           0       0.32      0.30      0.31        60
           1       0.20      0.19      0.19        43
           2       0.25      0.30      0.27        50
           3       0.32      0.31      0.32        71

    accuracy                           0.28       224
   macro avg       0.27      0.27      0.27       224
weighted avg       0.28      0.28      0.28       224
```

*Figure 5.4: The result of experiment I model*



```
14/14 [==============================] - 2s 171ms/step
Kappa Confficient:  0.007358865552989251
Hamming Distance:   0.7366071428571429
Jaccard_Index:      0.15167095115681234
```

*Figure 5.5: The evaluation result of experiment I model*

By comparison, it can be found that increasing epochs times only slightly improves the overall change, which is not outstanding.

## 5.2 Experiment 2: Turning the flatten layers in ResNet

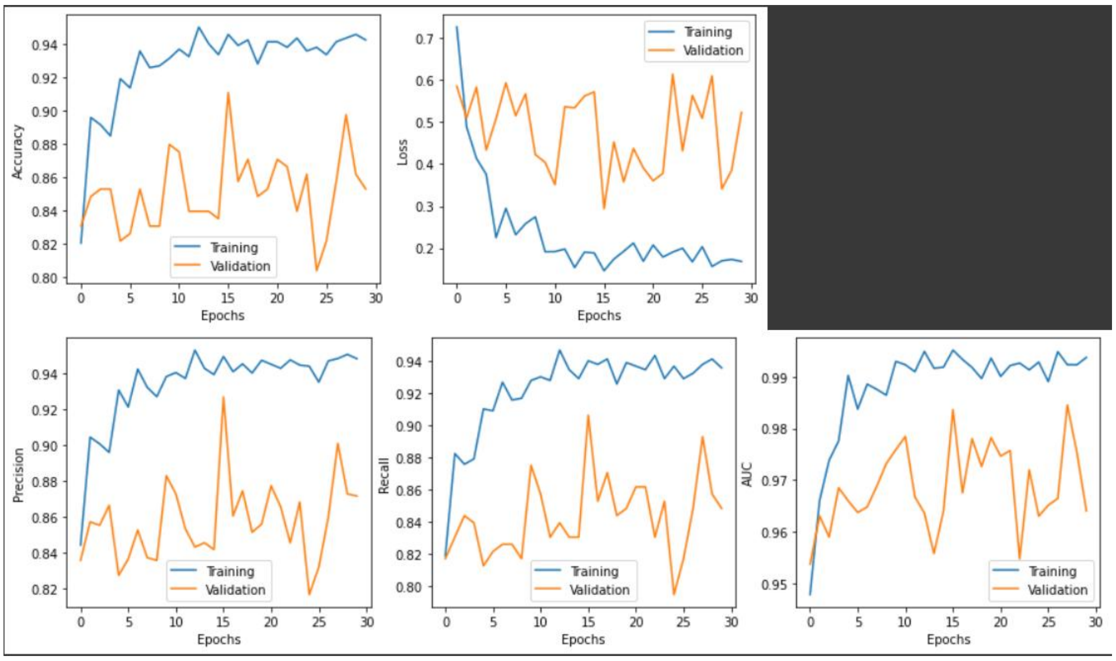
In the neural network model structure, there are many layers representing different roles and meanings (Kattenborn *et al.* 2021).

**Convolutional Layer:** The convolutional layer extracts the features of the image, breaks through the limitations of the traditional filter, extracts the desired features according to the objective function, and reduces the network parameters, ensuring the sparsity of the network.

**Pooling Layer:** The pooling layer is used to compress the input feature map. On the one hand, the feature map becomes smaller, and the network computing complexity is simplified. On the one hand, feature compression is carried out to extract the main features.
**Full Connected Layer:** Connect all features and send output values to the classifier (for example, softmax).
**Dense Layer:** dimensional data is mapped to another, while two-dimensional data cannot be directly processed.
**Flatten Layer:** Flattens the input and one-dimensional the multi-dimensional input. This is often used from the convolution layer to the full connected layer.

Therefore, we want to see if we can increase the number of flatten layers to compress the data further.

## The training results are as follows:



```
14/14 [==============================] - 5s 179ms/step
              precision    recall  f1-score   support

           0       0.29      0.20      0.24        60
           1       0.19      0.23      0.21        43
           2       0.24      0.32      0.28        50
           3       0.34      0.31      0.33        71

    accuracy                           0.27       224
   macro avg       0.27      0.27      0.26       224
weighted avg       0.28      0.27      0.27       224
```

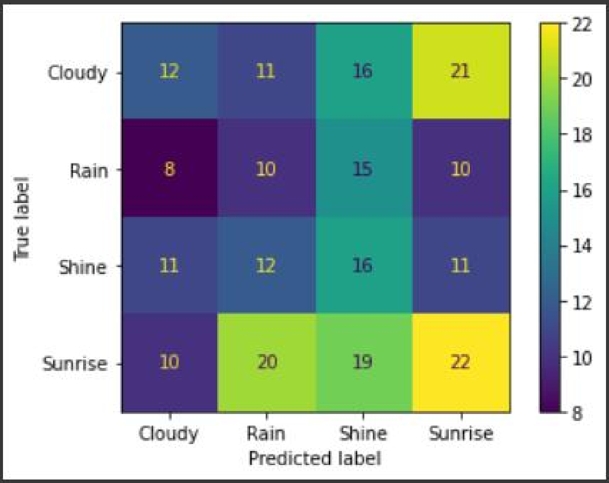


*Figure 5.6: The result of experiment II model*

```
14/14 [==============================] - 2s 180ms/step
Kappa Confficient:   0.01059017614815607
Hamming Distance:    0.7410714285714286
Jaccard_Index:       0.14871794871794872
```

*Figure 5.7: The evaluation result of experiment II model*

Through comparison, it can be found that the addition of flatten layer does not improve performance. On the contrary, it may make the performance of the model worse.

## 5.3 Experiment 3: Turning the activation function of the dense layer

We chose to modify the activation function, replacing "ReLu" with "sigmoid".

**Sigmoid:** The output of the Sigmoid function ranges from 0 to 1. Since the output values are limited to 0 to 1, it normalizes the output of each neuron. Furthermore, its gradient is smooth, the output value avoids jumping, and the function is differentiable. This means that we can find the slope of the sigmoid curve for any two points.
There are also disadvantages. Sigmoid tends to disappear the gradient, and the function output is not centred on 0, which will reduce the efficiency of weight update (Mercioni and Holban 2020).

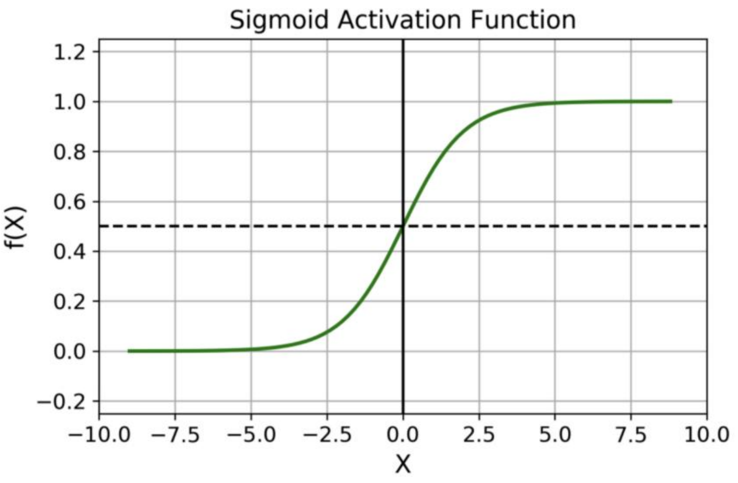


*Figure 5.8: The figure of Sigmoid activation function*

**ReLu:** Compared to Sigmoid, there is no gradient saturation problem when the input is positive. Because there is only a linear relationship in the ReLU function, the calculation is much faster.
There are also drawbacks, namely the Dead ReLU problem (Xu *et al.* 2020). ReLU fails when the input is negative, which is not a problem during forward propagation. Some areas are sensitive, and some are not. Nevertheless, in the backpropagation process, the gradient will be precisely zero if the input is negative.

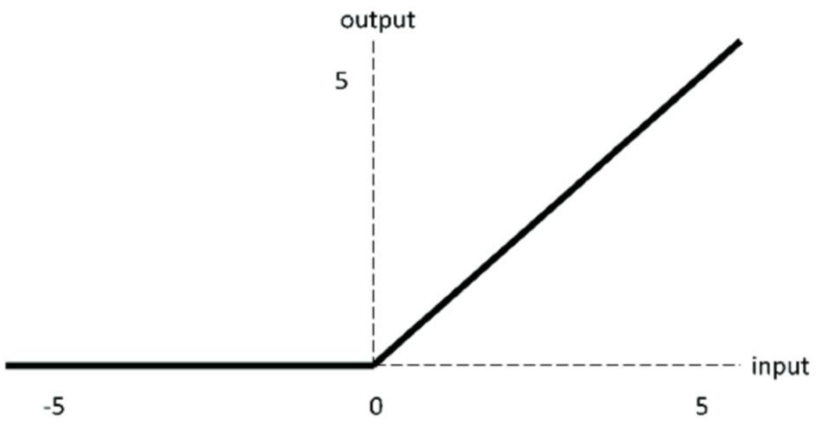
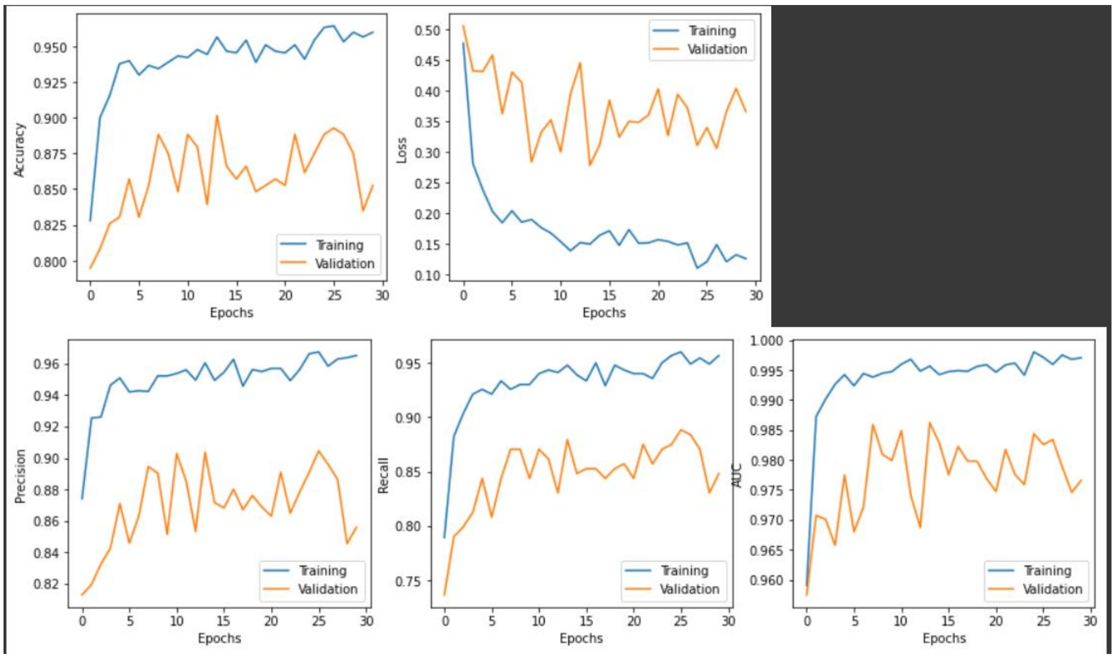
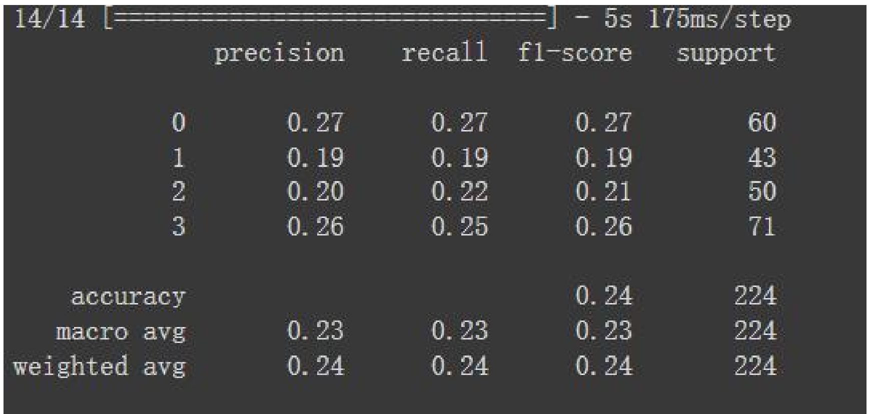


*Figure 5.9: The figure of ReLu activation function*

### The training results are as follows:





```
14/14 [==============================] - 5s 175ms/step
              precision    recall  f1-score   support

           0       0.27      0.27      0.27        60
           1       0.19      0.19      0.19        43
           2       0.20      0.22      0.21        50
           3       0.26      0.25      0.26        71

    accuracy                           0.24       224
   macro avg       0.23      0.23      0.23       224
weighted avg       0.24      0.24      0.24       224
```

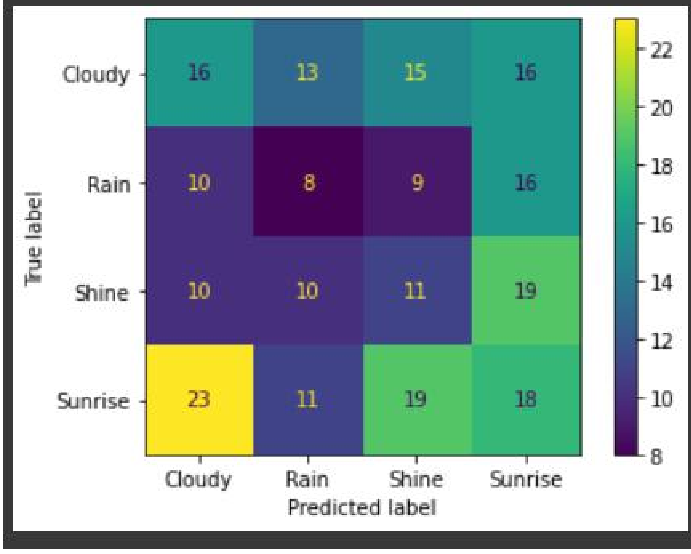
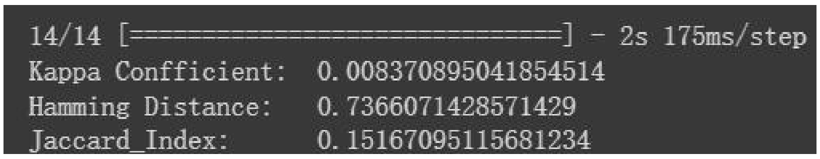


*Figure 5.10: The result of experiment III model*



```
14/14 [==============================] - 2s 175ms/step
Kappa Confficient:  0.008370895041854514
Hamming Distance:   0.7366071428571429
Jaccard_Index:      0.15167095115681234
```

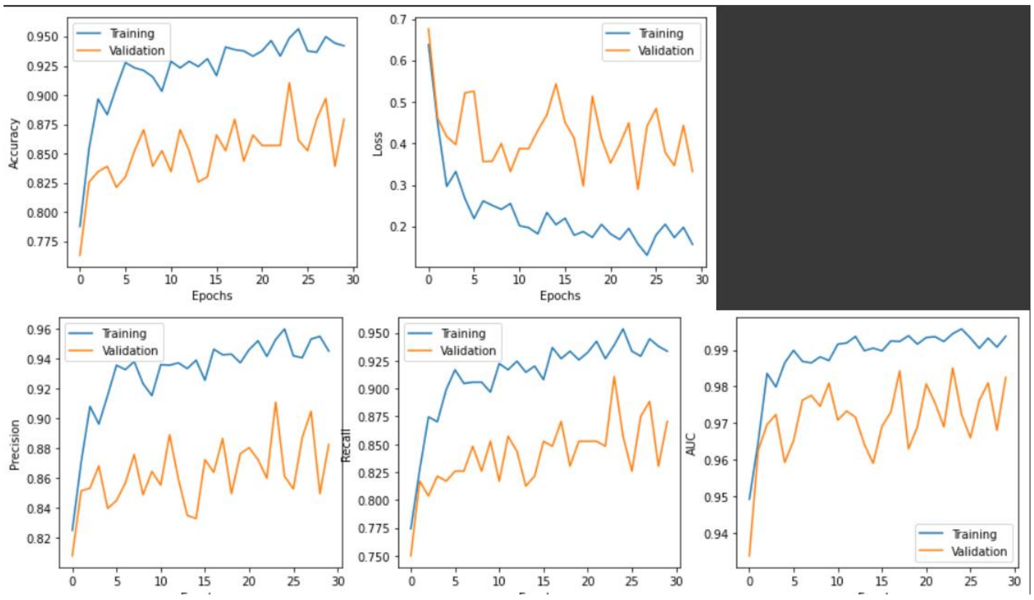
*Figure 5.11: The evaluation result of experiment III model*

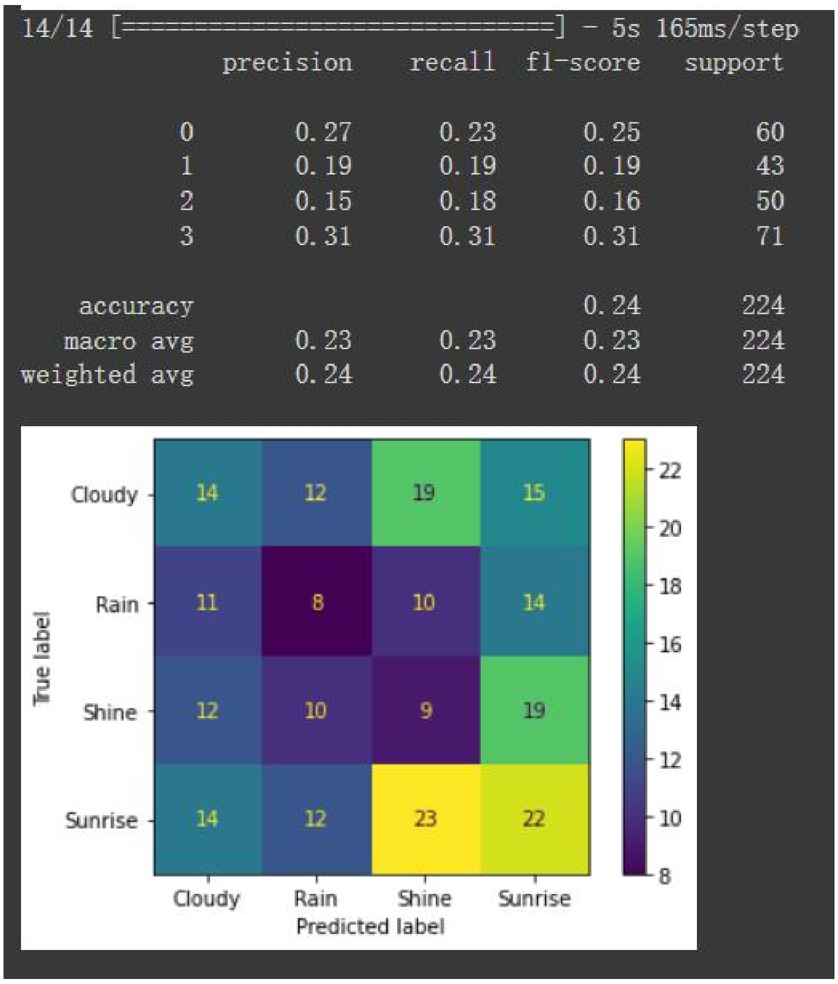
By comparison, we can see that the performance is slightly worse after the activation function is replaced. We guess that the sigmoid function is usually used for binary classification and will be poor in the case of multi-classification.

## 5.4 Experiment 4: Turning the optimiser of model

We chose to replace the optimizer with "sgd" instead of "adam".

**The training results are as follows:**




```
14/14 [==============================] - 5s 165ms/step
              precision    recall  f1-score   support

           0       0.27      0.23      0.25        60
           1       0.19      0.19      0.19        43
           2       0.15      0.18      0.16        50
           3       0.31      0.31      0.31        71

    accuracy                           0.24       224
   macro avg       0.23      0.23      0.23       224
weighted avg       0.24      0.24      0.24       224
```

| True label \ Predicted label | Cloudy | Rain | Shine | Sunrise |
|---|---|---|---|---|
| Cloudy | 14 | 12 | 19 | 15 |
| Rain | 11 | 8 | 10 | 14 |
| Shine | 12 | 10 | 9 | 19 |
| Sunrise | 14 | 12 | 23 | 22 |

***Figure 5.12: The result of experiment IV model***

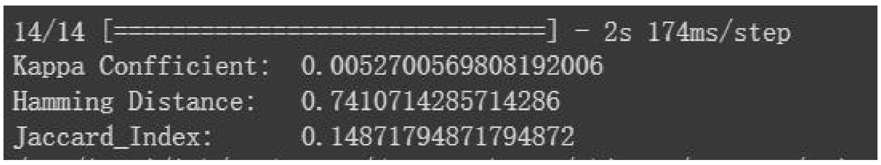

```
14/14 [==============================] - 2s 174ms/step
Kappa Confficient:  0.005270056980819206
Hamming Distance:   0.7410714285714286
Jaccard_Index:      0.14871794871794872
```

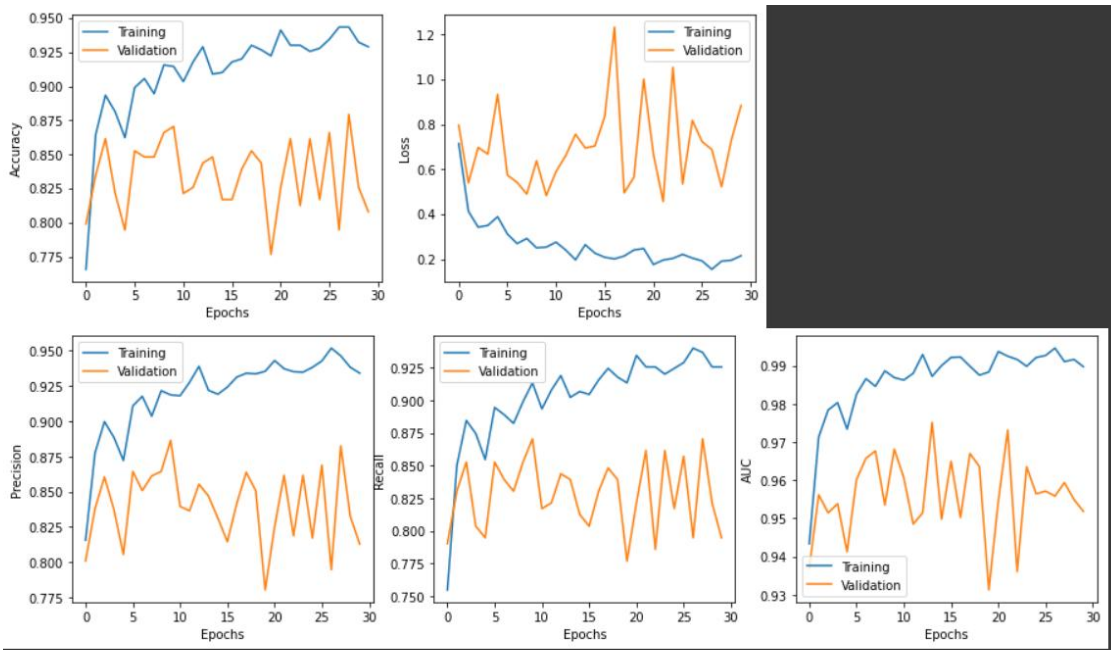
***Figure 5.13: The evaluation result of experiment IV model***

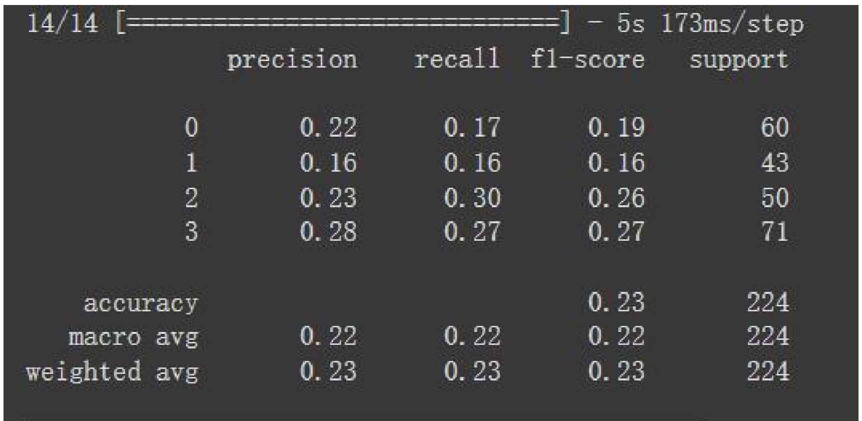
We found that after the replacement of the optimizer, the performance decreased significantly. We guessed that the optimizer "adam" was the optimizer with the excellent overall performance at present, while "sgd" had severe cost function fluctuations due to frequent updates and eventually stayed at the local minima or saddle point.

## 5.5 Experiment 5: Turning the Batch Normalisation layer of model

Through the previous analysis, we decided to add a Batch Normalisation layer to explore the impact.

**The training results are as follows:**





```
14/14 [==============================] - 5s 173ms/step
              precision    recall  f1-score   support

           0       0.22      0.17      0.19        60
           1       0.16      0.16      0.16        43
           2       0.23      0.30      0.26        50
           3       0.28      0.27      0.27        71

    accuracy                           0.23       224
   macro avg       0.22      0.22      0.22       224
weighted avg       0.23      0.23      0.23       224
```

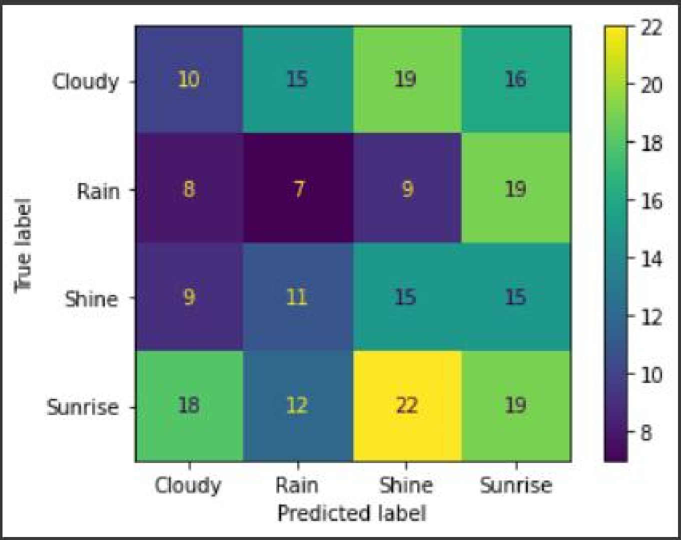
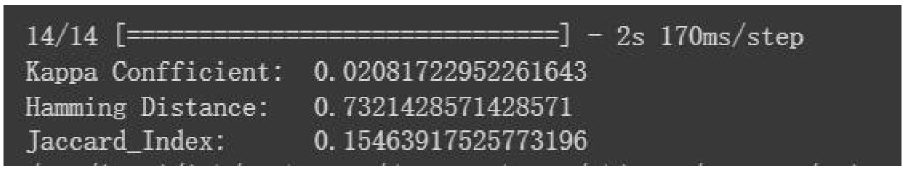


*Figure 5.14: The result of experiment V model*



```
14/14 [==============================] - 2s 170ms/step
Kappa Confficient:  0.02081722952261643
Hamming Distance:   0.7321428571428571
Jaccard_Index:      0.15463917525773196
```

*Figure 5.15: The evaluation result of experiment V model*

Through comparison, we found that although adding a Batch Normalisation layer would reduce the accuracy slightly, the curve tended to be flat and improved compared with the desired data.

# 6. Results and Evaluation of Models

After completing the above content, we obtained a model with 97% prediction success based on the existing data. In order to further confirm whether the model has good generalization ability and can anticipate practical problems, we adopted five evaluation methods to ensure the model's performance further.

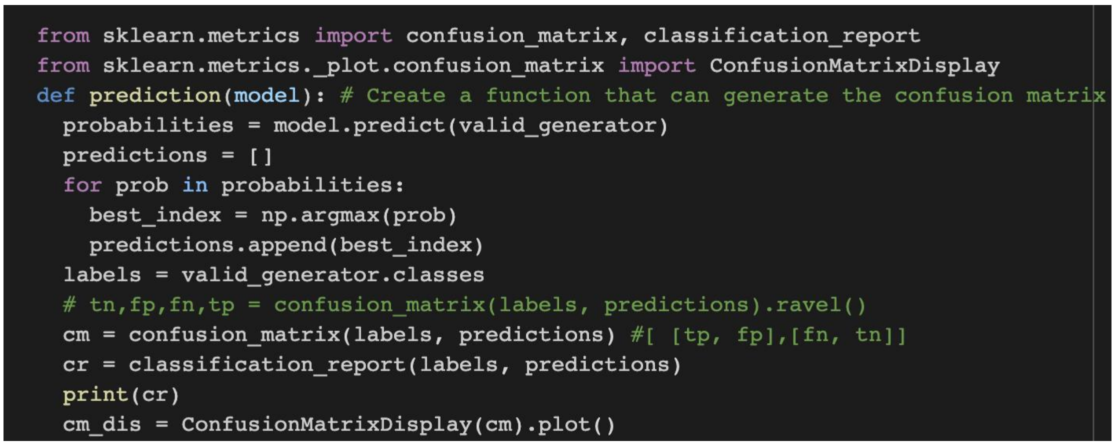
## 6.1 Confusion Matrix

Confusion Matrix is a standard evaluation method, essentially a table with relationships between actual and predicted values. There are four fundamental indicators (Visa *et al.* 2011).

The True value is positive, and the amount considered positive by the model (True Positive=TP)
The true value is positive, and the number that the model considers negative (False Negative=FN): this is a Type I Error in statistics (Kim 2015).
The true value is negative, and the number considered positive by the model (False Positive=FP) is the statistical Type II Error.
The True value is negative, the amount that the model thinks is negative (True Negative=TN)

In our predictive classification model, the expected accuracy rate is excellent. Therefore, in the corresponding confusion matrix, the number of expected TP and TN is large, while the number of expected FP and FN is small. So when we get the confusion matrix of the model, we need to see how many observations are in the corresponding positions in the second and fourth quadrants. The more values here, the better; Conversely, the fewer observations that occur in the first, third and fourth quadrants, the better.

In addition to the primary indicators mentioned above, secondary indicators, such as specificity and sensitivity, are calculated using SpecificFY. The three indexes, recall, precision and F1-score, were derived on this basis.



```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
def prediction(model): # Create a function that can generate the confusion matrix
  probabilities = model.predict(valid_generator)
  predictions = []
  for prob in probabilities:
    best_index = np.argmax(prob)
    predictions.append(best_index)
  labels = valid_generator.classes
  # tn,fp,fn,tp = confusion_matrix(labels, predictions).ravel()
  cm = confusion_matrix(labels, predictions) #[ [tp, fp],[fn, tn]]
  cr = classification_report(labels, predictions)
  print(cr)
  cm_dis = ConfusionMatrixDisplay(cm).plot()
```

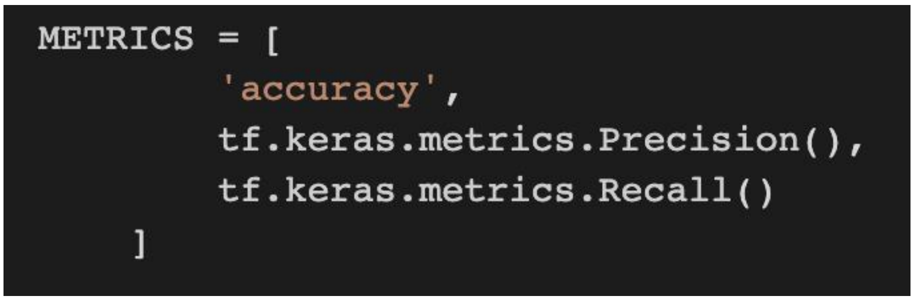
*Figure 6.1: Creat confusion matrix function and draw the plot*

## 6.2 F1-Score (Precision and Recall)

F1-score is a weighted average of Precision and Recall rates (Yacouby and Axman 2020). Precision reflects the model's ability to distinguish negative samples. The higher the Precision, the stronger the model's ability to distinguish negative samples. Recall reflects the model's ability to identify positive samples. The higher the Recall, the stronger the model's ability to identify positive samples. F1-score is a combination of the two. The higher F1-score is, the more robust the model is.

$$F_\beta = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} = \frac{(1+\beta^2) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

*Figure 6.2: The formula of F1-score*

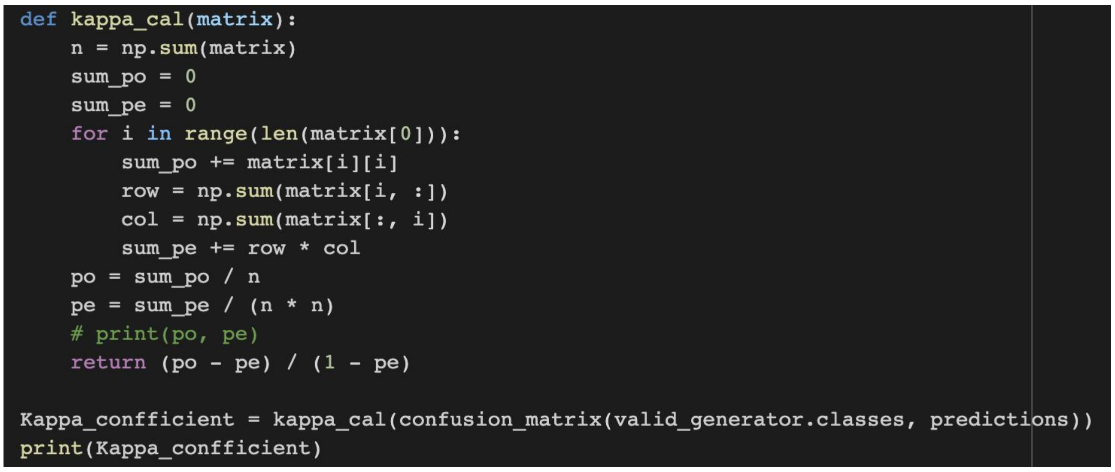


```
METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(),
        tf.keras.metrics.Recall()
    ]
```

*Figure 6.3: Use Precision and Recall function*

## 6.3 Kappa Coefficient

The Kappa coefficient is also realized based on the confusion matrix (Kraemer 2014), which is used to measure the consistency of the classification effect, that is, whether the predicted results of the model are consistent with the actual classification results. The Kappa coefficient is between -1 and 1, usually greater than 0. The larger the value, the more correct the classification result is.



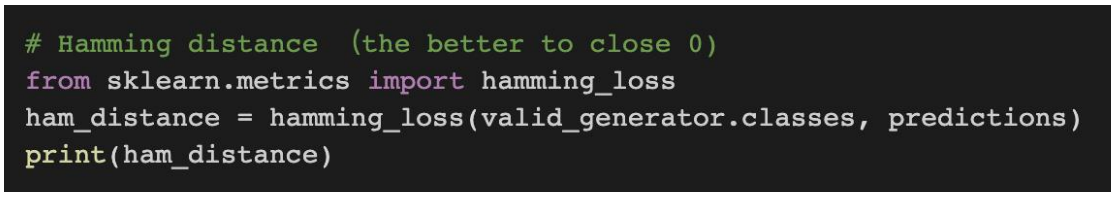
```python
def kappa_cal(matrix):
    n = np.sum(matrix)
    sum_po = 0
    sum_pe = 0
    for i in range(len(matrix[0])):
        sum_po += matrix[i][i]
        row = np.sum(matrix[i, :])
        col = np.sum(matrix[:, i])
        sum_pe += row * col
    po = sum_po / n
    pe = sum_pe / (n * n)
    # print(po, pe)
    return (po - pe) / (1 - pe)

Kappa_confficient = kappa_cal(confusion_matrix(valid_generator.classes, predictions))
print(Kappa_confficient)
```

*Figure 6.4: Create the Kappa coefficient function to use*

## 6.4 Hamming Distance

Hamming Distance is used to measure the distance between the predicted and accurate labels (Norouzi and Salakhutdinov 2012). The value ranges from 0 to 1. A distance of 0 means that the predicted result is precisely the same as the accurate result, and a distance of 1 means that the model is opposite to the result we want.

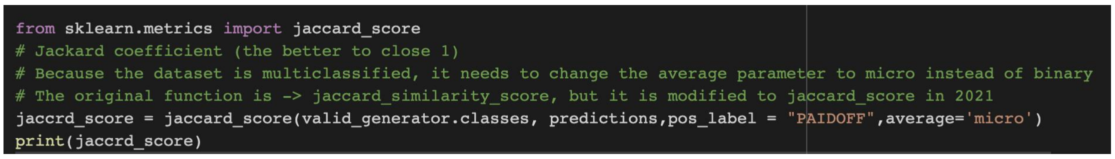


```python
# Hamming distance  (the better to close 0)
from sklearn.metrics import hamming_loss
ham_distance = hamming_loss(valid_generator.classes, predictions)
print(ham_distance)
```

*Figure 6.5: Create the hamming distance function to use*

## 6.5 Jaccard Coefficient

The difference between the Jaccard Coefficient and the Haiming distance lies in the denominator (Niwattanakul 2013). When the predicted results are consistent with the actual situation, the coefficient is 1. When the predicted result is entirely inconsistent with the actual situation, the coefficient is 0. When the prediction is a proper subset or true superset of the actual situation, the distance is between 0 and 1. We can get the algorithm's overall performance on the test set by averaging the prediction of all samples.



```python
from sklearn.metrics import jaccard_score
# Jackard coefficient (the better to close 1)
# Because the dataset is multiclassified, it needs to change the average parameter to micro instead of binary
# The original function is -> jaccard_similarity_score, but it is modified to jaccard_score in 2021
jaccrd_score = jaccard_score(valid_generator.classes, predictions,pos_label = "PAIDOFF",average='micro')
print(jaccrd_score)
```

*Figure 6.6: Create the jaccard coefficient function to use*

## 6.6 Cross Fold Validation

### *6.6.1 What is cross-folding validation?*

Cross-folding validation is a technique for evaluating ML models by training multiple ML models on a subset of the available input data and evaluating them on complementary subsets of the data. Common cross-folding validation methods include K-fold cross-validation, hierarchical cross-validation, etc.

In our project, we split the dataset into a validation set and a training set at a ratio of 0.5. After one training session, we swap the training and validation sets, and in the second training session, the training set is the validation set from the first training session and the validation set is the training set from the first training session. This constitutes a 2-fold cross-validation, and we can also run the model multiple times, thanks to the datagen.flow_from_directory() function with the parameter shuffle=True, which ensures that the data is randomly allocated for each run. Using these two methods, cross-folding validation can be accomplished.

### *6.6.2 two-fold cross-validation*

An example: we divide the dataset into two sets A B. The first training let A be the training set and B the validation set. The second training has A as the validation set and B as the training set.

First time:

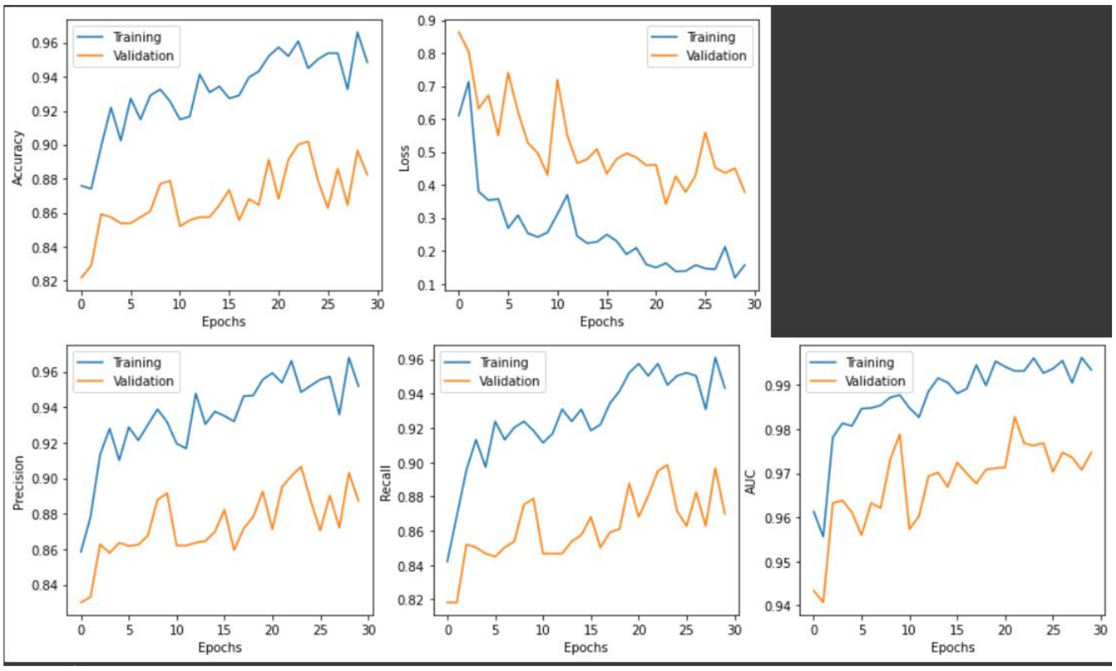


*Figure 6.7: Cross-folding validation first training*

Second time:

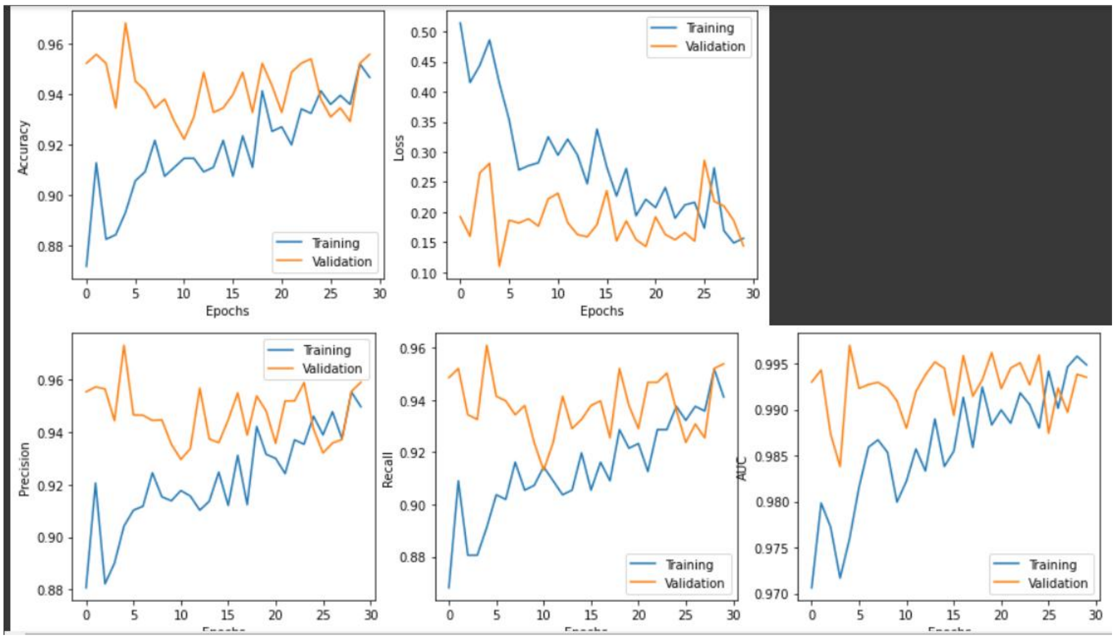


*Figure 6.8: Cross-folding verification second training*

# 7. Conclusion and Further Research

## 7.1 We have a preliminary conclusion to the weather prediction model through the above analysis and trial:

a. Although the test set's performance is good, there is a gap between it and the verification set. The small amount of sample data may cause our analysis.
b. The problem of overfitting inevitably appeared in the model, and we reduced the problem by Batch Normalisation.
c. In the data preprocessing stage, image size cropping and random processing can reduce the modelling burden and will not significantly impact the results.
d. Increasing the number of iterations for data samples could not significantly improve the model's performance.
e. Adding too much flatten layer will result in poor performance and should be adjusted or used less frequently.
f. More advanced and appropriate optimizers or activation functions should be adopted in the neural network model. Different choices will directly lead to differences in the final performance.

## 7.2 Because of some problems during the attempt, we can also do further research in the future:

a. When selecting the activation function, although "ReLu" performs well, it can be replaced with leaky relu because of the problem of dead relu.
b. Adopt larger and richer data sets and features.
c. In the preprocessing stage, more abundant photo processing can be adopted to avoid the occurrence of overfitting.
d. Explore the influence of the Batch Normalisation layer using location.

## 7.3 Effect of changing hyperparameters and increasing data on training results in cases where overfitting is found:

### *7.3.1 Some of the main reasons for overfitting*

1. There is a mismatch between the order of magnitude of the training set and the complexity of the model. The order of magnitude of the training set is smaller than the complexity of the model.

2. a mismatch between the distribution of features in the training and test sets.

3. excessive noise in the sample, so large that the model over-remembers the noise features and ignores the true input-output relationship.

4. the number of weight learning iterations (Overtraining) is large enough to fit the noise in the training data and the unrepresentative features in the training sample.

### *7.3.2 By comparing the original training data with five subsequent trials varying different hyperparameters:*

From Experiment 1 compared to the original training data we see that increasing the number of iterations is not effective in reducing overfitting, but rather increases the chance of overfitting as the number of iterations increases.
From Experiment 2 compared to the original training data we can see that adding flatten layers has no significant effect on reducing overfitting.
From Experiment 3 compared to the original training data, we can see that changing the activation function to a new, more advanced and academically respected activation function is more beneficial in reducing overfitting.
From Experiment 4 compared to the original training data we see that changing the optimiser has an effect on reducing overfitting, but not a significant one.
From Experiment 5 with the original training data, we see that changing the Batch Normalisation layer does not have much effect on changing the overfitting.

During our experiments, we had changed the dataset and when the number of photos in the dataset increased, we found that the overfitting was significantly alleviated and

the curve became smoother.
We ran this dataset with the ResNet152v2 model and the normal CNN five-layer model separately and found that the training results were better with the normal five-layer model and much worse with ResNet. We guess that our dataset is too small, but our model is too complex, especially for ResNet152v2. To train a dataset of just over a thousand images, we should have used a simpler model, and it would have been counterproductive to use an overly complex model.
We have varied the number of dopout layers in our experiments and we have found that when the number increases, overfitting is less likely to occur.

## 8. References

[1] Bjorck, N., Gomes, C.P., Selman, B. and Weinberger, K.Q., 2018. Understanding batch normalization. *Advances in neural information processing systems, 31*.

[2] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[3] Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

[4] Kattenborn, T., Leitloff, J., Schiefer, F. and Hinz, S., 2021. Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, *173*, pp.24-49.

[5] Kim, H.Y., 2015. Statistical notes for clinical researchers: Type I and type II errors in statistical decision. *Restorative Dentistry & Endodontics*, *40*(3), pp.249-252.

[6] Kraemer, H.C., 2014. Kappa coefficient. *Wiley StatsRef: statistics reference online*, pp.1-4.

[7] Mercioni, M.A. and Holban, S., 2020, May. The most used activation functions: Classic versus current. In *2020 International Conference on Development and Application Systems (DAS)* (pp. 141-145). IEEE.

[8] Monti, R.P., Tootoonian, S. and Cao, R., 2018, October. Avoiding degradation in deep feed-forward networks by phasing out skip-connections. In *International Conference on Artificial Neural Networks* (pp. 447-456). Springer, Cham.

[9] Niwattanakul, S., Singthongchai, J., Naenudorn, E. and Wanapu, S., 2013, March. Using of Jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists* (Vol. 1, No. 6, pp. 380-384).

[10] Norouzi, M., Fleet, D.J. and Salakhutdinov, R.R., 2012. Hamming distance metric learning. *Advances in neural information processing systems*, *25*.

[11] Sharma, S., Sharma, S. and Athaiya, A., 2017. Activation functions in neural networks. *towards data science*, *6*(12), pp.310-316.

[12] Visa, S., Ramsay, B., Ralescu, A.L. and Van Der Knaap, E., 2011. Confusion matrix-based feature selection. *MAICS*, *710*(1), pp.120-127.

[13] Xu, C., Lu, C., Liang, X., Gao, J., Zheng, W., Wang, T. and Yan, S., 2015. Multi-loss regularized deep neural network. *IEEE Transactions on Circuits and Systems for Video Technology*, *26*(12), pp.2273-2283.

[14] Xu, J., Li, Z., Du, B., Zhang, M. and Liu, J., 2020, July. Reluplex made more practical: Leaky ReLU. In *2020 IEEE Symposium on Computers and communications (ISCC)* (pp. 1-7). IEEE.

[15] Yacouby, R. and Axman, D., 2020, November. Probabilistic extension of precision, recall, and F1 score for more thorough evaluation of classification models. In *Proceedings of the first workshop on evaluation and comparison of NLP systems* (pp. 79-91).

[16] Zhang, Z., 2018, June. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)* (pp. 1-2). Ieee.