CS6005
DEEP LEARNING TECHNIQUES

PROJECT TITLE: Branch Feature Fusion Convolution Network for Remote Sensing Scene Classification

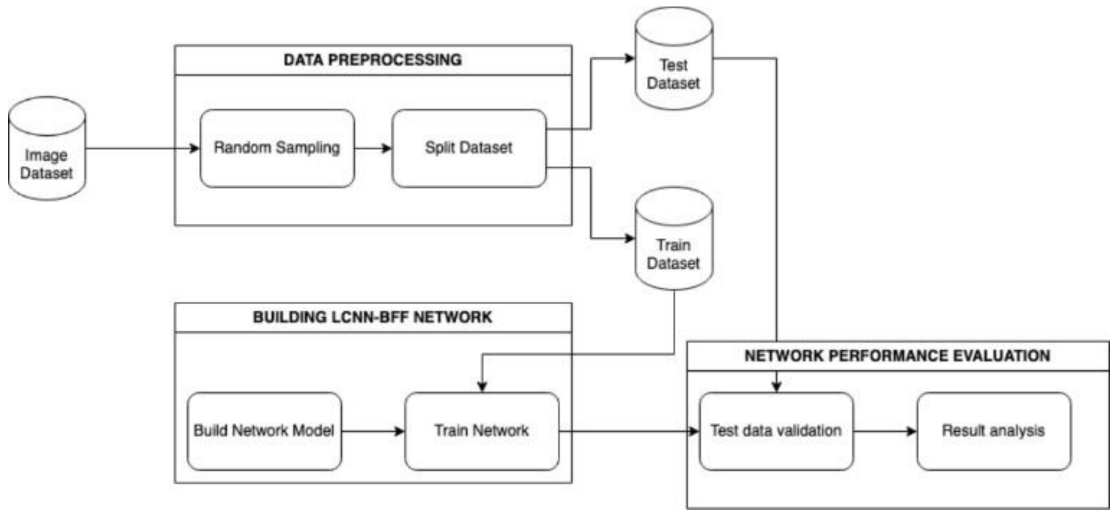
TEAM MEMBERS:
AJITESH M (2019103503)
VISHNUPRIYA N (2019103599)

ABSTRACT

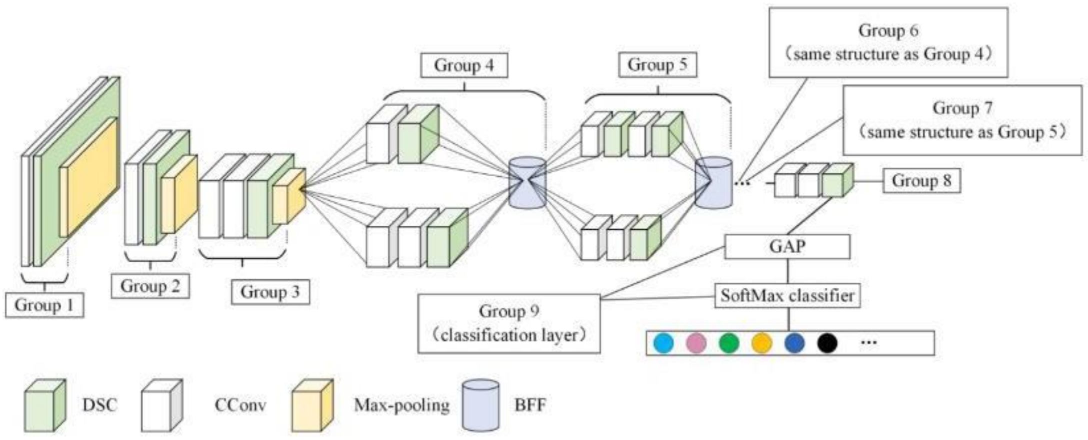
In the classification of remote sensing scenes, Convolutional Neural Networks (CNNs) have out-standing advantages. Deep CNN models with better classification performance typically have high complexity, whereas shallow CNN models with low complexity rarely achieve good classification performance for remote sensing images with complex spatial structures. A new lightweight CNN classification method based on branch feature fusion (LCNN-BFF) for remote sensing scene classification can be used for attaining these results. In contrast to a conventional single linear convolution structure, this model has a bilinear feature extraction structure. The BFF method is used to fuse the feature information extracted from the two branches, which improved the classification accuracy. In addition, combining depth wise separable convolution and conventional convolution to extract image features greatly reduced the complexity of the model on the premise of ensuring the accuracy of classification. We tested the method on four standard datasets. The experimental results showed that, compared with recent classification methods, the number of weight parameters of the proposed method only accounted for less than 5% of the other methods; however, the classification accuracy was equivalent to or even superior to certain high-performance classification methods.

NETWORK ARCHITECTURE

The following is the network architecture:



The following is the overall structure of the proposed LCNN-BFF method. DSC and conventional convolution (CConv).


Group 1
Group 2
Group 3
Group 4
Group 5
Group 6 (same structure as Group 4)
Group 7 (same structure as Group 5)
Group 8
GAP
SoftMax classifier
Group 9 (classification layer)
DSC
CConv
Max-pooling
BFF

The proposed LCNN-BFF network was composed of nine parts (Groups 1–9). According to the structure of the first three modules in the VGG16 network and the strategy of reducing model complexity introduced in this article, Groups 1–3 were defined.

- In Groups 1–3, the maximum pooling layer was used to down sample the remote sensing images, to reduce the spatial dimensions of the images, retain the main features of the images, and avoid the problem of overfitting.
- Groups 4–8 were mainly defined to extract representative features.
- Groups 4–7 used a bilinear convolution structure to extract more abundant feature information.

According to the bilinear convolution structure, the BFF method was proposed. BFF fuses the feature information extracted from the two branches to obtain more effective feature information.

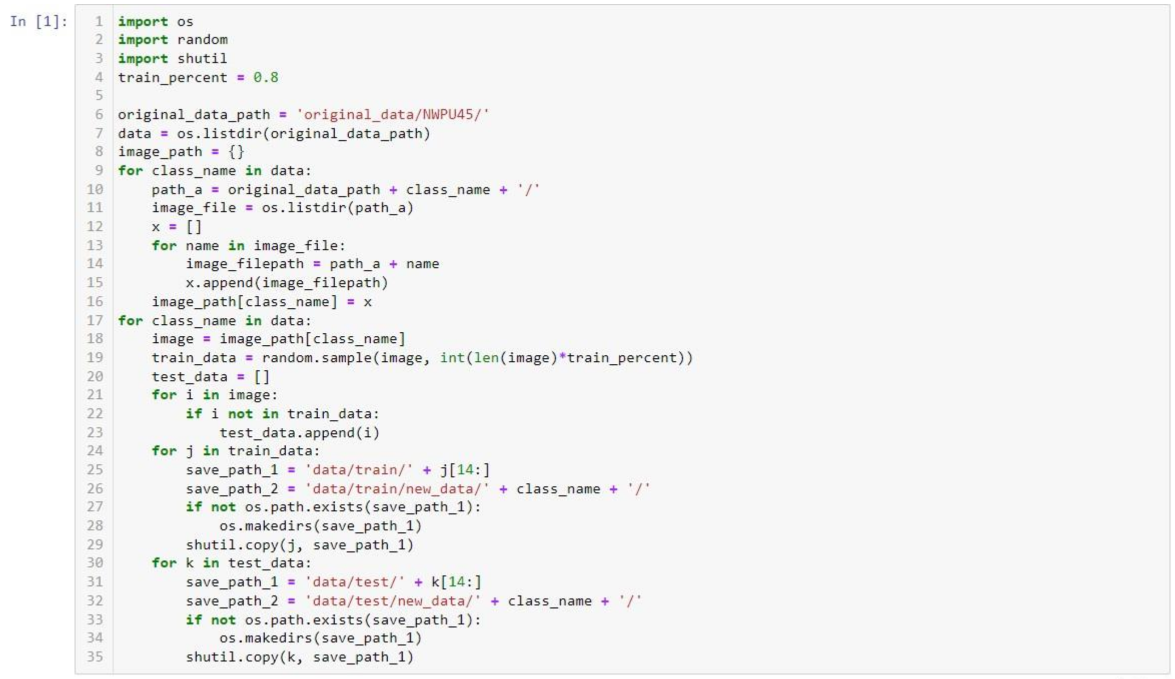
Group 9 was defined for classification, to convert the extracted feature information into the probability of each scene class. In the feature extraction structure (Groups 1–8), lightweight convolution DSC and CConv were combined to extract image features, which greatly reduced the complexity of the model.

Batch normalization (BN) was used to normalize the output of each volume accumulation layer, and then the rectified linear unit function was used to activate the neurons. After BN processing, the learning speed of the model could be accelerated and converged quickly. To a certain extent, this can avoid the problem of the gradient disappearing with the deepening of the network, and improve the generalization ability of the model. In addition, due to the small number of images in the divided training set, this may cause the problem of overfitting in the process of network training.

Therefore, an L2 regularization penalty was added to the weight of the convolution layer, and the penalty coefficient was 0.0005.

In Group 9, the global average pooling (GAP) was used instead of the flatten layer, which can reduce the model size and overfitting.

DATASET SPLITING AND RANDOM SAMPLING



```
In [1]:
import os
import random
import shutil
train_percent = 0.8

original_data_path = 'original_data/NWPU45/'
data = os.listdir(original_data_path)
image_path = {}
for class_name in data:
    path_a = original_data_path + class_name + '/'
    image_file = os.listdir(path_a)
    x = []
    for name in image_file:
        image_filepath = path_a + name
        x.append(image_filepath)
    image_path[class_name] = x
for class_name in data:
    image = image_path[class_name]
    train_data = random.sample(image, int(len(image)*train_percent))
    test_data = []
    for i in image:
        if i not in train_data:
            test_data.append(i)
    for j in train_data:
        save_path_1 = 'data/train/' + j[14:]
        save_path_2 = 'data/train/new_data/' + class_name + '/'
        if not os.path.exists(save_path_1):
            os.makedirs(save_path_1)
        shutil.copy(j, save_path_1)
    for k in test_data:
        save_path_1 = 'data/test/' + k[14:]
        save_path_2 = 'data/test/new_data/' + class_name + '/'
        if not os.path.exists(save_path_1):
            os.makedirs(save_path_1)
        shutil.copy(k, save_path_1)
```

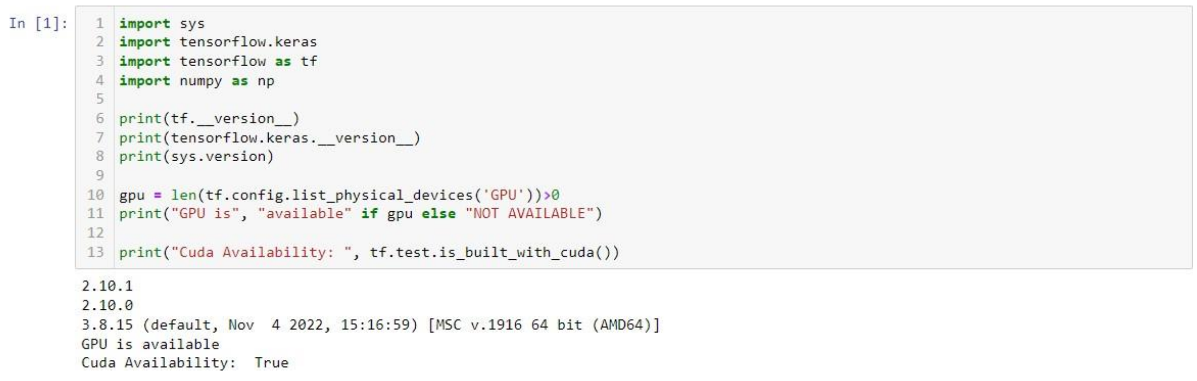
To train and test our model, we split the data through random sampling for training and testing. Here we have used 0.8 training percent; the data is split in 80:20 ration i.e., 80% of the data will be used for training the model while 20% will be used for testing the model that is built out of it.

CHECKING AVAILABILITY OF GPU AND CUDA

GPUs can perform multiple, simultaneous computations. This enables the distribution of training processes and can significantly speed machine learning operations. We can accumulate many cores that use fewer resources without sacrificing efficiency or power.

CUDA is a toolkit that includes various libraries and components. These provide support for debugging and optimization, compiling, documentation, runtimes, signal processing, and parallel algorithms. CUDA Toolkit libraries support all NVIDIA GPUs.

We check the availability of GPU and CUDA with the following code:

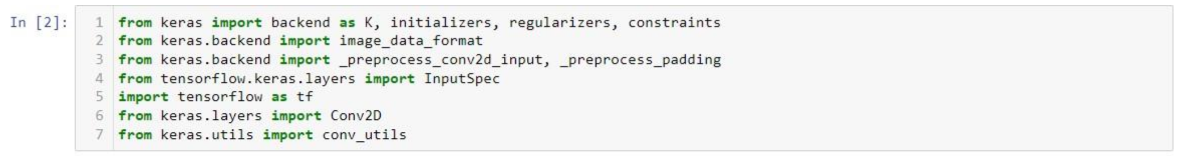


```
In [1]:
import sys
import tensorflow.keras
import tensorflow as tf
import numpy as np

print(tf.__version__)
print(tensorflow.keras.__version__)
print(sys.version)

gpu = len(tf.config.list_physical_devices('GPU'))>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")

print("Cuda Availability: ", tf.test.is_built_with_cuda())
```

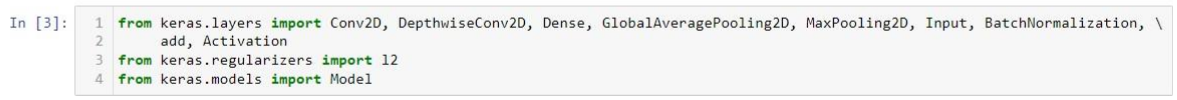
```
2.10.1
2.10.0
3.8.15 (default, Nov  4 2022, 15:16:59) [MSC v.1916 64 bit (AMD64)]
GPU is available
Cuda Availability:  True
```

IMPORTING REQUIRED PACKAGES

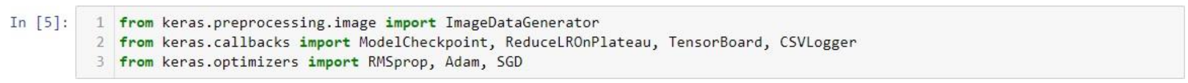
We import the following packages that are used later in the model from keras



In [2]:
```python
from keras import backend as K, initializers, regularizers, constraints
from keras.backend import image_data_format
from keras.backend import _preprocess_conv2d_input, _preprocess_padding
from tensorflow.keras.layers import InputSpec
import tensorflow as tf
from keras.layers import Conv2D
from keras.utils import conv_utils
```

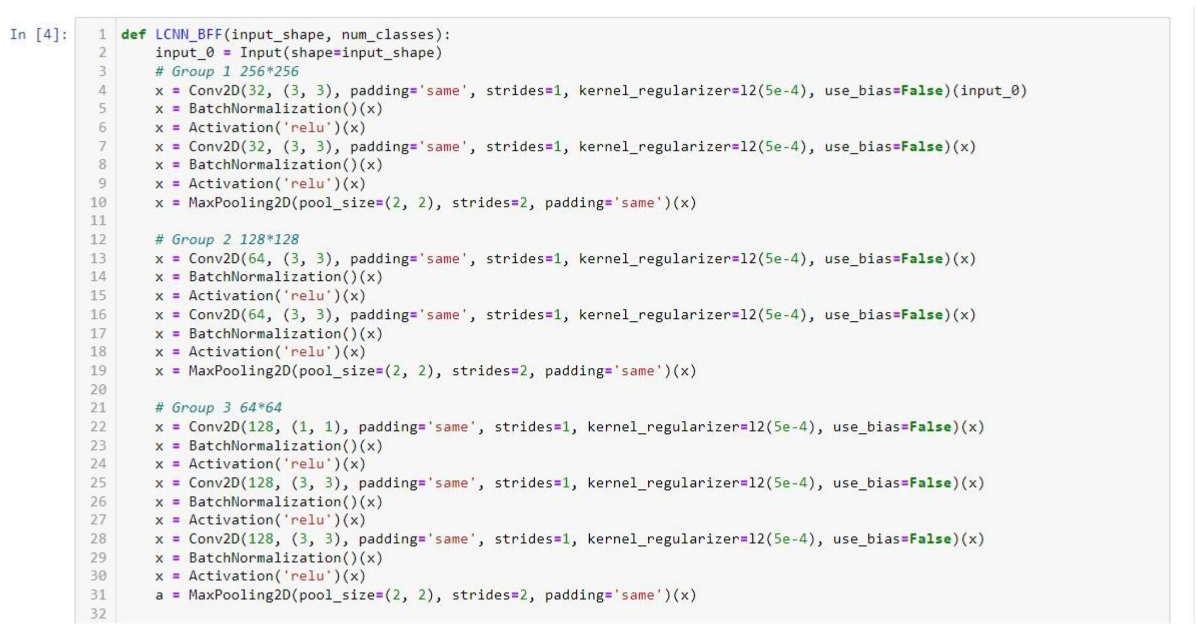


In [3]:
```python
from keras.layers import Conv2D, DepthwiseConv2D, Dense, GlobalAveragePooling2D, MaxPooling2D, Input, BatchNormalization, \
     add, Activation
from keras.regularizers import l2
from keras.models import Model
```

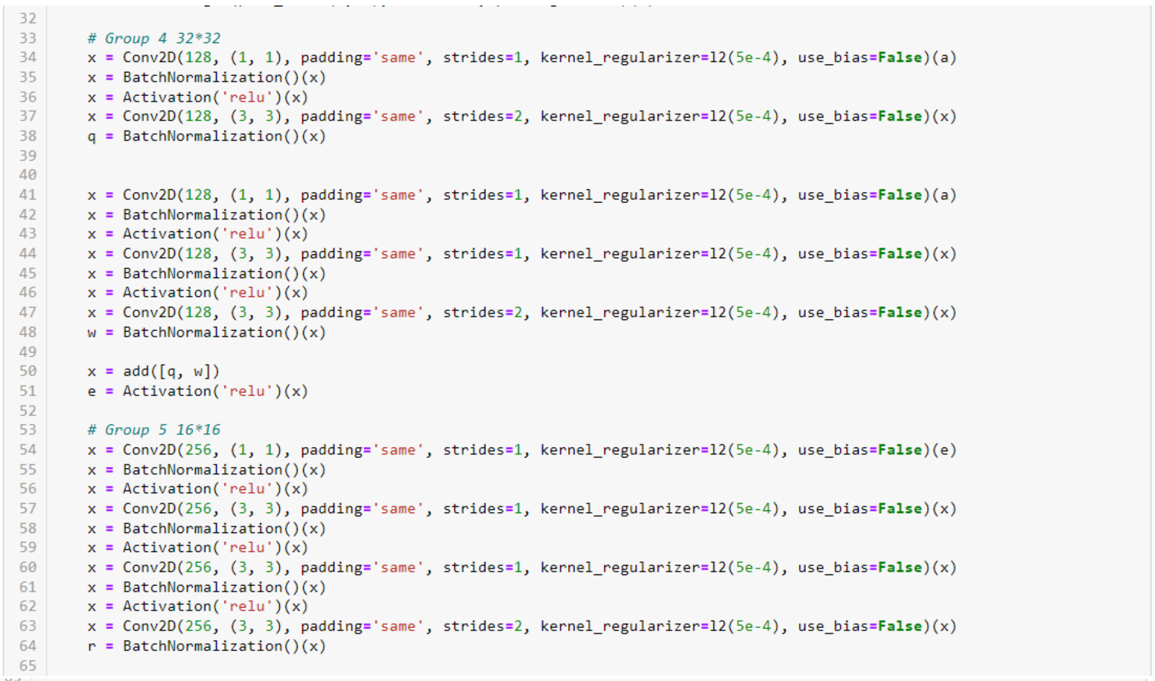


In [5]:
```python
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, TensorBoard, CSVLogger
from keras.optimizers import RMSprop, Adam, SGD
```

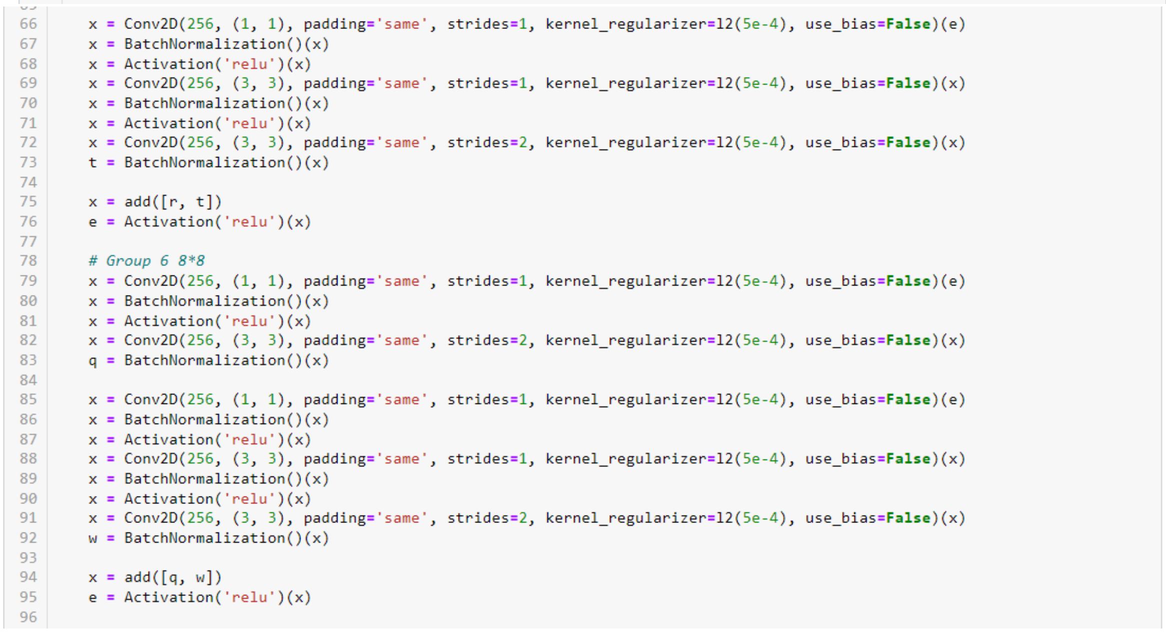
The following is the implementation of the model:



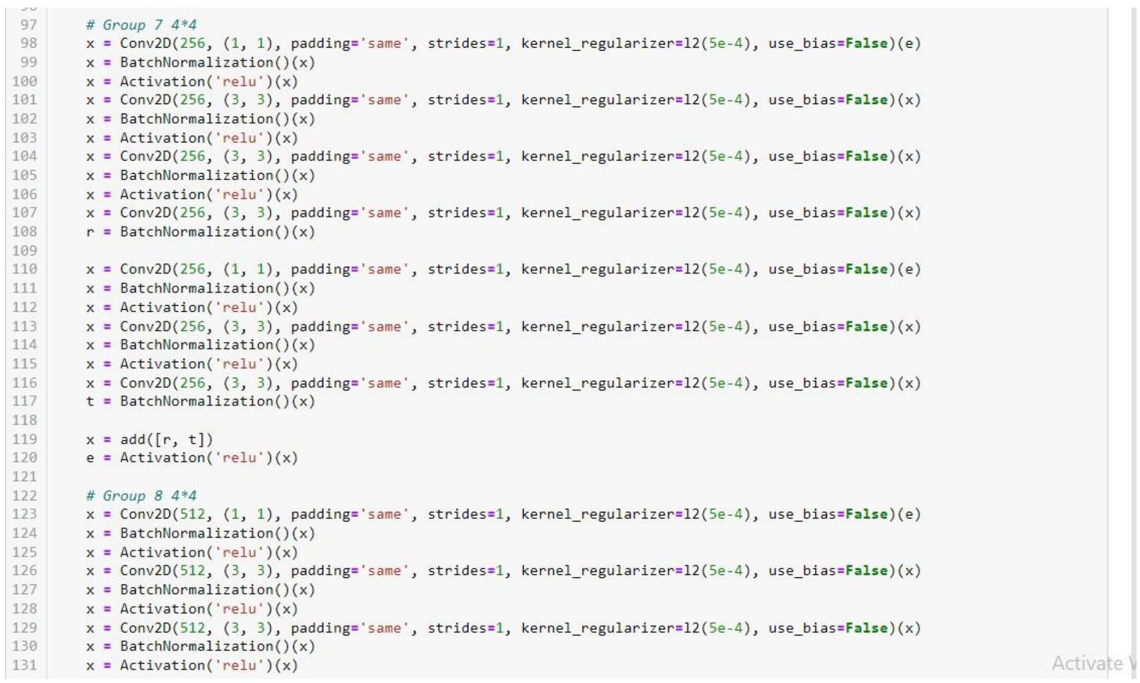
In [4]:
```python
def LCNN_BFF(input_shape, num_classes):
    input_0 = Input(shape=input_shape)
    # Group 1 256*256
    x = Conv2D(32, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(input_0)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(32, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(x)

    # Group 2 128*128
    x = Conv2D(64, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(64, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(x)

    # Group 3 64*64
    x = Conv2D(128, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    a = MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(x)

```



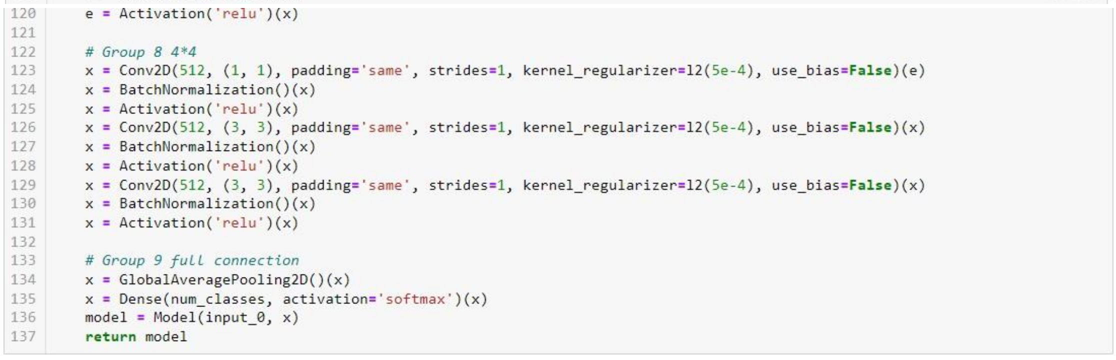
```
    # Group 4 32*32
    x = Conv2D(128, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(a)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    q = BatchNormalization()(x)


    x = Conv2D(128, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(a)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    w = BatchNormalization()(x)

    x = add([q, w])
    e = Activation('relu')(x)

    # Group 5 16*16
    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    r = BatchNormalization()(x)

```



```
    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    t = BatchNormalization()(x)

    x = add([r, t])
    e = Activation('relu')(x)

    # Group 6 8*8
    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    q = BatchNormalization()(x)

    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=2, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    w = BatchNormalization()(x)

    x = add([q, w])
    e = Activation('relu')(x)

```



```python
    # Group 7 4*4
    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    r = BatchNormalization()(x)

    x = Conv2D(256, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    t = BatchNormalization()(x)

    x = add([r, t])
    e = Activation('relu')(x)

    # Group 8 4*4
    x = Conv2D(512, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
```



```python
    e = Activation('relu')(x)

    # Group 8 4*4
    x = Conv2D(512, (1, 1), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(e)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', strides=1, kernel_regularizer=l2(5e-4), use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
```

```python

    # Group 9 full connection
    x = GlobalAveragePooling2D()(x)
    x = Dense(num_classes, activation='softmax')(x)
    model = Model(input_0, x)
    return model
```

TRAINING THE NETWORK

HYPERPARAMETERS

Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning.

- **Learning Rate** – The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. Here we have used a learning rate of 1e-2, i.e., 0.01 (1 percent)

- **Number of Epochs** – The number of epochs is the number of complete passes through the training dataset. Here we have set num_epochs to 1000.

- **Momentum** – Momentum is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space. We have set the momentum to 0.9.

- **Batch Size** – The size of a batch refers to the number of training examples utilized in one iteration. We have taken a batch size of 16.

OPTIMISERS

In deep learning, optimizers are used to adjust the parameters for a model. The purpose of an optimizer is to adjust model weights to maximize a loss function.
The optimizers we have used are **SGD and ReduceLROnPlateau.**

1. **SGD (Stochastic Gradient Descent)**

Gradient descent is the preferred way to optimize neural networks and many other machine learning algorithms but is often used as a black box.
Stochastic gradient descent (SGD) performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J\left(\theta; x^{(i)}; y^{(i)}\right).$$

SGD handles redundancy by performing one update at a time. It is therefore usually much faster and performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

2. **ReduceLROnPlateau**

As the name suggests, it reduces learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

It has certain arguments:

- **monitor:** quantity to be monitored.
- **factor:** factor by which the learning rate will be reduced. new_lr = lr * factor.
- **patience:** number of epochs with no improvement after which learning rate will be reduced.
- **verbose:** int. 0: quiet, 1: update messages.

Here we monitor the value lost (val_loss) with a factor 0.1, patience 32 and verbose 1.

LOSS FUNCTION

The loss function is used as a way to measure how well the model is performing. It maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.

Here we use **categorical cross** entropy loss function. It computes the categorical cross entropy loss.

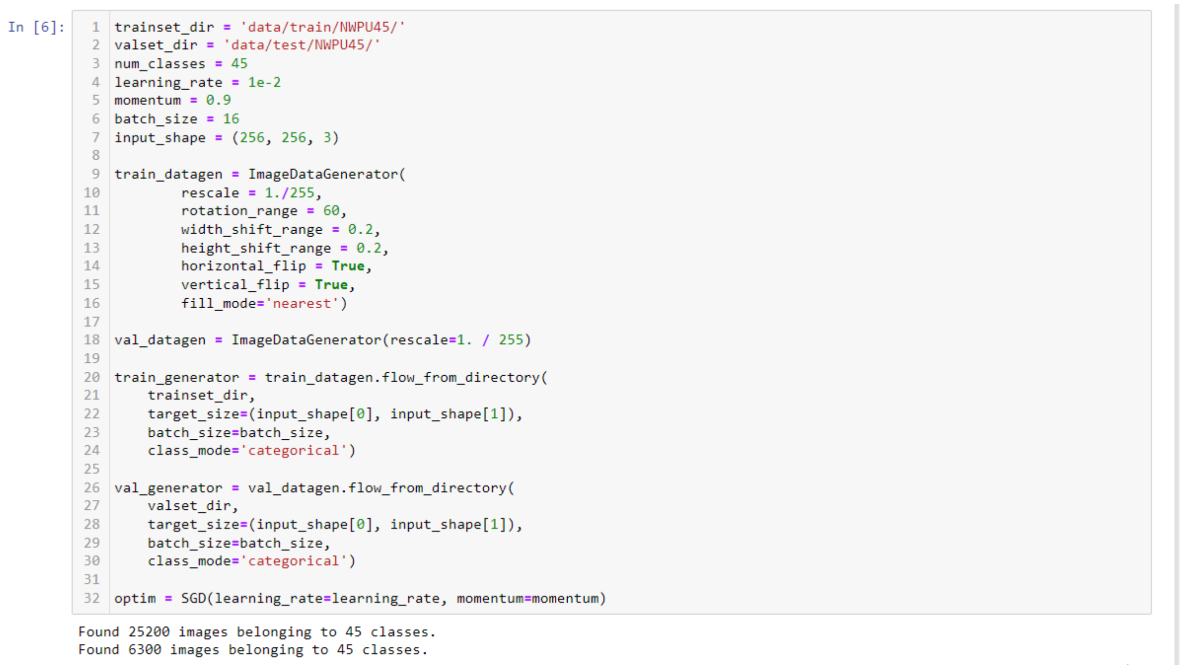
Arguments

- **y_true:** Tensor of one-hot true targets.
- **y_pred:** Tensor of predicted targets.
- **from_logits:** Whether y_pred is expected to be a logits tensor. By default, we assume that y_pred encodes a probability distribution.
- **label_smoothing:** Float in [0, 1]. If > 0 then smooth the labels. For example, if 0.1, use 0.1 / num_classes for non-target labels and 0.9 + 0.1 / num_classes for target labels.
- **axis:** Defaults to -1. The dimension along which the entropy is computed.

Returns

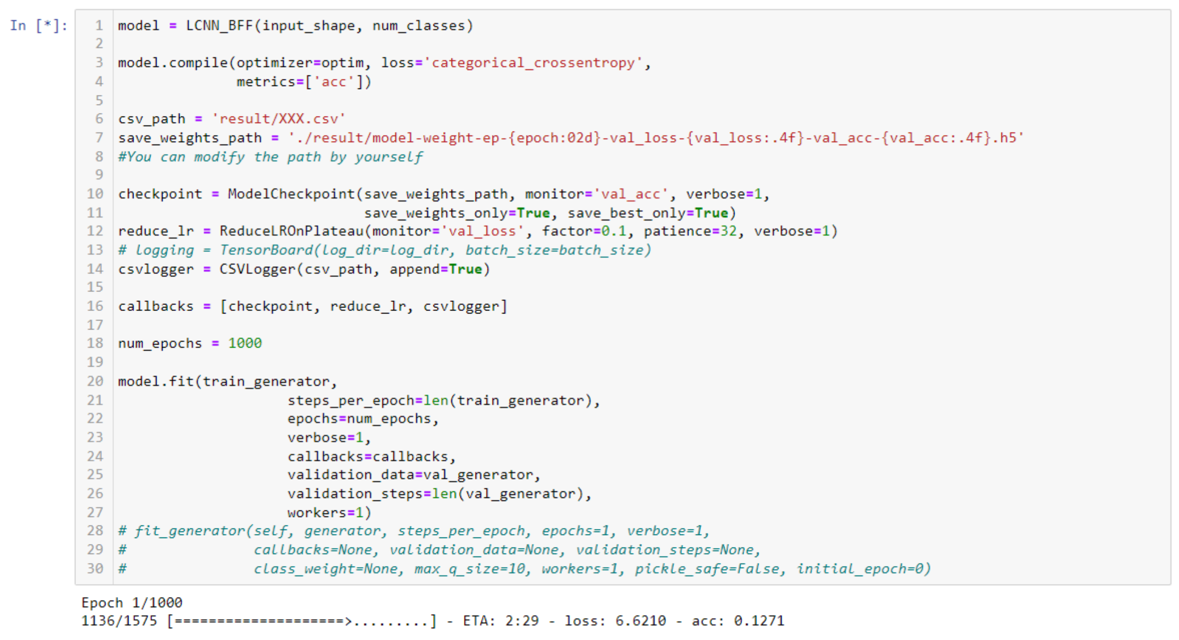
Categorical cross entropy loss value.

The implementation for training the network:



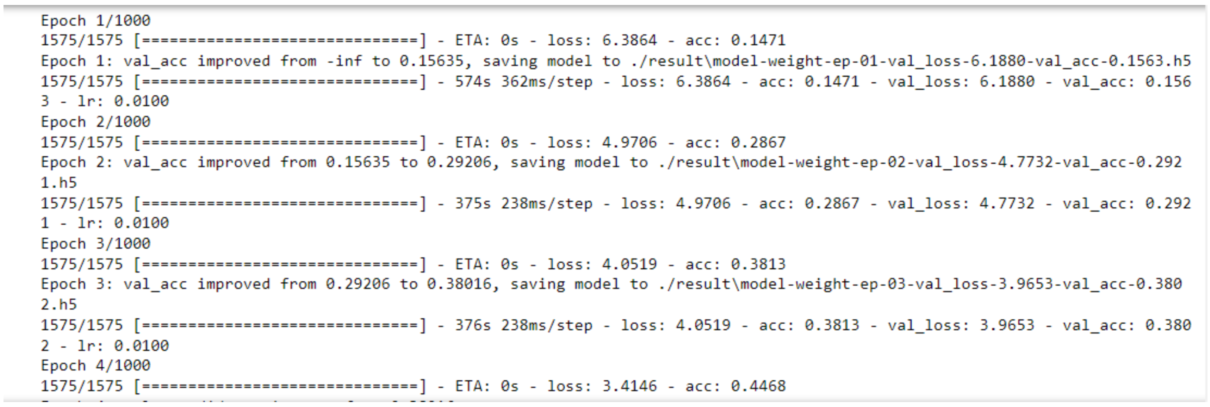
```
In [6]:
trainset_dir = 'data/train/NWPU45/'
valset_dir = 'data/test/NWPU45/'
num_classes = 45
learning_rate = 1e-2
momentum = 0.9
batch_size = 16
input_shape = (256, 256, 3)

train_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 60,
        width_shift_range = 0.2,
        height_shift_range = 0.2,
        horizontal_flip = True,
        vertical_flip = True,
        fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    trainset_dir,
    target_size=(input_shape[0], input_shape[1]),
    batch_size=batch_size,
    class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
    valset_dir,
    target_size=(input_shape[0], input_shape[1]),
    batch_size=batch_size,
    class_mode='categorical')

optim = SGD(learning_rate=learning_rate, momentum=momentum)
```

```
Found 25200 images belonging to 45 classes.
Found 6300 images belonging to 45 classes.
```

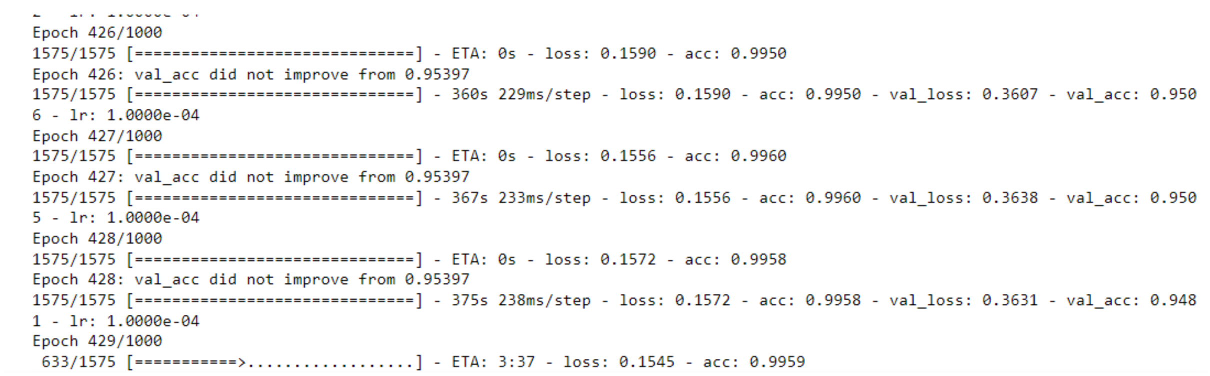


```
In [*]:
model = LCNN_BFF(input_shape, num_classes)

model.compile(optimizer=optim, loss='categorical_crossentropy',
              metrics=['acc'])

csv_path = 'result/XXX.csv'
save_weights_path = './result/model-weight-ep-{epoch:02d}-val_loss-{val_loss:.4f}-val_acc-{val_acc:.4f}.h5'
#You can modify the path by yourself

checkpoint = ModelCheckpoint(save_weights_path, monitor='val_acc', verbose=1,
                              save_weights_only=True, save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=32, verbose=1)
# logging = TensorBoard(log_dir=log_dir, batch_size=batch_size)
csvlogger = CSVLogger(csv_path, append=True)

callbacks = [checkpoint, reduce_lr, csvlogger]

num_epochs = 1000

model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=num_epochs,
                    verbose=1,
                    callbacks=callbacks,
                    validation_data=val_generator,
                    validation_steps=len(val_generator),
                    workers=1)
# fit_generator(self, generator, steps_per_epoch, epochs=1, verbose=1,
#               callbacks=None, validation_data=None, validation_steps=None,
#               class_weight=None, max_q_size=10, workers=1, pickle_safe=False, initial_epoch=0)
```

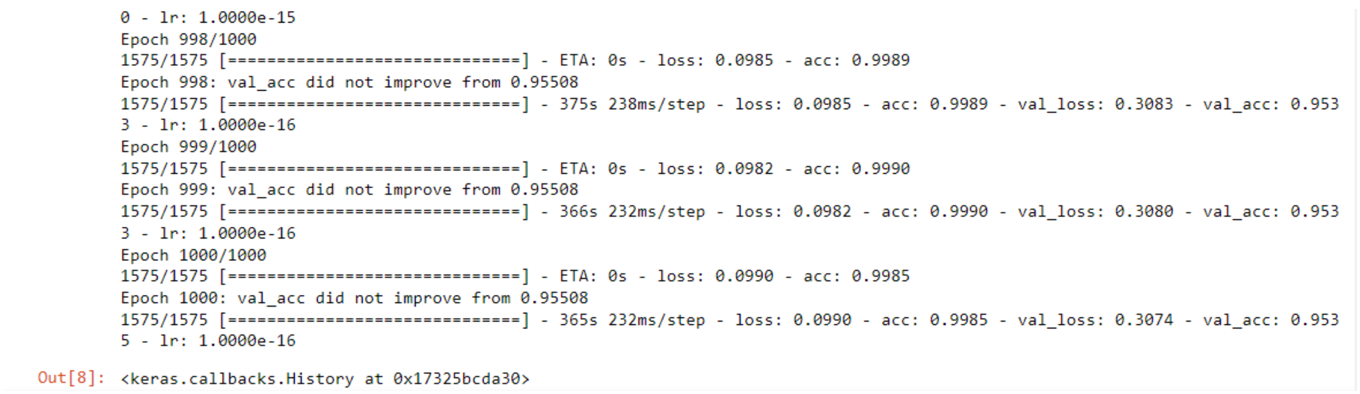
```
Epoch 1/1000
1136/1575 [====================>.........] - ETA: 2:29 - loss: 6.6210 - acc: 0.1271
```



```
Epoch 1/1000
1575/1575 [==============================] - ETA: 0s - loss: 6.3864 - acc: 0.1471
Epoch 1: val_acc improved from -inf to 0.15635, saving model to ./result\model-weight-ep-01-val_loss-6.1880-val_acc-0.1563.h5
1575/1575 [==============================] - 574s 362ms/step - loss: 6.3864 - acc: 0.1471 - val_loss: 6.1880 - val_acc: 0.156
3 - lr: 0.0100
Epoch 2/1000
1575/1575 [==============================] - ETA: 0s - loss: 4.9706 - acc: 0.2867
Epoch 2: val_acc improved from 0.15635 to 0.29206, saving model to ./result\model-weight-ep-02-val_loss-4.7732-val_acc-0.292
1.h5
1575/1575 [==============================] - 375s 238ms/step - loss: 4.9706 - acc: 0.2867 - val_loss: 4.7732 - val_acc: 0.292
1 - lr: 0.0100
Epoch 3/1000
1575/1575 [==============================] - ETA: 0s - loss: 4.0519 - acc: 0.3813
Epoch 3: val_acc improved from 0.29206 to 0.38016, saving model to ./result\model-weight-ep-03-val_loss-3.9653-val_acc-0.380
2.h5
1575/1575 [==============================] - 376s 238ms/step - loss: 4.0519 - acc: 0.3813 - val_loss: 3.9653 - val_acc: 0.380
2 - lr: 0.0100
Epoch 4/1000
1575/1575 [==============================] - ETA: 0s - loss: 3.4146 - acc: 0.4468
```

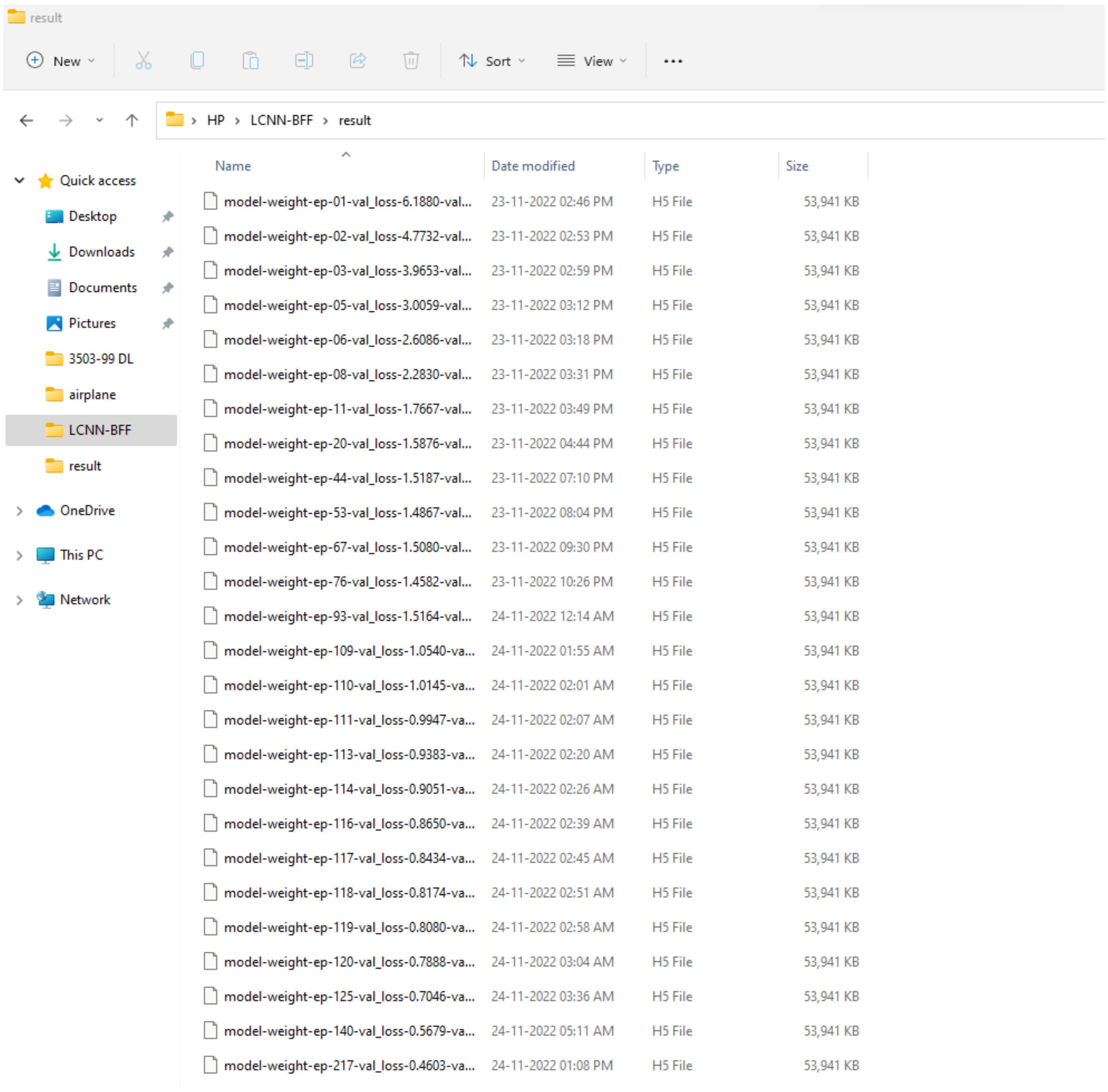


```
Epoch 426/1000
1575/1575 [==============================] - ETA: 0s - loss: 0.1590 - acc: 0.9950
Epoch 426: val_acc did not improve from 0.95397
1575/1575 [==============================] - 360s 229ms/step - loss: 0.1590 - acc: 0.9950 - val_loss: 0.3607 - val_acc: 0.950
6 - lr: 1.0000e-04
Epoch 427/1000
1575/1575 [==============================] - ETA: 0s - loss: 0.1556 - acc: 0.9960
Epoch 427: val_acc did not improve from 0.95397
1575/1575 [==============================] - 367s 233ms/step - loss: 0.1556 - acc: 0.9960 - val_loss: 0.3638 - val_acc: 0.950
5 - lr: 1.0000e-04
Epoch 428/1000
1575/1575 [==============================] - ETA: 0s - loss: 0.1572 - acc: 0.9958
Epoch 428: val_acc did not improve from 0.95397
1575/1575 [==============================] - 375s 238ms/step - loss: 0.1572 - acc: 0.9958 - val_loss: 0.3631 - val_acc: 0.948
1 - lr: 1.0000e-04
Epoch 429/1000
 633/1575 [===========>..................] - ETA: 3:37 - loss: 0.1545 - acc: 0.9959
```

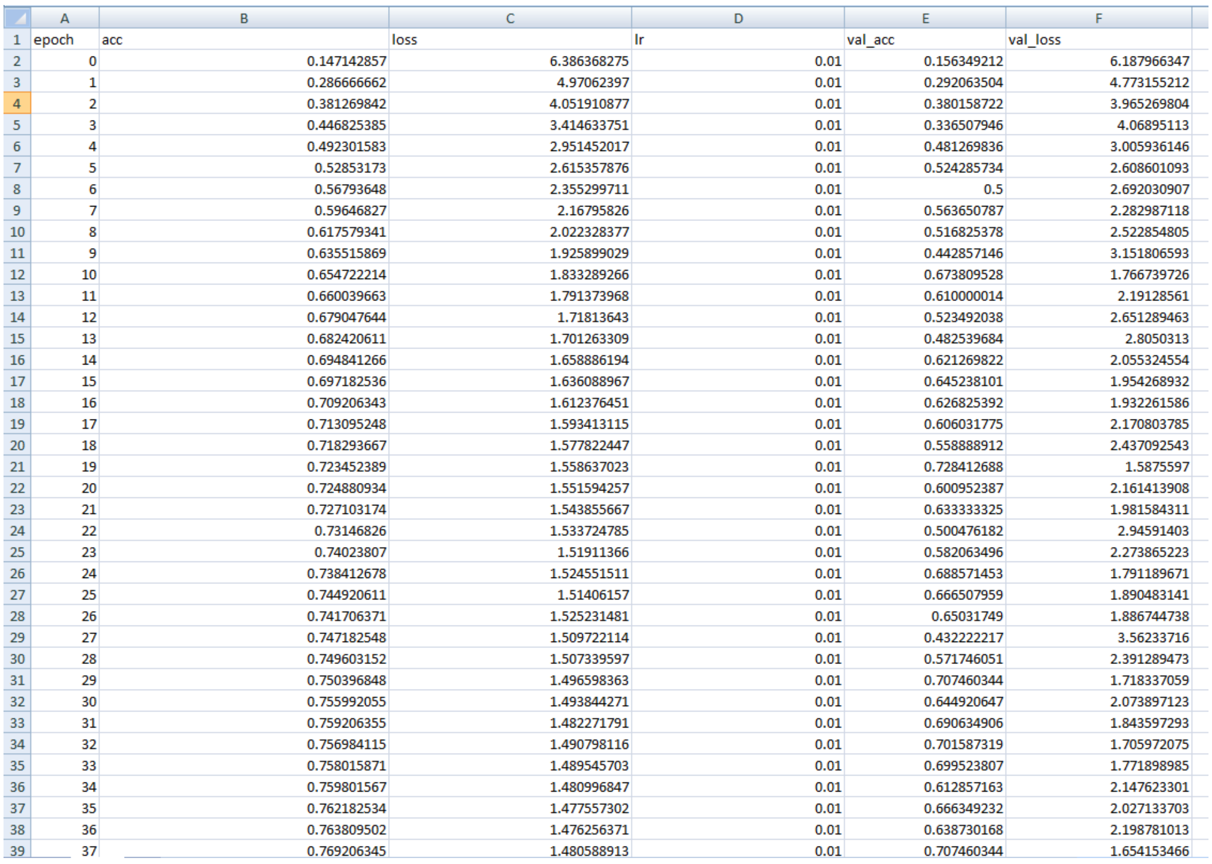


```
      0 - lr: 1.0000e-15
      Epoch 998/1000
      1575/1575 [==============================] - ETA: 0s - loss: 0.0985 - acc: 0.9989
      Epoch 998: val_acc did not improve from 0.95508
      1575/1575 [==============================] - 375s 238ms/step - loss: 0.0985 - acc: 0.9989 - val_loss: 0.3083 - val_acc: 0.953
      3 - lr: 1.0000e-16
      Epoch 999/1000
      1575/1575 [==============================] - ETA: 0s - loss: 0.0982 - acc: 0.9990
      Epoch 999: val_acc did not improve from 0.95508
      1575/1575 [==============================] - 366s 232ms/step - loss: 0.0982 - acc: 0.9990 - val_loss: 0.3080 - val_acc: 0.953
      3 - lr: 1.0000e-16
      Epoch 1000/1000
      1575/1575 [==============================] - ETA: 0s - loss: 0.0990 - acc: 0.9985
      Epoch 1000: val_acc did not improve from 0.95508
      1575/1575 [==============================] - 365s 232ms/step - loss: 0.0990 - acc: 0.9985 - val_loss: 0.3074 - val_acc: 0.953
      5 - lr: 1.0000e-16

Out[8]: <keras.callbacks.History at 0x17325bcda30>
```

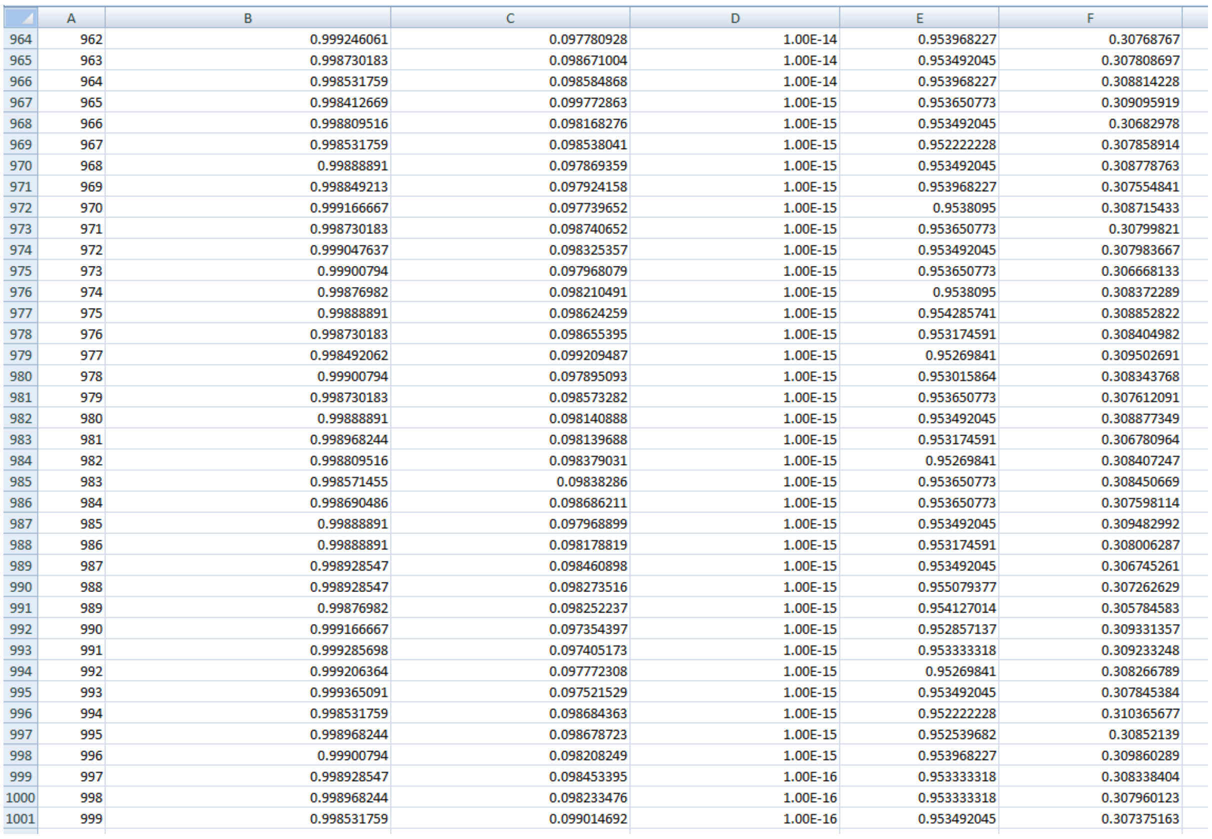
As the result of the training phase, the checkpoints of the model which produce the best results are stored, and the result of each epoch is logged in a CSV file.



Name	Date modified	Type	Size
model-weight-ep-01-val_loss-6.1880-val...	23-11-2022 02:46 PM	H5 File	53,941 KB
model-weight-ep-02-val_loss-4.7732-val...	23-11-2022 02:53 PM	H5 File	53,941 KB
model-weight-ep-03-val_loss-3.9653-val...	23-11-2022 02:59 PM	H5 File	53,941 KB
model-weight-ep-05-val_loss-3.0059-val...	23-11-2022 03:12 PM	H5 File	53,941 KB
model-weight-ep-06-val_loss-2.6086-val...	23-11-2022 03:18 PM	H5 File	53,941 KB
model-weight-ep-08-val_loss-2.2830-val...	23-11-2022 03:31 PM	H5 File	53,941 KB
model-weight-ep-11-val_loss-1.7667-val...	23-11-2022 03:49 PM	H5 File	53,941 KB
model-weight-ep-20-val_loss-1.5876-val...	23-11-2022 04:44 PM	H5 File	53,941 KB
model-weight-ep-44-val_loss-1.5187-val...	23-11-2022 07:10 PM	H5 File	53,941 KB
model-weight-ep-53-val_loss-1.4867-val...	23-11-2022 08:04 PM	H5 File	53,941 KB
model-weight-ep-67-val_loss-1.5080-val...	23-11-2022 09:30 PM	H5 File	53,941 KB
model-weight-ep-76-val_loss-1.4582-val...	23-11-2022 10:26 PM	H5 File	53,941 KB
model-weight-ep-93-val_loss-1.5164-val...	24-11-2022 12:14 AM	H5 File	53,941 KB
model-weight-ep-109-val_loss-1.0540-va...	24-11-2022 01:55 AM	H5 File	53,941 KB
model-weight-ep-110-val_loss-1.0145-va...	24-11-2022 02:01 AM	H5 File	53,941 KB
model-weight-ep-111-val_loss-0.9947-va...	24-11-2022 02:07 AM	H5 File	53,941 KB
model-weight-ep-113-val_loss-0.9383-va...	24-11-2022 02:20 AM	H5 File	53,941 KB
model-weight-ep-114-val_loss-0.9051-va...	24-11-2022 02:26 AM	H5 File	53,941 KB
model-weight-ep-116-val_loss-0.8650-va...	24-11-2022 02:39 AM	H5 File	53,941 KB
model-weight-ep-117-val_loss-0.8434-va...	24-11-2022 02:45 AM	H5 File	53,941 KB
model-weight-ep-118-val_loss-0.8174-va...	24-11-2022 02:51 AM	H5 File	53,941 KB
model-weight-ep-119-val_loss-0.8080-va...	24-11-2022 02:58 AM	H5 File	53,941 KB
model-weight-ep-120-val_loss-0.7888-va...	24-11-2022 03:04 AM	H5 File	53,941 KB
model-weight-ep-125-val_loss-0.7046-va...	24-11-2022 03:36 AM	H5 File	53,941 KB
model-weight-ep-140-val_loss-0.5679-va...	24-11-2022 05:11 AM	H5 File	53,941 KB
model-weight-ep-217-val_loss-0.4603-va...	24-11-2022 01:08 PM	H5 File	53,941 KB



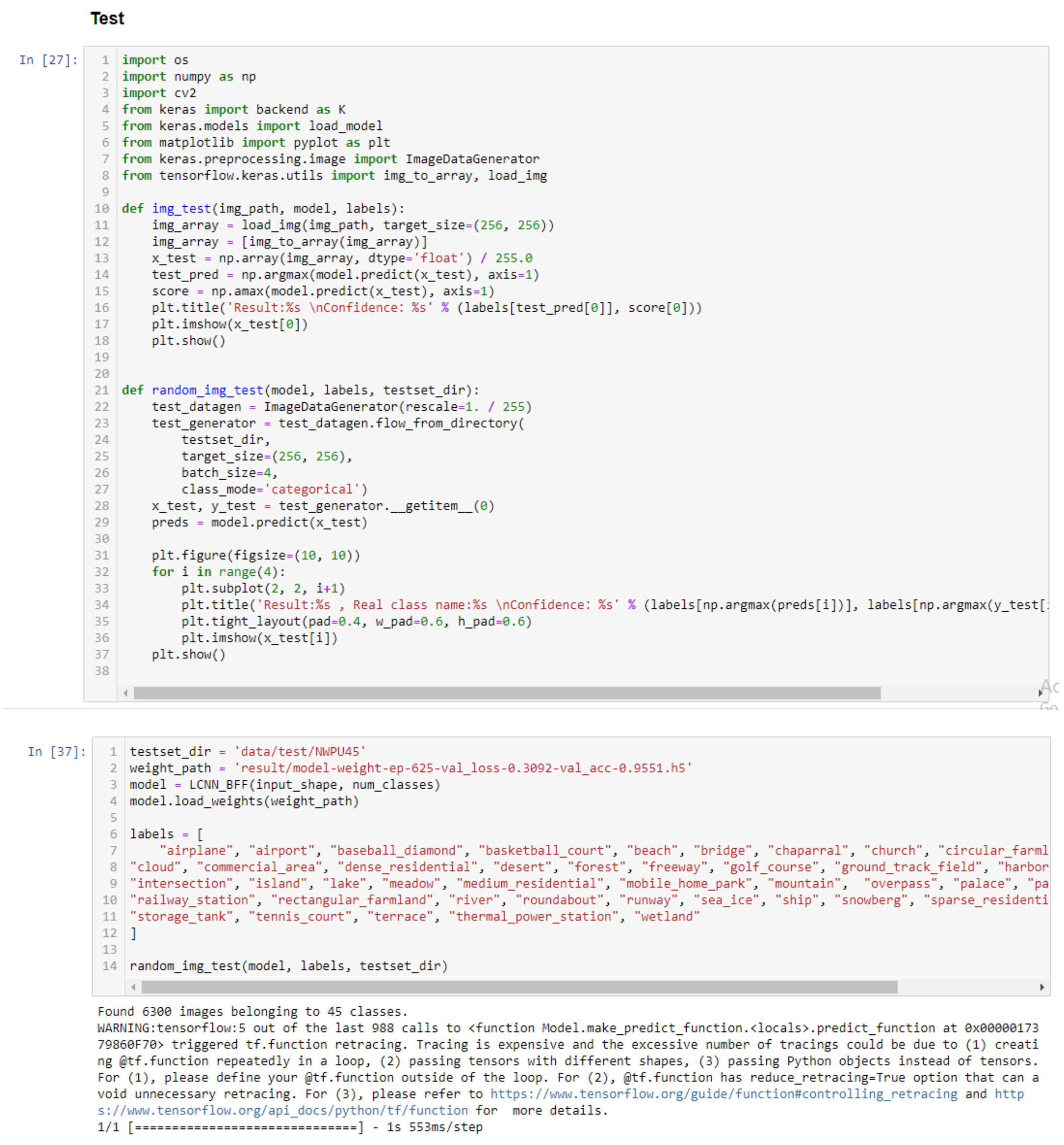
	A	B	C	D	E	F
1	epoch	acc	loss	lr	val_acc	val_loss
2	0	0.147142857	6.386368275	0.01	0.156349212	6.187966347
3	1	0.286666662	4.97062397	0.01	0.292063504	4.773155212
4	2	0.381269842	4.051910877	0.01	0.380158722	3.965269804
5	3	0.446825385	3.414633751	0.01	0.336507946	4.06895113
6	4	0.492301583	2.951452017	0.01	0.481269836	3.005936146
7	5	0.52853173	2.615357876	0.01	0.524285734	2.608601093
8	6	0.56793648	2.355299711	0.01	0.5	2.692030907
9	7	0.59646827	2.16795826	0.01	0.563650787	2.282987118
10	8	0.617579341	2.022328377	0.01	0.516825378	2.522854805
11	9	0.635515869	1.925899029	0.01	0.442857146	3.151806593
12	10	0.654722214	1.833289266	0.01	0.673809528	1.766739726
13	11	0.660039663	1.791373968	0.01	0.610000014	2.19128561
14	12	0.679047644	1.71813643	0.01	0.523492038	2.651289463
15	13	0.682420611	1.701263309	0.01	0.482539684	2.8050313
16	14	0.694841266	1.658886194	0.01	0.621269822	2.055324554
17	15	0.697182536	1.636088967	0.01	0.645238101	1.954268932
18	16	0.709206343	1.612376451	0.01	0.626825392	1.932261586
19	17	0.713095248	1.593413115	0.01	0.606031775	2.170803785
20	18	0.718293667	1.577822447	0.01	0.558888912	2.437092543
21	19	0.723452389	1.558637023	0.01	0.728412688	1.5875597
22	20	0.724880934	1.55159425	0.01	0.600952387	2.161413908
23	21	0.727103174	1.543855667	0.01	0.633333325	1.981584311
24	22	0.73146826	1.533724785	0.01	0.500476182	2.94591403
25	23	0.74023807	1.51911366	0.01	0.582063496	2.273865223
26	24	0.738412678	1.524551511	0.01	0.688571453	1.791189671
27	25	0.744920611	1.51406157	0.01	0.666507959	1.890483141
28	26	0.741706371	1.525231481	0.01	0.65031749	1.886744738
29	27	0.747182548	1.509722114	0.01	0.432222217	3.56233716
30	28	0.749603152	1.507339597	0.01	0.571746051	2.391289473
31	29	0.750396848	1.496598363	0.01	0.707460344	1.718337059
32	30	0.755992055	1.493844271	0.01	0.644920647	2.073897123
33	31	0.759206355	1.482271791	0.01	0.690634906	1.843597293
34	32	0.756984115	1.490798116	0.01	0.701587319	1.705972075
35	33	0.758015871	1.489545703	0.01	0.699523807	1.771898985
36	34	0.759801567	1.480996847	0.01	0.612857163	2.147623301
37	35	0.762182534	1.477557302	0.01	0.666349232	2.027133703
38	36	0.763809502	1.476256371	0.01	0.638730168	2.198781013
39	37	0.769206345	1.480588913	0.01	0.707460344	1.654153466



	A	B	C	D	E	F
964	962	0.999246061	0.097780928	1.00E-14	0.953968227	0.30768767
965	963	0.998730183	0.098671004	1.00E-14	0.953492045	0.307808697
966	964	0.998531759	0.098584868	1.00E-14	0.953968227	0.308814228
967	965	0.998412669	0.099772863	1.00E-15	0.953650773	0.309095919
968	966	0.998809516	0.098168276	1.00E-15	0.953492045	0.30682978
969	967	0.998531759	0.098538041	1.00E-15	0.952222228	0.307858914
970	968	0.99888891	0.097869359	1.00E-15	0.953492045	0.308778763
971	969	0.998849213	0.097924158	1.00E-15	0.953968227	0.307554841
972	970	0.999166667	0.097739652	1.00E-15	0.9538095	0.308715433
973	971	0.998730183	0.098740652	1.00E-15	0.953650773	0.30799821
974	972	0.999047637	0.098325357	1.00E-15	0.953492045	0.307983667
975	973	0.99900794	0.097968079	1.00E-15	0.953650773	0.306668133
976	974	0.99876982	0.098210491	1.00E-15	0.9538095	0.308372289
977	975	0.99888891	0.098624259	1.00E-15	0.954285741	0.308852822
978	976	0.998730183	0.098655395	1.00E-15	0.953174591	0.308404982
979	977	0.998492062	0.099209487	1.00E-15	0.95269841	0.309502691
980	978	0.99900794	0.097895093	1.00E-15	0.953015864	0.308343768
981	979	0.998730183	0.098573282	1.00E-15	0.953650773	0.307612091
982	980	0.99888891	0.098140888	1.00E-15	0.953492045	0.308877349
983	981	0.998968244	0.098139688	1.00E-15	0.953174591	0.306780964
984	982	0.998809516	0.098379031	1.00E-15	0.95269841	0.308407247
985	983	0.998571455	0.09838286	1.00E-15	0.953650773	0.308450669
986	984	0.998690486	0.098686211	1.00E-15	0.953650773	0.307598114
987	985	0.99888891	0.097968899	1.00E-15	0.953492045	0.309482992
988	986	0.99888891	0.098178819	1.00E-15	0.953174591	0.308006287
989	987	0.998928547	0.098460898	1.00E-15	0.953492045	0.306745261
990	988	0.998928547	0.098273516	1.00E-15	0.955079377	0.307262629
991	989	0.99876982	0.098252237	1.00E-15	0.954127014	0.305784583
992	990	0.999166667	0.097354397	1.00E-15	0.952857137	0.309331357
993	991	0.999285698	0.097405173	1.00E-15	0.953333318	0.309233248
994	992	0.999206364	0.097772308	1.00E-15	0.95269841	0.308266789
995	993	0.999365091	0.097521529	1.00E-15	0.953492045	0.307845384
996	994	0.998531759	0.098684363	1.00E-15	0.952222228	0.310365677
997	995	0.998968244	0.098678723	1.00E-15	0.952539682	0.30852139
998	996	0.99900794	0.098208249	1.00E-15	0.953968227	0.309860289
999	997	0.998928547	0.098453395	1.00E-16	0.953333318	0.308338404
1000	998	0.998968244	0.098233476	1.00E-16	0.953333318	0.307960123
1001	999	0.998531759	0.099014692	1.00E-16	0.953492045	0.307375163

TESTING THE NETWORK

In the testing the network, random images are taken from the available test dataset and is passed to the model.



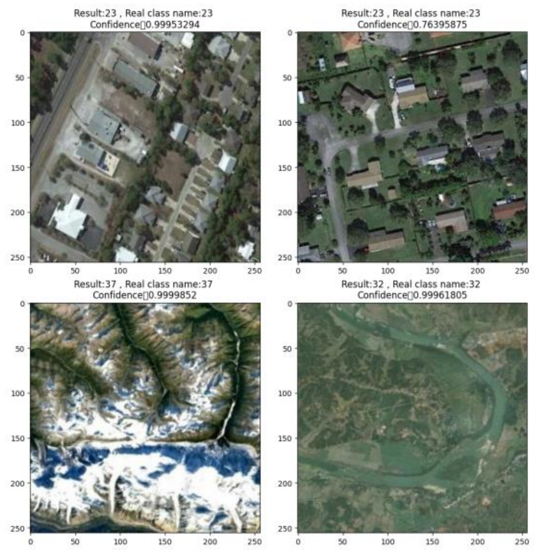
Test

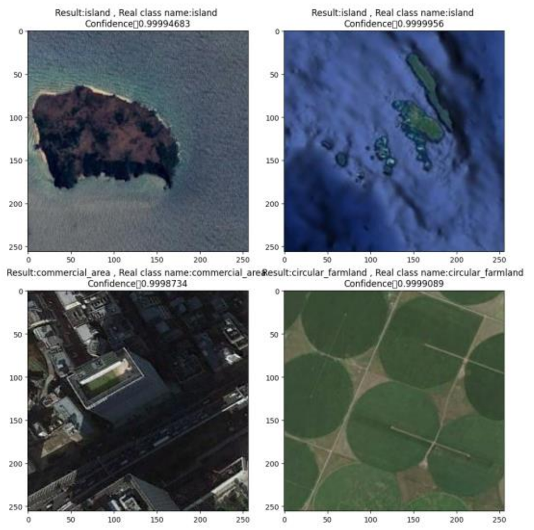
In [27]:
```python
import os
import numpy as np
import cv2
from keras import backend as K
from keras.models import load_model
from matplotlib import pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array, load_img

def img_test(img_path, model, labels):
    img_array = load_img(img_path, target_size=(256, 256))
    img_array = [img_to_array(img_array)]
    x_test = np.array(img_array, dtype='float') / 255.0
    test_pred = np.argmax(model.predict(x_test), axis=1)
    score = np.amax(model.predict(x_test), axis=1)
    plt.title('Result:%s \nConfidence: %s' % (labels[test_pred[0]], score[0]))
    plt.imshow(x_test[0])
    plt.show()


def random_img_test(model, labels, testset_dir):
    test_datagen = ImageDataGenerator(rescale=1. / 255)
    test_generator = test_datagen.flow_from_directory(
        testset_dir,
        target_size=(256, 256),
        batch_size=4,
        class_mode='categorical')
    x_test, y_test = test_generator.__getitem__(0)
    preds = model.predict(x_test)

    plt.figure(figsize=(10, 10))
    for i in range(4):
        plt.subplot(2, 2, i+1)
        plt.title('Result:%s , Real class name:%s \nConfidence: %s' % (labels[np.argmax(preds[i])], labels[np.argmax(y_test[:
        plt.tight_layout(pad=0.4, w_pad=0.6, h_pad=0.6)
        plt.imshow(x_test[i])
    plt.show()

```

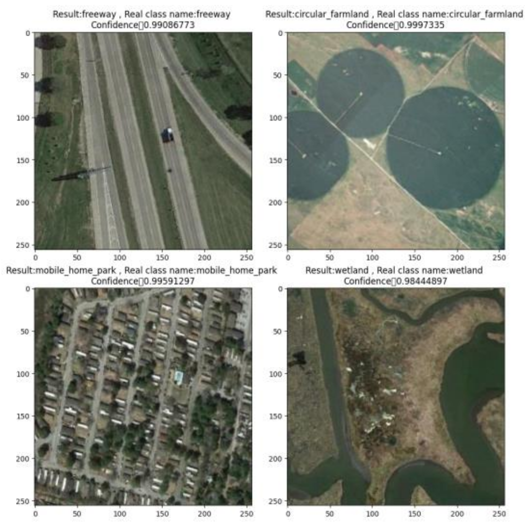
In [37]:
```python
testset_dir = 'data/test/NWPU45'
weight_path = 'result/model-weight-ep-625-val_loss-0.3092-val_acc-0.9551.h5'
model = LCNN_BFF(input_shape, num_classes)
model.load_weights(weight_path)

labels = [
    "airplane", "airport", "baseball_diamond", "basketball_court", "beach", "bridge", "chaparral", "church", "circular_farml
"cloud", "commercial_area", "dense_residential", "desert", "forest", "freeway", "golf_course", "ground_track_field", "harbor
"intersection", "island", "lake", "meadow", "medium_residential", "mobile_home_park", "mountain",  "overpass", "palace", "pa
"railway_station", "rectangular_farmland", "river", "roundabout", "runway", "sea_ice", "ship", "snowberg", "sparse_residenti
"storage_tank", "tennis_court", "terrace", "thermal_power_station", "wetland"
]

random_img_test(model, labels, testset_dir)
```

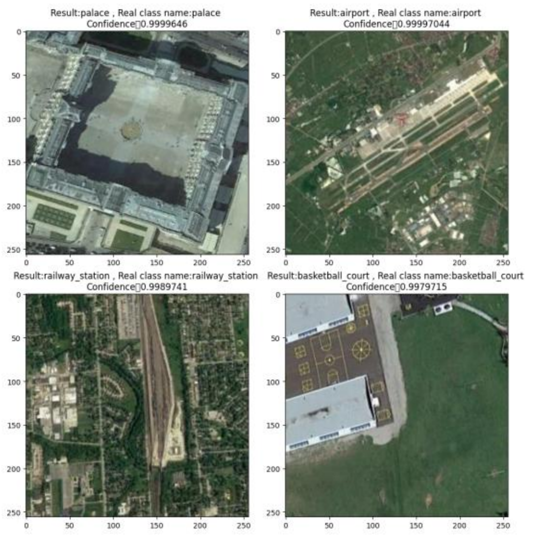
```
Found 6300 images belonging to 45 classes.
WARNING:tensorflow:5 out of the last 988 calls to <function Model.make_predict_function.<locals>.predict_function at 0x00000173
79860F70> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creati
ng @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can a
void unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and http
s://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 1s 553ms/step
```

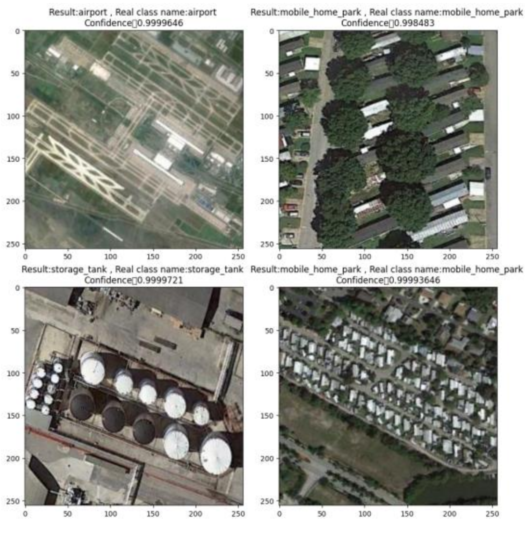
The result obtained contains the actual class, predicted class and confidence level for each image.

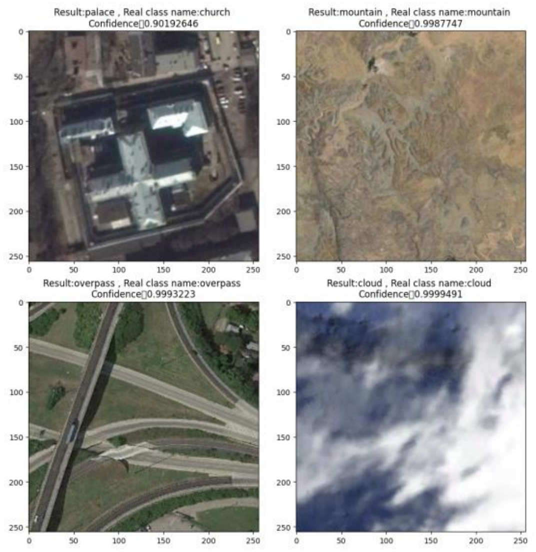

Result:23 , Real class name:23
Confidence□0.99953294
Result:23 , Real class name:23
Confidence□0.76395875
Result:37 , Real class name:37
Confidence□0.9999852
Result:32 , Real class name:32
Confidence□0.99961805

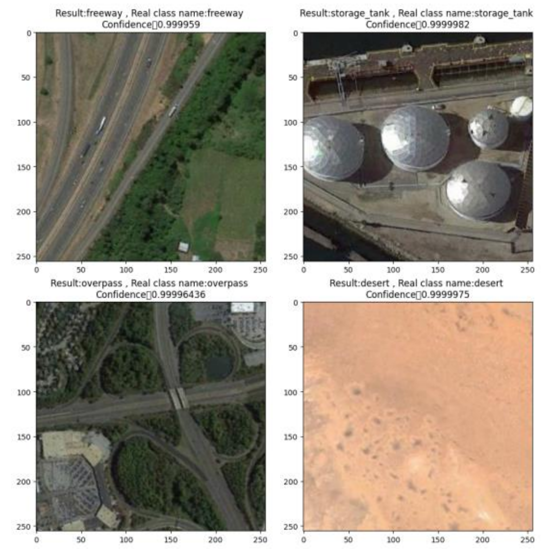

Result:island , Real class name:island
Confidence□0.99994683
Result:island , Real class name:island
Confidence□0.9999956
Result:commercial_area , Real class name:commercial_area
Confidence□0.9998734
Result:circular_farmland , Real class name:circular_farmland
Confidence□0.9999089

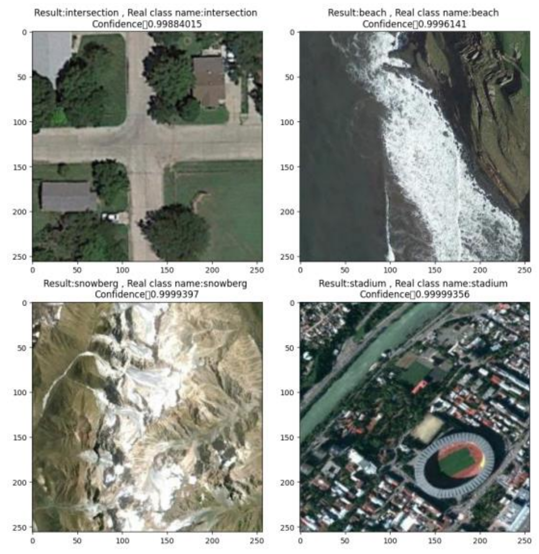

Result:freeway , Real class name:freeway
Confidence□0.99086773
Result:circular_farmland , Real class name:circular_farmland
Confidence□0.9997335
Result:mobile_home_park , Real class name:mobile_home_park
Confidence□0.99591297
Result:wetland , Real class name:wetland
Confidence□0.98444897

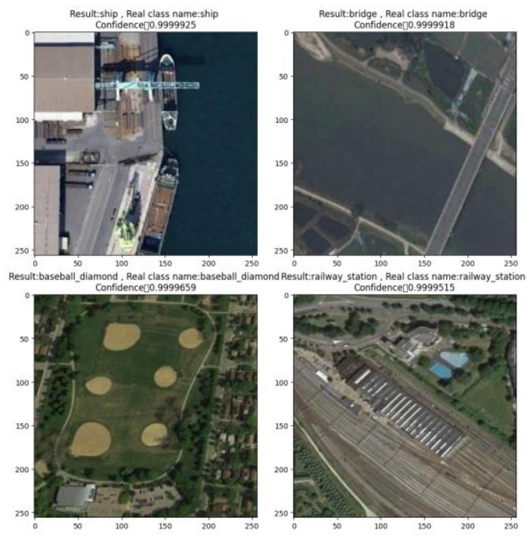

Result:palace , Real class name:palace
Confidence□0.9999646
Result:airport , Real class name:airport
Confidence□0.99997044
Result:railway_station , Real class name:railway_station
Confidence□0.9989741
Result:basketball_court , Real class name:basketball_court
Confidence□0.9979715

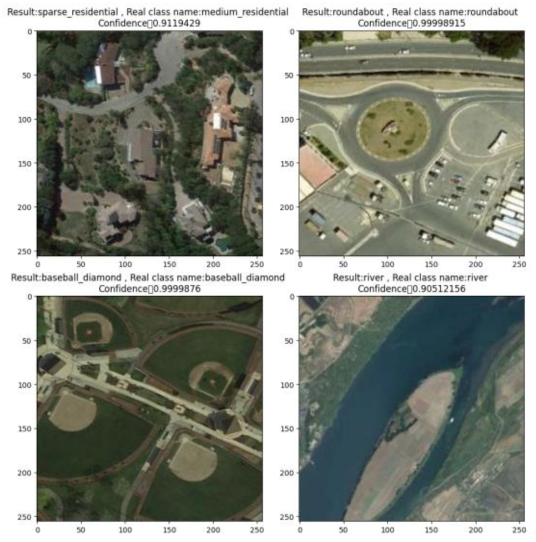

Result:airport , Real class name:airport
Confidence□0.9999646
Result:mobile_home_park , Real class name:mobile_home_park
Confidence□0.998483
Result:storage_tank , Real class name:storage_tank
Confidence□0.9999721
Result:mobile_home_park , Real class name:mobile_home_park
Confidence□0.99993646


Result:palace , Real class name:church
Confidence□0.90192646
Result:mountain , Real class name:mountain
Confidence□0.9987747
Result:overpass , Real class name:overpass
Confidence□0.9993223
Result:cloud , Real class name:cloud
Confidence□0.9999491


Result:freeway , Real class name:freeway
Confidence□0.999959
Result:storage_tank , Real class name:storage_tank
Confidence□0.9999982
Result:overpass , Real class name:overpass
Confidence□0.99996436
Result:desert , Real class name:desert
Confidence□0.9999975

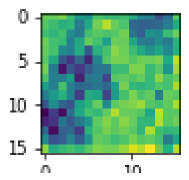

Result:intersection , Real class name:intersection
Confidence□0.99884015
Result:beach , Real class name:beach
Confidence□0.9996141
Result:snowberg , Real class name:snowberg
Confidence□0.9999397
Result:stadium , Real class name:stadium
Confidence□0.99999356


Result:ship , Real class name:ship
Confidence□0.9999925
Result:bridge , Real class name:bridge
Confidence□0.9999918
Result:baseball_diamond , Real class name:baseball_diamond
Confidence□0.9999659
Result:railway_station , Real class name:railway_station
Confidence□0.9999515

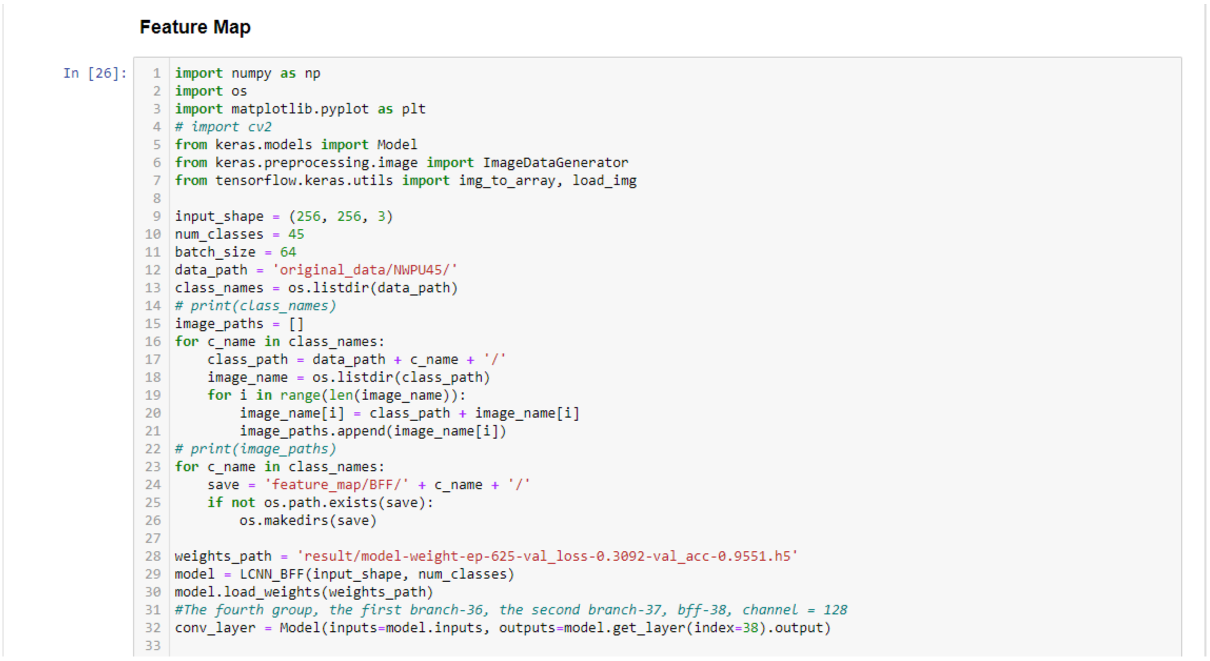

Result:sparse_residential , Real class name:medium_residential
Confidence□0.9119429
Result:roundabout , Real class name:roundabout
Confidence□0.99998915
Result:baseball_diamond , Real class name:baseball_diamond
Confidence□0.9999876
Result:river , Real class name:river
Confidence□0.90512156

RESULT ANALYSIS

FEATURE MAP GENERATION



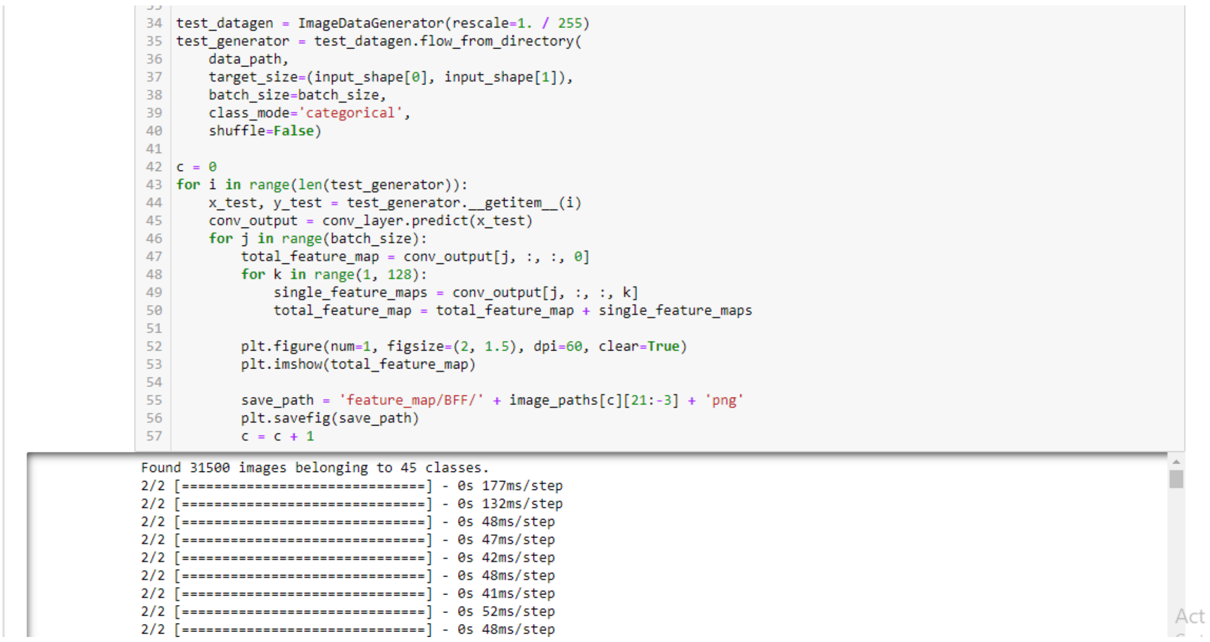
Feature map is the result of the convolution operation. Upon passing the input image over the convolution layers, the original image is converted into a feature image, which is then passed to a classification layer.



Feature Map

In [26]:

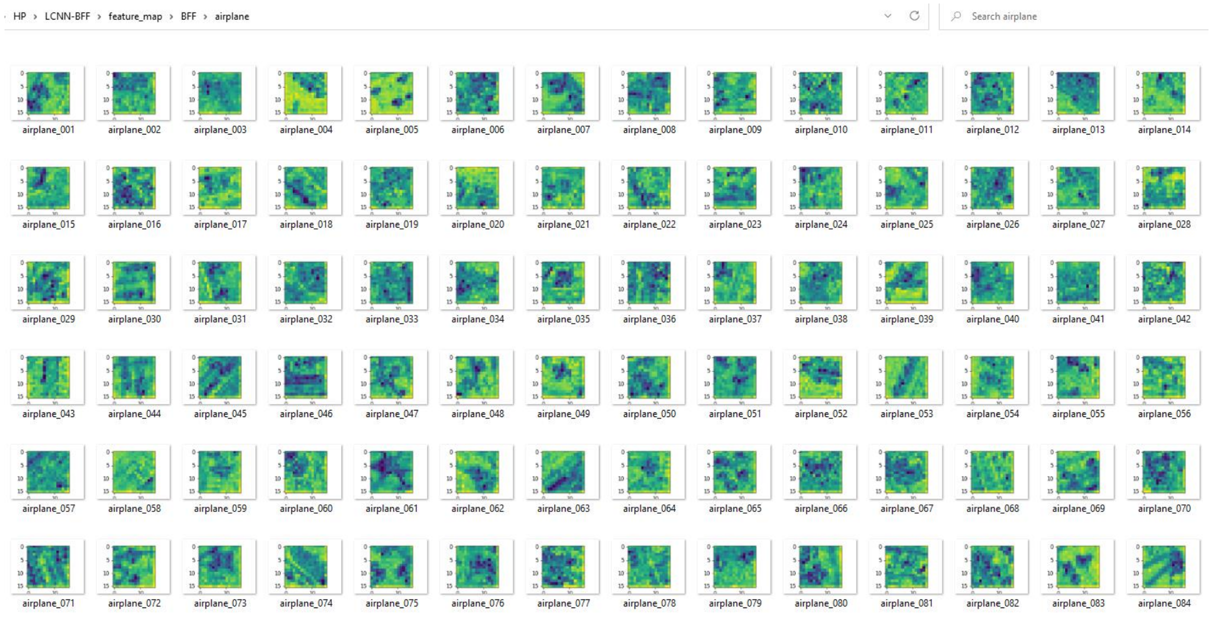
```
import numpy as np
import os
import matplotlib.pyplot as plt
# import cv2
from keras.models import Model
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array, load_img

input_shape = (256, 256, 3)
num_classes = 45
batch_size = 64
data_path = 'original_data/NWPU45/'
class_names = os.listdir(data_path)
# print(class_names)
image_paths = []
for c_name in class_names:
    class_path = data_path + c_name + '/'
    image_name = os.listdir(class_path)
    for i in range(len(image_name)):
        image_name[i] = class_path + image_name[i]
        image_paths.append(image_name[i])
# print(image_paths)
for c_name in class_names:
    save = 'feature_map/BFF/' + c_name + '/'
    if not os.path.exists(save):
        os.makedirs(save)

weights_path = 'result/model-weight-ep-625-val_loss-0.3092-val_acc-0.9551.h5'
model = LCNN_BFF(input_shape, num_classes)
model.load_weights(weights_path)
#The fourth group, the first branch-36, the second branch-37, bff-38, channel = 128
conv_layer = Model(inputs=model.inputs, outputs=model.get_layer(index=38).output)

```



```
test_datagen = ImageDataGenerator(rescale=1. / 255)
test_generator = test_datagen.flow_from_directory(
    data_path,
    target_size=(input_shape[0], input_shape[1]),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

c = 0
for i in range(len(test_generator)):
    x_test, y_test = test_generator.__getitem__(i)
    conv_output = conv_layer.predict(x_test)
    for j in range(batch_size):
        total_feature_map = conv_output[j, :, :, 0]
        for k in range(1, 128):
            single_feature_maps = conv_output[j, :, :, k]
            total_feature_map = total_feature_map + single_feature_maps

        plt.figure(num=1, figsize=(2, 1.5), dpi=60, clear=True)
        plt.imshow(total_feature_map)

        save_path = 'feature_map/BFF/' + image_paths[c][21:-3] + 'png'
        plt.savefig(save_path)
        c = c + 1
```

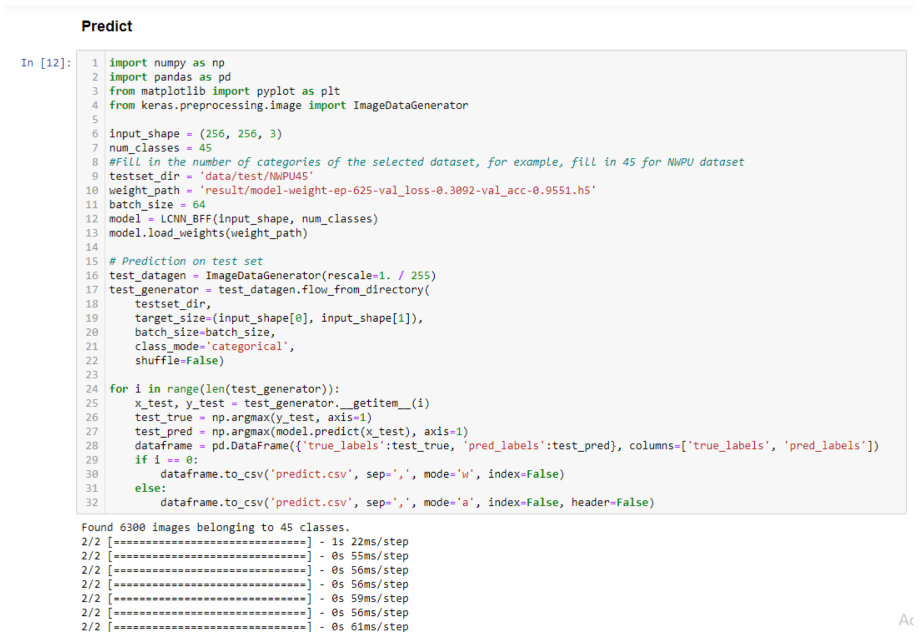
```
Found 31500 images belonging to 45 classes.
2/2 [==============================] - 0s 177ms/step
2/2 [==============================] - 0s 132ms/step
2/2 [==============================] - 0s 48ms/step
2/2 [==============================] - 0s 47ms/step
2/2 [==============================] - 0s 42ms/step
2/2 [==============================] - 0s 48ms/step
2/2 [==============================] - 0s 41ms/step
2/2 [==============================] - 0s 52ms/step
2/2 [==============================] - 0s 48ms/step
```

After training the network, feature map is generated for every image in the dataset. These are the feature maps.



PREDICTION AND CLASSIFICATION

Random samples are taken from the test set and passed to the model for prediction and stored in **predict.csv** file.

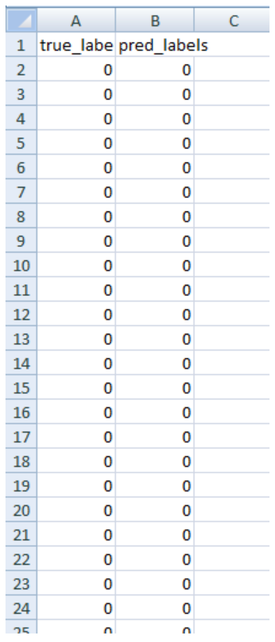


Predict

In [12]:
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from keras.preprocessing.image import ImageDataGenerator

input_shape = (256, 256, 3)
num_classes = 45
#Fill in the number of categories of the selected dataset, for example, fill in 45 for NWPU dataset
testset_dir = 'data/test/NWPU45'
weight_path = 'result/model-weight-ep-625-val_loss-0.3092-val_acc-0.9551.h5'
batch_size = 64
model = LCNN_BFF(input_shape, num_classes)
model.load_weights(weight_path)

# Prediction on test set
test_datagen = ImageDataGenerator(rescale=1. / 255)
test_generator = test_datagen.flow_from_directory(
    testset_dir,
    target_size=(input_shape[0], input_shape[1]),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

for i in range(len(test_generator)):
    x_test, y_test = test_generator.__getitem__(i)
    test_true = np.argmax(y_test, axis=1)
    test_pred = np.argmax(model.predict(x_test), axis=1)
    dataframe = pd.DataFrame({'true_labels':test_true, 'pred_labels':test_pred}, columns=['true_labels', 'pred_labels'])
    if i == 0:
        dataframe.to_csv('predict.csv', sep=',', mode='w', index=False)
    else:
        dataframe.to_csv('predict.csv', sep=',', mode='a', index=False, header=False)
```

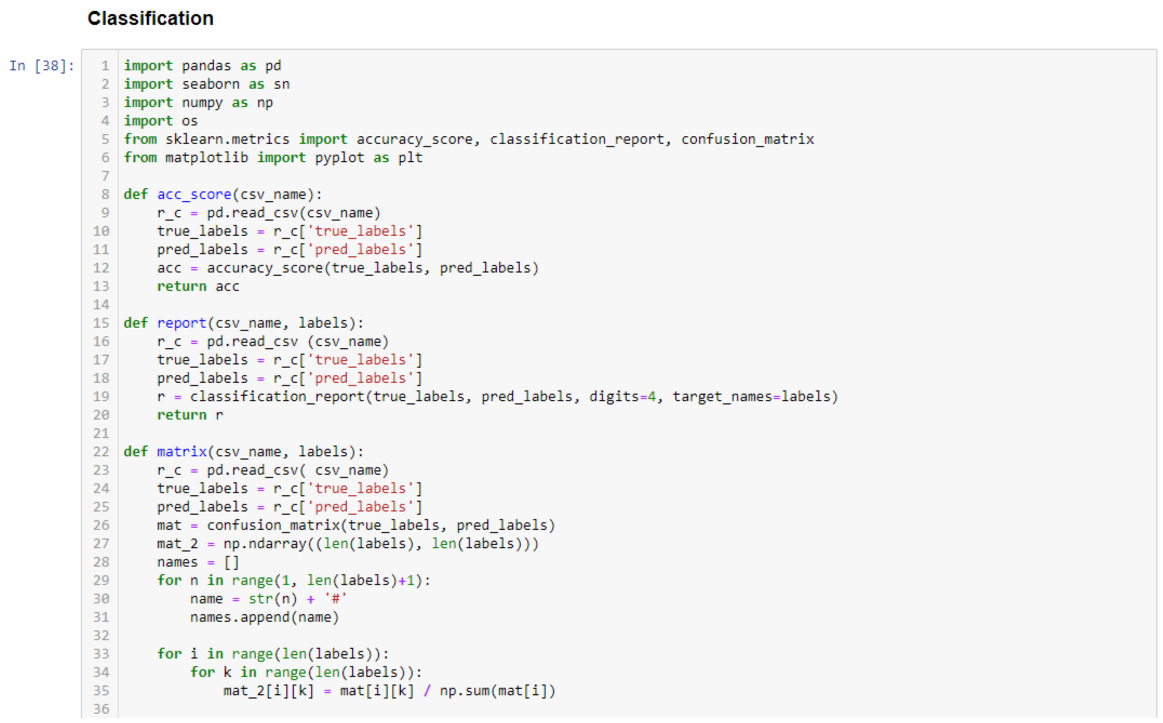
```
Found 6300 images belonging to 45 classes.
2/2 [==============================] - 1s 22ms/step
2/2 [==============================] - 0s 55ms/step
2/2 [==============================] - 0s 56ms/step
2/2 [==============================] - 0s 56ms/step
2/2 [==============================] - 0s 59ms/step
2/2 [==============================] - 0s 56ms/step
2/2 [==============================] - 0s 61ms/step
```

predict.csv



	A	B	C
1	true_labe	pred_labels	
2	0	0	
3	0	0	
4	0	0	
5	0	0	
6	0	0	
7	0	0	
8	0	0	
9	0	0	
10	0	0	
11	0	0	
12	0	0	
13	0	0	
14	0	0	
15	0	0	
16	0	0	
17	0	0	
18	0	0	
19	0	0	
20	0	0	
21	0	0	
22	0	0	
23	0	0	
24	0	0	

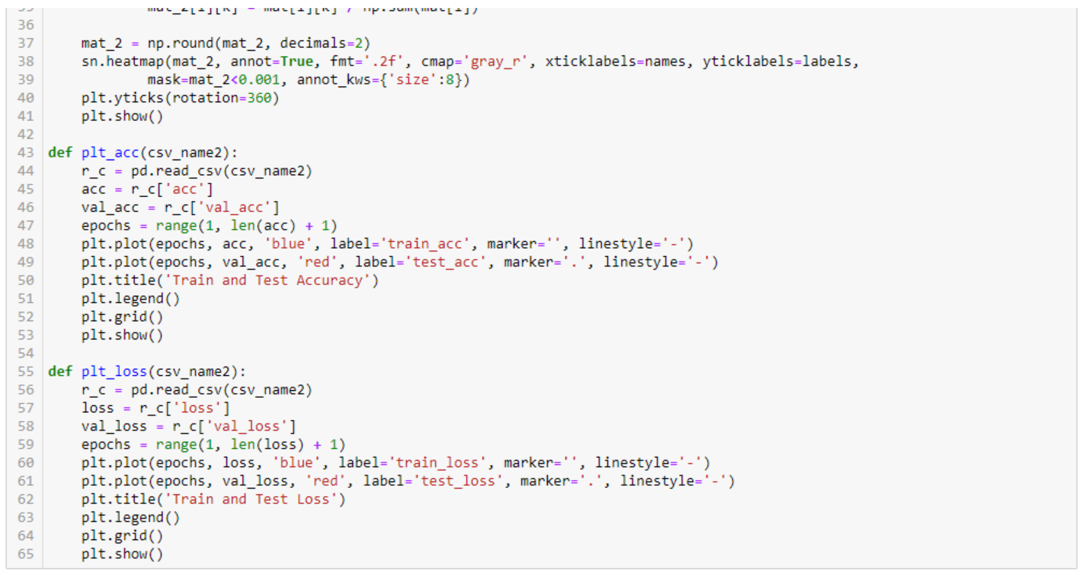
We compute classification metrics: accuracy score, confusion matrix, plot accuracy and plot loss.



Classification

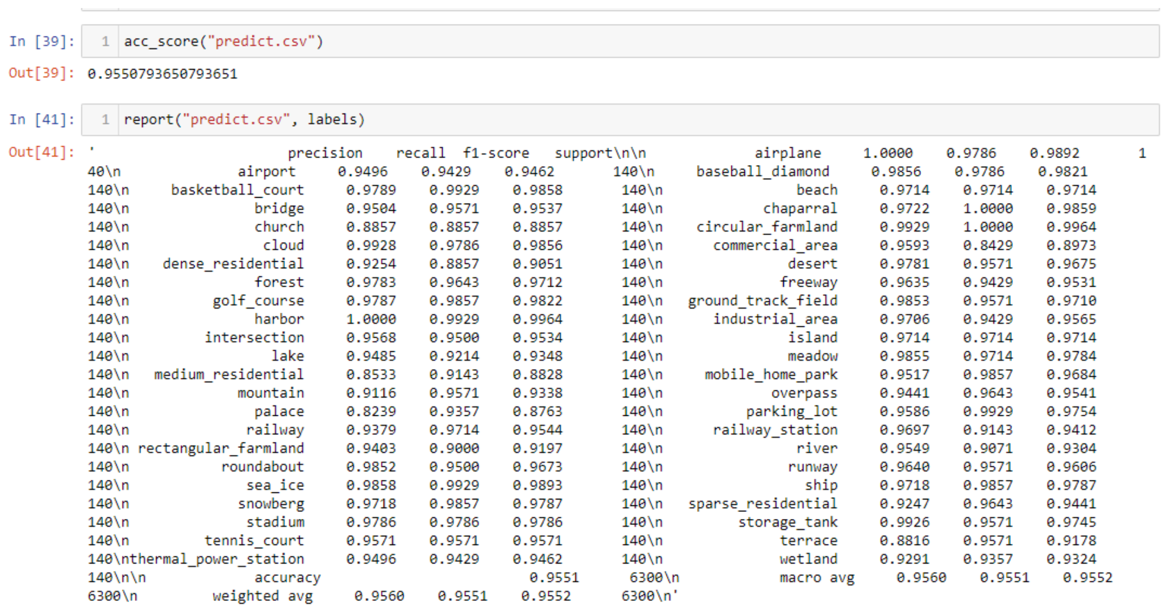
In [38]:

```python
import pandas as pd
import seaborn as sn
import numpy as np
import os
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from matplotlib import pyplot as plt

def acc_score(csv_name):
    r_c = pd.read_csv(csv_name)
    true_labels = r_c['true_labels']
    pred_labels = r_c['pred_labels']
    acc = accuracy_score(true_labels, pred_labels)
    return acc

def report(csv_name, labels):
    r_c = pd.read_csv (csv_name)
    true_labels = r_c['true_labels']
    pred_labels = r_c['pred_labels']
    r = classification_report(true_labels, pred_labels, digits=4, target_names=labels)
    return r

def matrix(csv_name, labels):
    r_c = pd.read_csv( csv_name)
    true_labels = r_c['true_labels']
    pred_labels = r_c['pred_labels']
    mat = confusion_matrix(true_labels, pred_labels)
    mat_2 = np.ndarray((len(labels), len(labels)))
    names = []
    for n in range(1, len(labels)+1):
        name = str(n) + '#'
        names.append(name)

    for i in range(len(labels)):
        for k in range(len(labels)):
            mat_2[i][k] = mat[i][k] / np.sum(mat[i])

```



```python

    mat_2 = np.round(mat_2, decimals=2)
    sn.heatmap(mat_2, annot=True, fmt='.2f', cmap='gray_r', xticklabels=names, yticklabels=labels,
            mask=mat_2<0.001, annot_kws={'size':8})
    plt.yticks(rotation=360)
    plt.show()

def plt_acc(csv_name2):
    r_c = pd.read_csv(csv_name2)
    acc = r_c['acc']
    val_acc = r_c['val_acc']
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'blue', label='train_acc', marker='', linestyle='-')
    plt.plot(epochs, val_acc, 'red', label='test_acc', marker='.', linestyle='-')
    plt.title('Train and Test Accuracy')
    plt.legend()
    plt.grid()
    plt.show()

def plt_loss(csv_name2):
    r_c = pd.read_csv(csv_name2)
    loss = r_c['loss']
    val_loss = r_c['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'blue', label='train_loss', marker='', linestyle='-')
    plt.plot(epochs, val_loss, 'red', label='test_loss', marker='.', linestyle='-')
    plt.title('Train and Test Loss')
    plt.legend()
    plt.grid()
    plt.show()
```

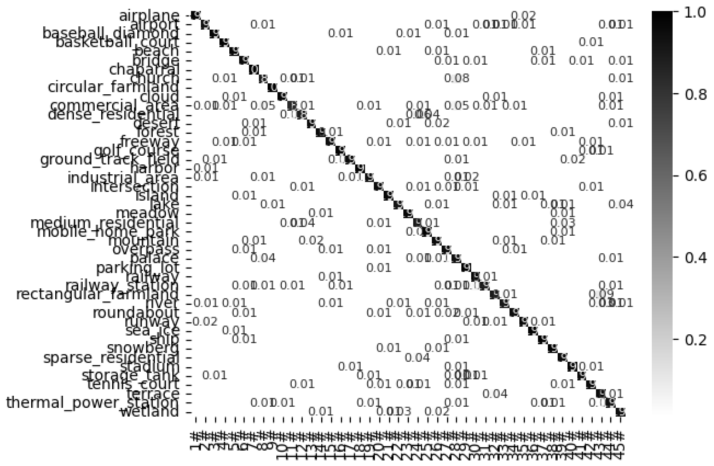
- **Accuracy score:** Accuracy score is used to measure the model performance in terms of measuring the ratio of sum of true positive and true negatives out of all the predictions made.



```
In [39]:  acc_score("predict.csv")
Out[39]: 0.9550793650793651

In [41]:  report("predict.csv", labels)
Out[41]: '                        precision    recall  f1-score   support\n\n               airplane     1.0000    0.9786    0.9892       1
         40\n            airport     0.9496    0.9429    0.9462       140\n    baseball_diamond     0.9856    0.9786    0.9821
         140\n    basketball_court     0.9789    0.9929    0.9858       140\n               beach     0.9714    0.9714    0.9714
         140\n              bridge     0.9504    0.9571    0.9537       140\n           chaparral     0.9722    1.0000    0.9859
         140\n              church     0.8857    0.8857    0.8857       140\n   circular_farmland     0.9929    1.0000    0.9964
         140\n               cloud     0.9928    0.9786    0.9856       140\n     commercial_area     0.9593    0.8429    0.8973
         140\n   dense_residential     0.9254    0.8857    0.9051       140\n              desert     0.9781    0.9571    0.9675
         140\n              forest     0.9783    0.9643    0.9712       140\n             freeway     0.9635    0.9429    0.9531
         140\n         golf_course     0.9787    0.9857    0.9822       140\n  ground_track_field     0.9853    0.9571    0.9710
         140\n              harbor     1.0000    0.9929    0.9964       140\n     industrial_area     0.9706    0.9429    0.9565
         140\n        intersection     0.9568    0.9500    0.9534       140\n              island     0.9714    0.9714    0.9714
         140\n                lake     0.9485    0.9214    0.9348       140\n              meadow     0.9855    0.9714    0.9784
         140\n  medium_residential     0.8533    0.9143    0.8828       140\n     mobile_home_park     0.9517    0.9857    0.9684
         140\n            mountain     0.9116    0.9571    0.9338       140\n            overpass     0.9441    0.9643    0.9541
         140\n              palace     0.8239    0.9357    0.8763       140\n         parking_lot     0.9586    0.9929    0.9754
         140\n             railway     0.9379    0.9714    0.9544       140\n     railway_station     0.9697    0.9143    0.9412
         140\n rectangular_farmland     0.9403    0.9000    0.9197       140\n               river     0.9549    0.9071    0.9304
         140\n          roundabout     0.9852    0.9500    0.9673       140\n              runway     0.9640    0.9571    0.9606
         140\n             sea_ice     0.9858    0.9929    0.9893       140\n                ship     0.9718    0.9857    0.9787
         140\n            snowberg     0.9718    0.9857    0.9787       140\n  sparse_residential     0.9247    0.9643    0.9441
         140\n             stadium     0.9786    0.9786    0.9786       140\n        storage_tank     0.9926    0.9571    0.9745
         140\n        tennis_court     0.9571    0.9571    0.9571       140\n             terrace     0.8816    0.9571    0.9178
         140\nthermal_power_station     0.9496    0.9429    0.9462       140\n             wetland     0.9291    0.9357    0.9324
         140\n\n             accuracy                         0.9551      6300\n            macro avg     0.9560    0.9551    0.9552
         6300\n         weighted avg     0.9560    0.9551    0.9552      6300\n'
```

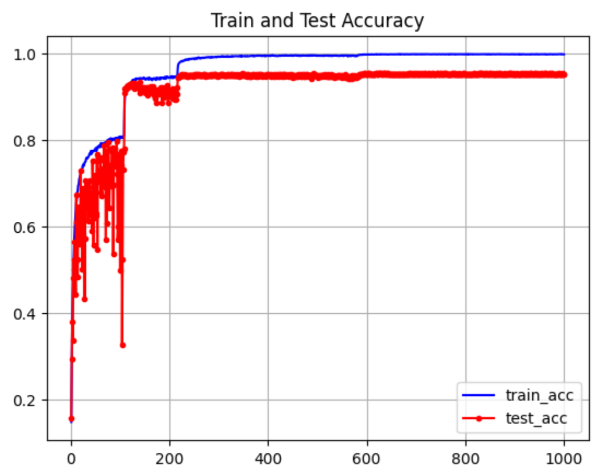
- **Confusion matrix:** A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm.

```
In [42]:  matrix("predict.csv", labels)
```



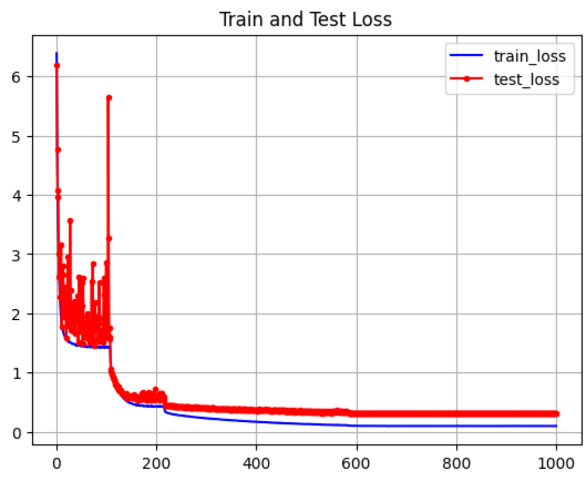
- **Plot accuracy:** We pick up the training data accuracy (“acc”) and the validation data accuracy (“val_acc”) for plotting. As you can see in the diagram, the accuracy increases rapidly in the first 100 epochs, indicating that the network is learning fast.

```
In [44]: plt_acc("./result/XXX.csv")
```



- **Plot loss:** Plots the loss function of an object containing the results of a gradient descent object implementation. To prevent overfitting, we can make use of the loaded function plot_loss() to plot training loss against validation loss.

```
In [45]: plt_loss("./result/XXX.csv")
```



****