**Machine Learning - Project Report**

**Project Task**

Scene-Centric Image Recognition Task with SUN397 dataset

**Group Member**

Xiang Wang (1106943), Ao Chen (1107150)

**Introduction**

In this project, we used the SUN397 dataset, which is the scene image dataset. The SUN dataset is a scene understanding dataset containing 899 categories and 130,519 images. Publishers use 397 well-sampled categories to evaluate the many state-of-the-art algorithms used for scene recognition and establish new performance boundaries. The dataset was released in 2014 by the Princeton School of Visual and Robotics. We used a variety of deep learning network structures to test this data set. This includes VGG16 and DenseNet, DenseNet201. We use the Colab platform to read data and then use the Google cloud platform. To train the network, use different weights to detect the accuracy of the network through various attempts, and finally we got the result.

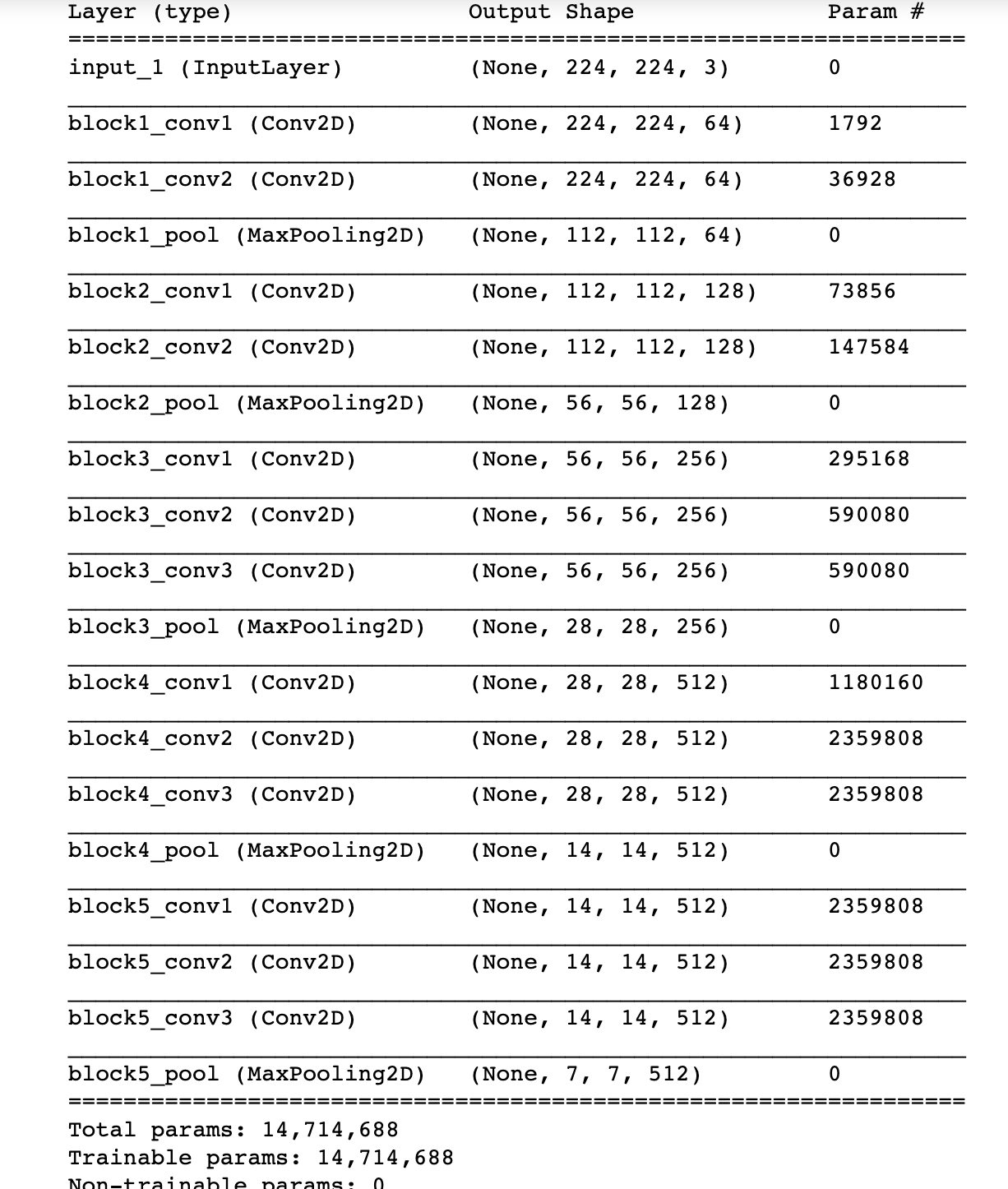
**Methodology**

Neural Networks

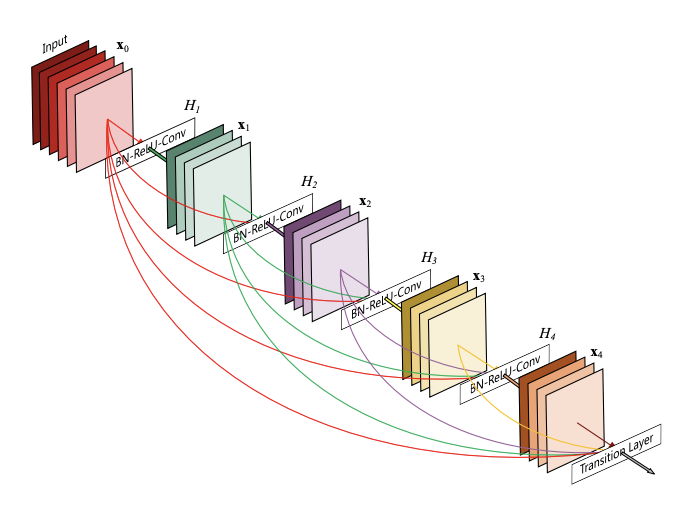
First we used the VGG16 network model. VGG16 contains 16 hidden layers (13 convolution layers and 3 fully connected layers). VGG16 is an improvement over AlexNet. It uses several consecutive 3x3 convolution kernels instead of AlexNet. Larger convolution kernel (11x11, 7x7, 5x5). For a given receptive field (the local size of the input picture associated with the output), the use of stacked small convolution kernels is superior to the use of large convolution kernels, because multiple layers of nonlinear layers can increase network depth to ensure more complex learning. Mode, and the cost is still relatively small (less parameters), after which we used DenseNet201, DenseNet-201 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 201 layers deep and can Classify images into 1000 object categories, such as a keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by- 224



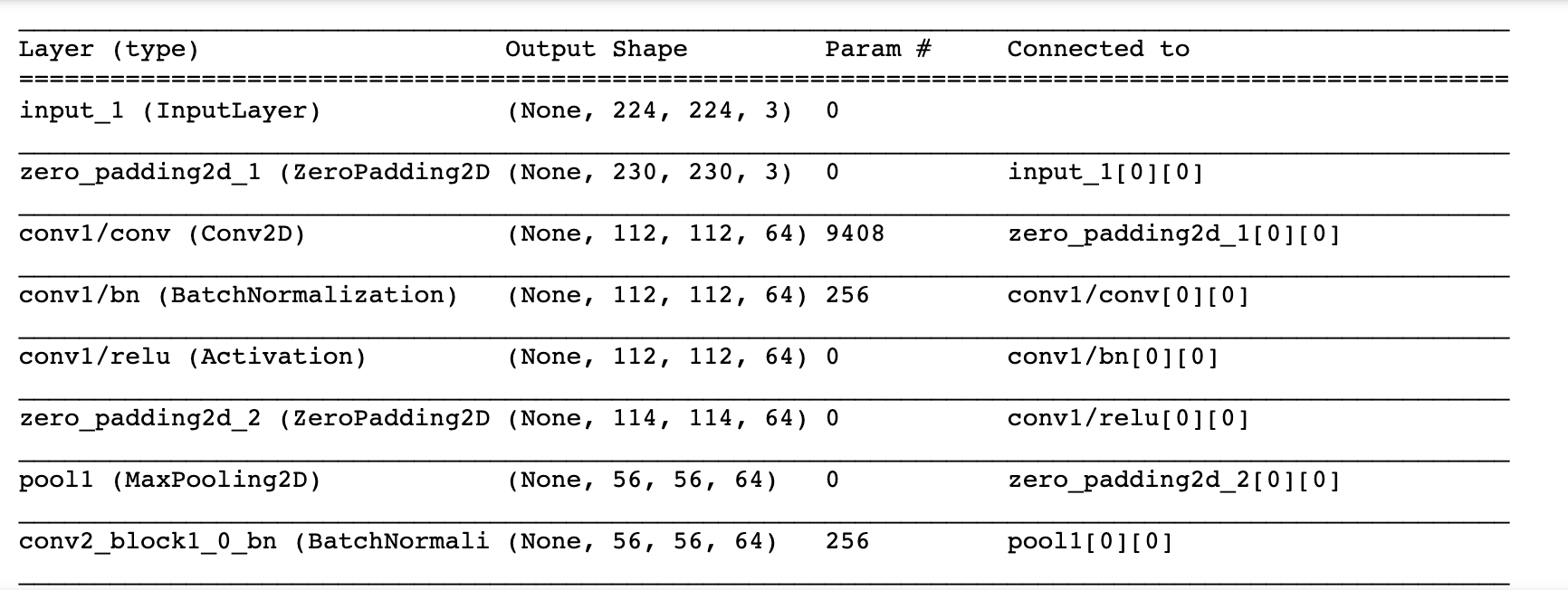
The structure of VGG16

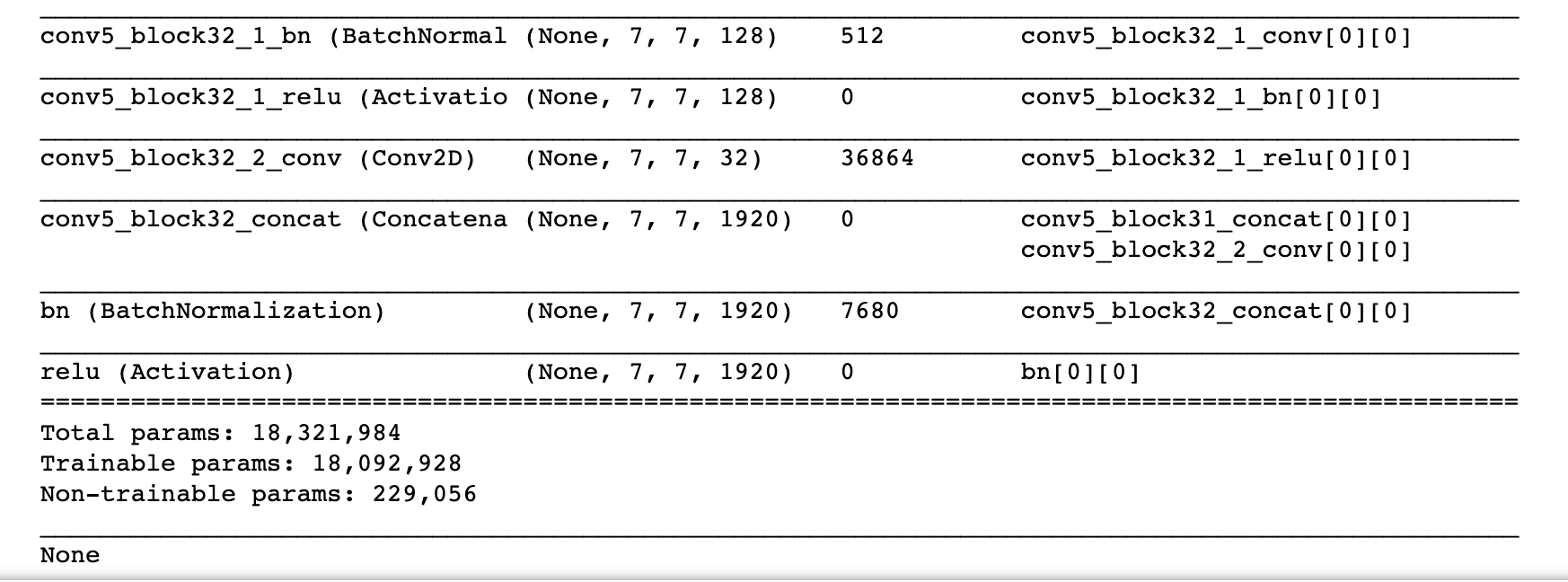


DenseNet-201 is a convolutional neural network that is trained on more than a million images from the ImageNet database . The network is 201 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224



The structure of DenseNet 201





**Experiment**

DataSet

The dataset we used is SUN397. SUN397 is a scene image dataset. It contains 397 classes. Scene categorization is a fundamental problem in computer vision. However, scene understanding research has been constrained by the limited scope of currently-used databases which do not capture the full variety of scene categories. Whereas standard databases for object categorization contain hundreds of different classes of objects, the largest available dataset of scene categories contains only 15 classes. Scene UNderstanding (SUN) database that contains 899 categories and 130,519 images. We use 397 well-sampled categories to evaluate numerous state-of-the-art algorithms for scene recognition and establish new bounds of performance. We measure human scene classification performance on the SUN database and compare this with computational methods. we use a subset of the dataset that has 50 training images and 50 testing images per class. We devide the dataset into training set and testing set by ourselves. Training set involves 397 classes, each class has 50 images which are randomly picked. Testing set involves 397 classes, each class has 50 images which are also randomly picked and different from training set. So both training set and testing set has 397 classes and 19850 images. In order to load data more conveniently, we save all the training data and testing data in 4 npy files. They include the image data for training set ‘x\_train.npy’ and testing set ‘x\_test.npy’, the one-hot matrix for label in training set ‘y\_train.npy’ and testing set ‘y-test.npy’.

Environment:

We run the project on virtual machine on Google Cloud. The hardware environment: The CPU has 8 cores; The memory is 100GB; GPU is Tesla P100; The operation system is Ubuntu. The software environment: Python 3.6; Keras 2.2.4; Tensorflow-gpu 1.10; Cuda 9.0; Cudnn 7.0.

Setting:

We have tried 3 methods which are VGG16, Densenet121 and Densenet201. For all the methods, we set the same learning rate: 1e-4 within 10 epochs, 1e-5 between 10 to 20 epochs, 1e-6 between 20 -30 epochs, 1e-7 for more than 30 epochs. We use the pretrained model of place365 in VGG16, and use the pretrained model of ImageNet in DenseNet.The loss function for VGG16 is mean square error, and categorical crossentropy for Densenet. We change a little network structure in VGG16 and Densenet. For VGG16, we didn’t use all the layers in Keras model. We add a flatten layer after convolution network to flatten the 2d data to a tebular data. Then the tebular data go through a fully connected layer with 4096 neurons. After the we add another fully connected layer and a dropout layer which has a 0.3 dropout rate. We add a softmax layer after dropout layer to predict the label. For the 2 Densenet experiments, we add a average pooling layer after convolutional network. After average pooling is a softmax layer to predict the label.

**Source Code**

**Data Processing**

nb\_classes = 397

img\_depth = 3

data\_dir ='/content/drive/My Drive/Colab Notebooks/SUN\_Practice/'+'SUN397'

train\_img\_file = '/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/Training\_01.txt' # Training\_01.txt

test\_img\_file = '/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/Testing\_01.txt' # Testing\_01.txt

classes\_name\_list = '/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/ClassName.txt'

train\_label\_file ='/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/ClassName.txt'

test\_label\_file = '/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/ClassName.txt'

train\_img\_file\_path='/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/Training\_01.txt'

test\_img\_file\_path='/content/drive/My Drive/Colab Notebooks/SUN\_Practice/Partitions/Testing\_01.txt'

start\_time\_ = time.time()

train\_img\_file\_path = [str(line.strip()) for line in open(train\_img\_file\_path).readlines()] # read all the image file name

nb\_sample = len(train\_img\_file\_path)

print('Image count: %d' % nb\_sample)

data\_resized\_holder = np.empty([nb\_sample, desired\_img\_dim, desired\_img\_dim, img\_depth], dtype='float32')

for idx in range(nb\_sample):

img\_file1 = train\_img\_file\_path[idx].replace("\\", "/") # the image file path

# print(str(img\_file1))

# 1. read the image

img1 = image.load\_img(data\_dir+img\_file1)

# 2. resize

img1 = img1.resize((desired\_img\_dim, desired\_img\_dim), resample=0)

# 6. give to the holder

data\_resized\_holder[idx] = img1

if(idx % 1000==0):

print('%d image loaded.' % idx)

print('\nImage file loaded, the shape is ' + str(data\_resized\_holder.shape))

one\_hot = True

print('Loading label file %s' % train\_label\_file)

label\_str = [str(line.strip()) for line in open(train\_label\_file).readlines()]

nb\_unique = len(label\_str)

labels\_unique = le.transform(label\_str)

# print(labels\_unique)

labels\_holder = np.hstack(( [ labels\_unique[i] ] \* 50 for i in range(nb\_unique)))

# print(labels\_holder)

nb\_sample = len(labels\_holder)

if one\_hot == True:

labels = np.array([[float(i == l) for i in range(nb\_classes)] for l in labels\_holder])

else:

labels = labels\_holder

print('Labels loaded, shape is:' + str(labels.shape))

**DenseNet 201 test this dataset**

import os

os.environ["CUDA\_VISIBLE\_DEVICES"]="0";

import keras

from keras.applications.vgg16 import VGG16

from keras.applications.densenet import DenseNet201

import numpy as np

from keras.applications.densenet import preprocess\_input

from keras.layers import Flatten, Dense, Dropout

from keras.models import Model

from keras.layers.pooling import GlobalAveragePooling2D

from keras import optimizers

from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

x\_train=np.load("x\_train.npy")

x\_test=np.load("x\_test.npy")

y\_train=np.load("y\_train.npy")

y\_test=np.load("y\_test.npy")

x\_test = preprocess\_input(x\_test)

x\_train = preprocess\_input(x\_train)

model = DenseNet201(weights="imagenet", include\_top=False, classes=397, input\_shape=(224,224,3))

print(model.summary())

x = model.get\_layer('relu').output

x = GlobalAveragePooling2D(name='pool')(x)

x = Dense(397, activation='softmax', name='fc1')(x)

model\_updated = Model(inputs=model.input, outputs=x)

def learning\_rate\_schedule(epoch):

if epoch <= 10:

return 1e-4 # 0.00001

elif epoch <= 20:

return 1e-5

elif epoch <= 30:

return 1e-6

else:

return 1e-7

return LR

model\_updated.load\_weights('model\_initial.h5')

training\_runs = []

for i in range(3):

model\_updated.compile(loss='categorical\_crossentropy', optimizer=optimizers.adam(lr=0.0001), metrics=['accuracy'])

keras.callbacks.LearningRateScheduler(learning\_rate\_schedule)

history = model\_updated.fit(x\_train, y\_train, batch\_size=32, shuffle=True, epochs=7, validation\_data=(x\_test, y\_test))

training\_runs.append(history)

model\_updated.get\_weights()

if i == 2:

model\_updated.save\_weights('model1\_from\_scratch.h5')

else:

model\_updated.load\_weights('model\_initial.h5')

print("Average training accuracy: {}".format(np.mean([training\_runs[0].history['acc'][-1],

training\_runs[1].history['acc'][-1], training\_runs[2].history['acc'][-1]])))

print("Average testing accuracy: {}".format(np.mean([training\_runs[0].history['val\_acc'][-1],

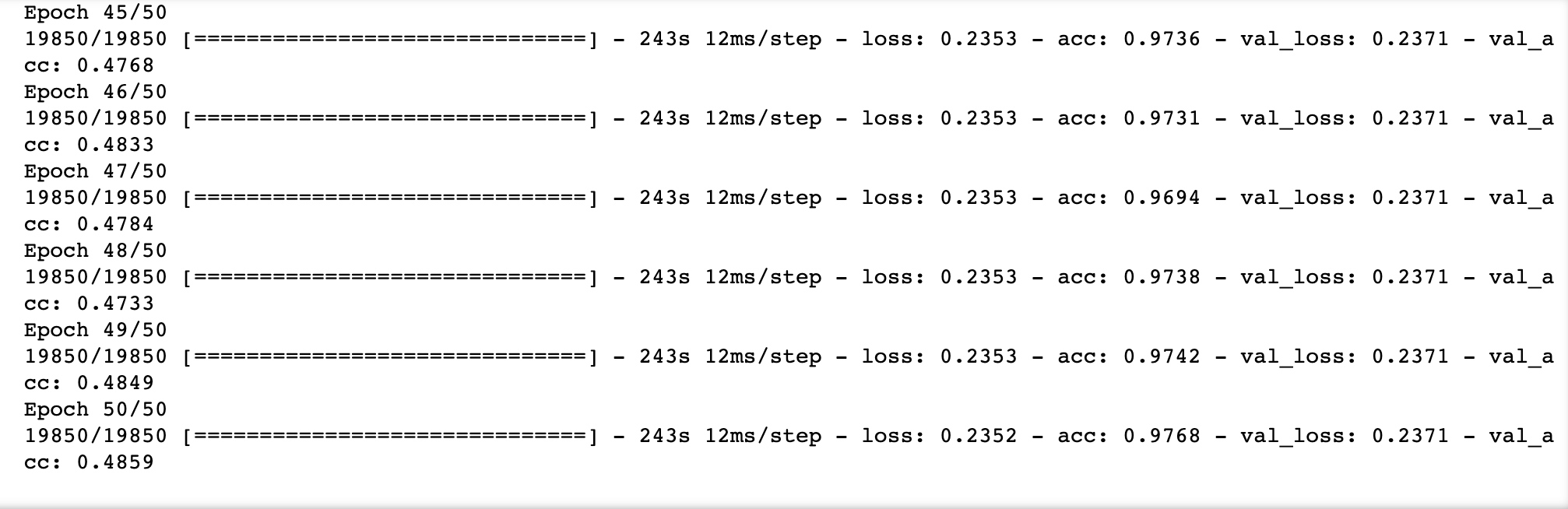
training\_runs[1].history['val\_acc'][-1], training\_runs[2].history['val\_acc'][-1]])))

**Result**

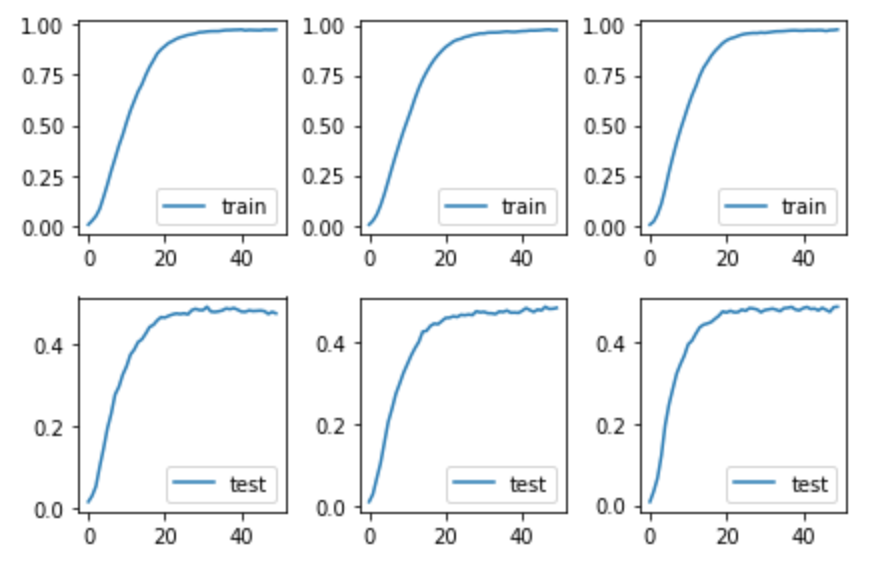
We carried out 3 experiments based on VGG16, DenseNet121 and DenseNet 201.The top-1 accuracy is on DenseNet201. We will mention the other 2 experiments. We get accuracy in VGG16. We train the network for epochs.

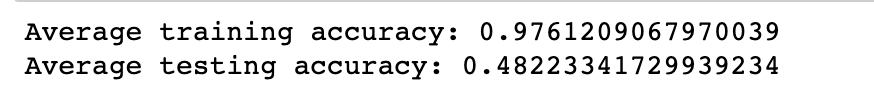
**VGG16 with places365 weights**

We get 48.22% accuracy in VGG16 with places365 weights. We have train the network for 50 epochs. We have found that the accuracy and loss become stable in 37 epochs.



And we get the train accuracy and test accuracy diagram





**DensNet121**

We get 55.80% accuracy in DensNet121. We have train the network for 20 epochs. We have found that the accuracy and loss become stable in 11 epochs.



**DenseNet 201**

In the end, we got the best accuracy in DenseNet201. We only train the network for 7 epochs and the loss and accuracy become stable. We repeat the experiment for 3 times and the performances are very solid. The average testing accuracy reaches 59.37%.

First time:



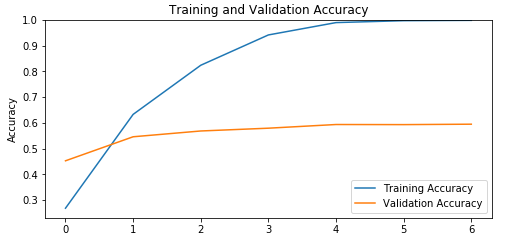
Second time:



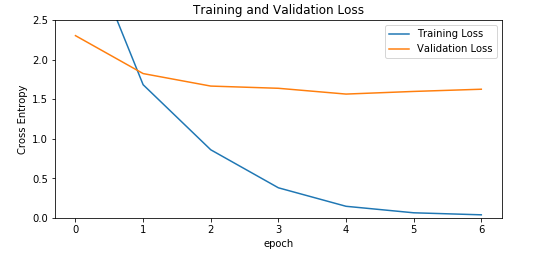
Third time:

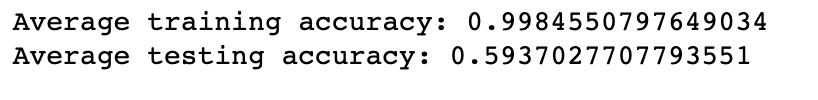


The average accuracy in training and testing:



The loss in training and testing:





**Conclusion**

In this project, we selected the scene test set Sun397. By selecting 50 sheets as training data and 50 sheets as testing data in each class, we built training dataset and testing dataset. The purpose of this is to make it better. Training network. We use a variety of network models for testing, such as: VGG16 based on imagenet weights and VGG16 based on places365 weights. and DenseNet121 and more layers of DenseNet201 model. Because the size of this data set is 39GB, ordinary machines can't complete this task. We selected Colab and Google Cloud Platform for experiment. Finally, we completed the experiment and achieved the best accuracy with DenseNet 201: 59.37%.