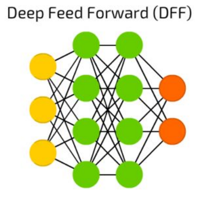
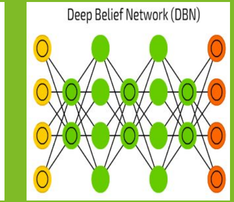
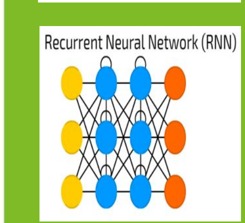
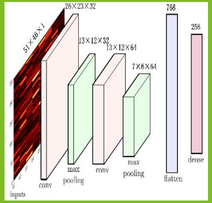
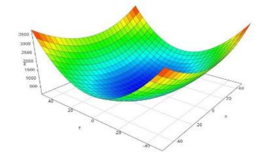
Deep Learning CSI_7_DEL

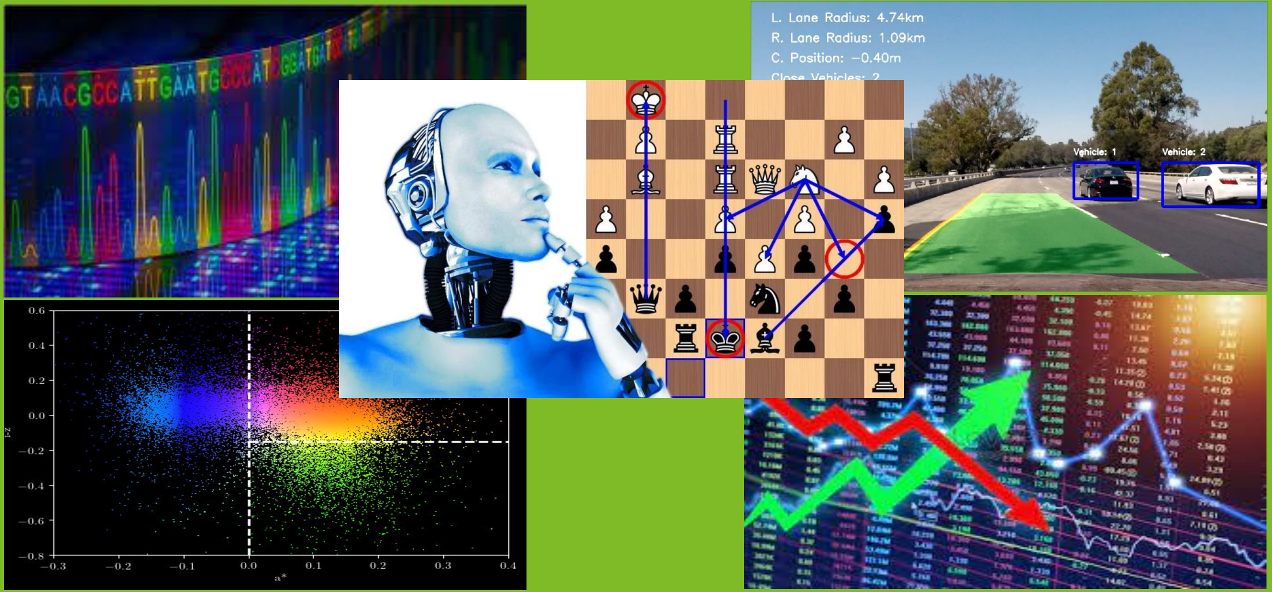








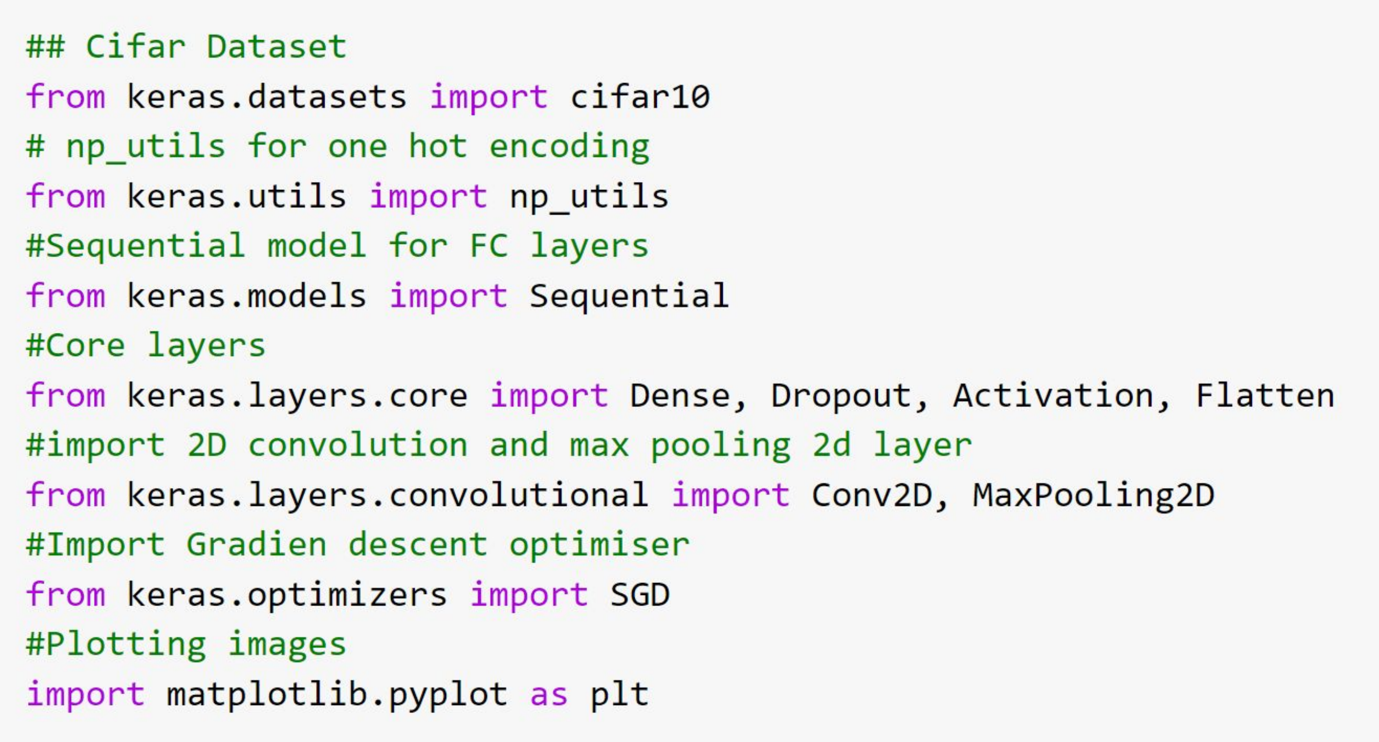




Tutorial 7:Convolutional Neural Networks (CNN)

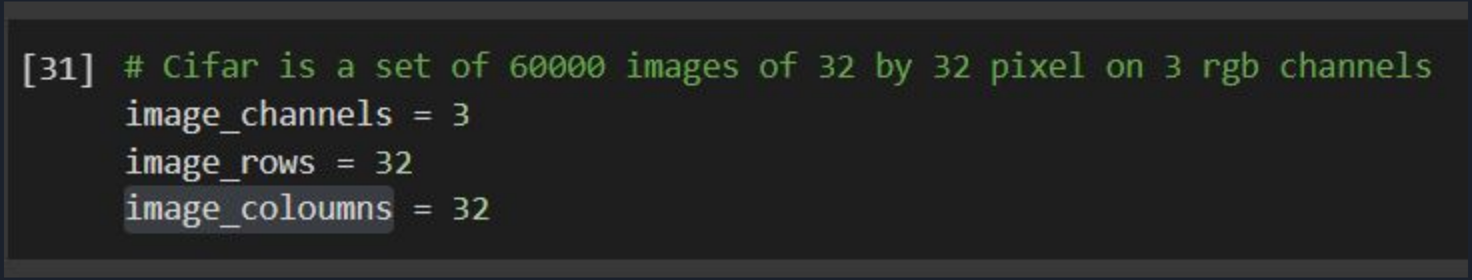
London South Bank University
EST 1892

Import libraries



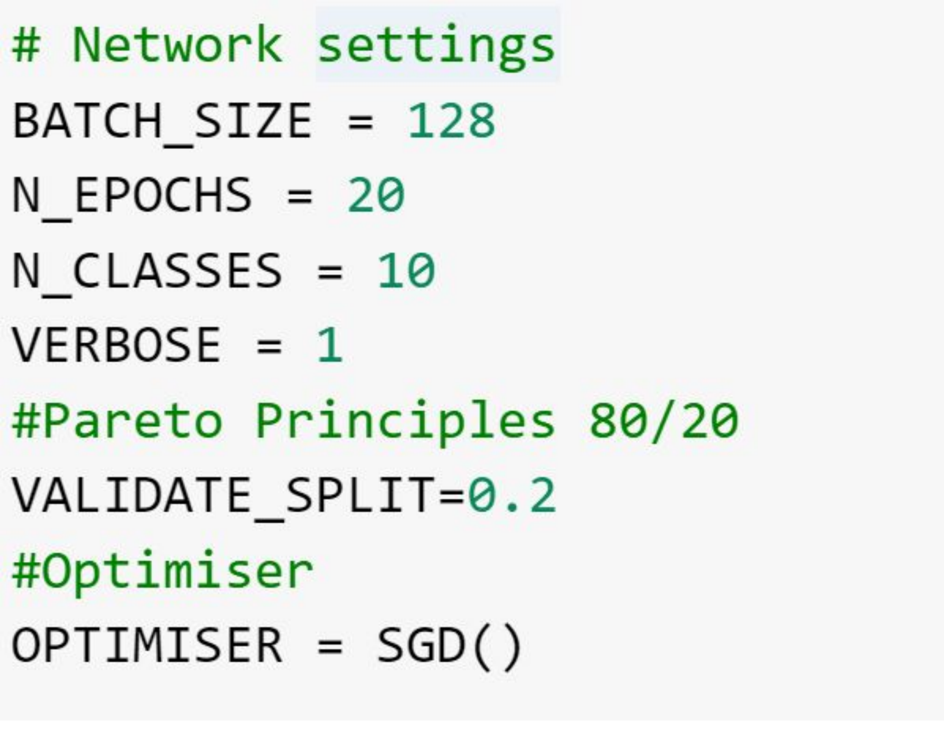
```
## Cifar Dataset
from keras.datasets import cifar10
# np_utils for one hot encoding
from keras.utils import np_utils
#Sequential model for FC layers
from keras.models import Sequential
#Core layers
from keras.layers.core import Dense, Dropout, Activation, Flatten
#import 2D convolution and max pooling 2d layer
from keras.layers.convolutional import Conv2D, MaxPooling2D
#Import Gradien descent optimiser
from keras.optimizers import SGD
#Plotting images
import matplotlib.pyplot as plt
```

Define the RGB dimension of the picture



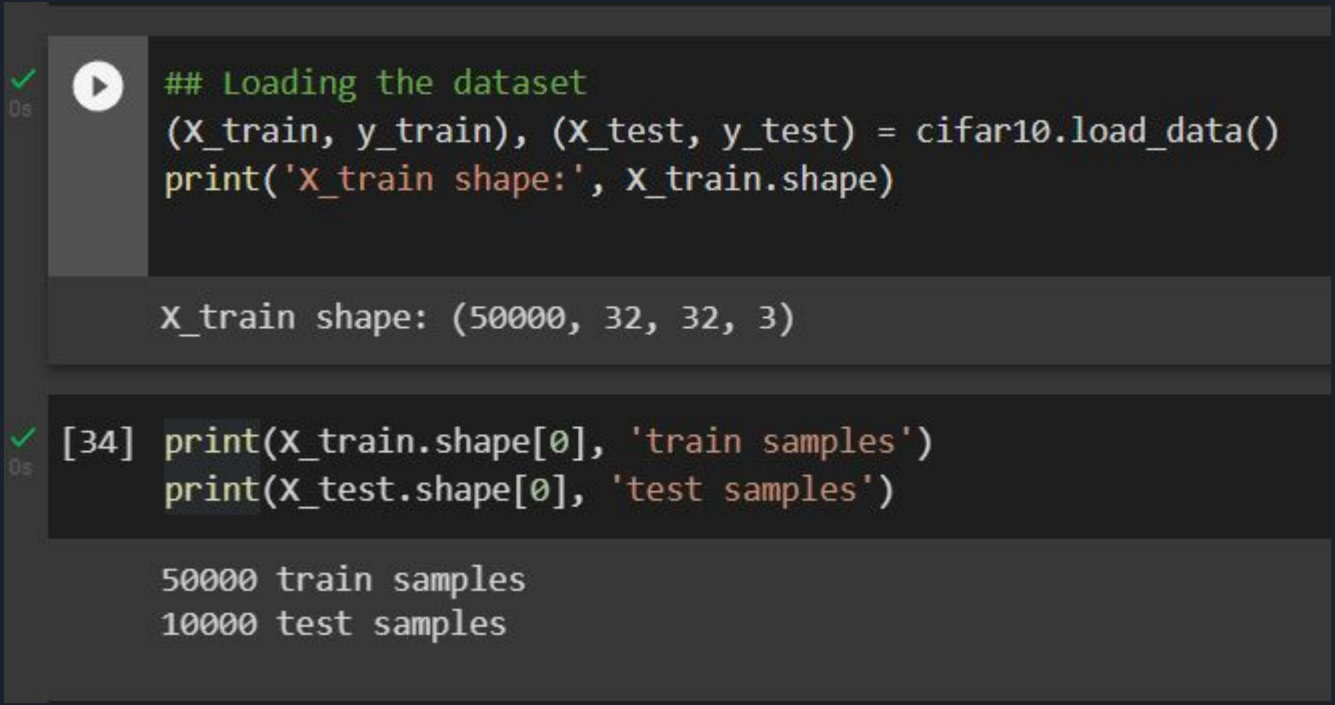
```
[31] # Cifar is a set of 60000 images of 32 by 32 pixel on 3 rgb channels
     image_channels = 3
     image_rows = 32
     image_coloumns = 32
```

Specify the hyper-parameters



```
# Network settings
BATCH_SIZE = 128
N_EPOCHS = 20
N_CLASSES = 10
VERBOSE = 1
#Pareto Principles 80/20
VALIDATE_SPLIT=0.2
#Optimiser
OPTIMISER = SGD()
```

Get the size of the dataset



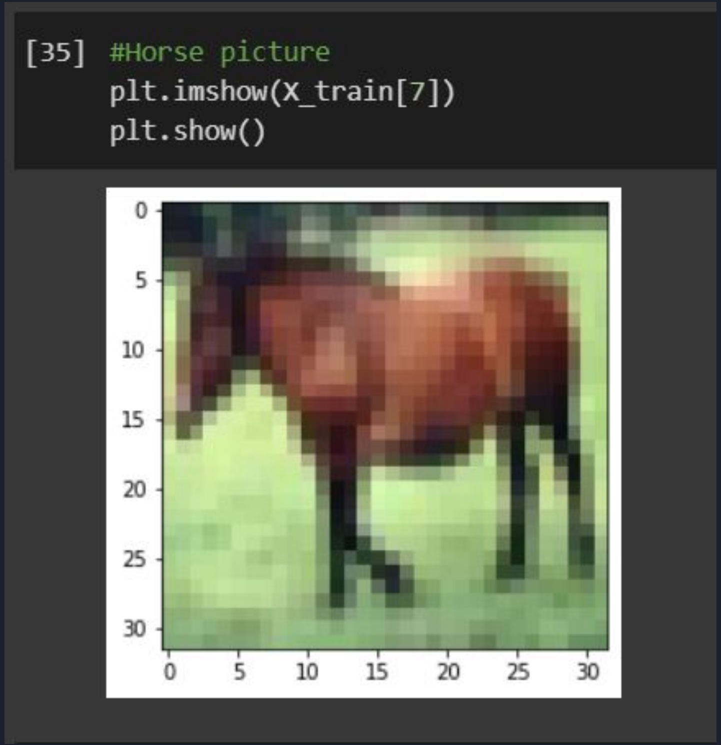
```
## Loading the dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
```

```
X_train shape: (50000, 32, 32, 3)
```

```
[34] print(X_train.shape[0], 'train samples')
     print(X_test.shape[0], 'test samples')
```

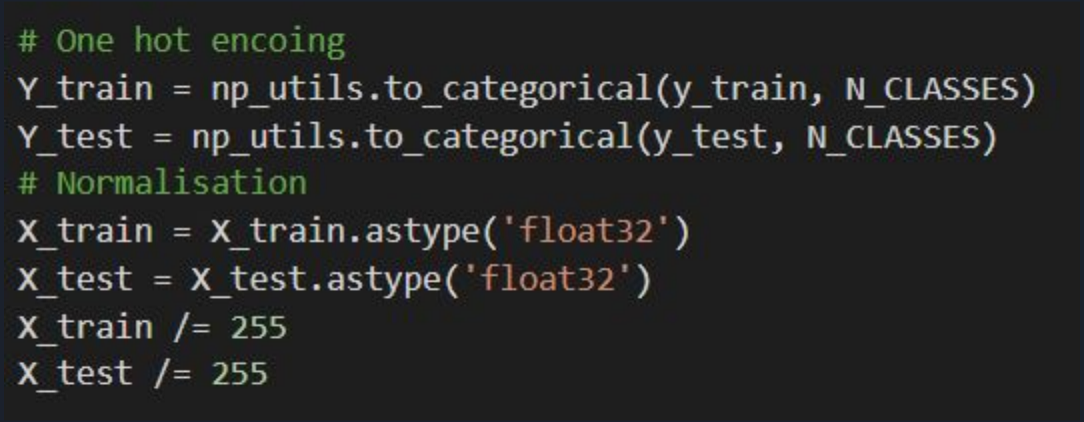
```
50000 train samples
10000 test samples
```

Print some random image



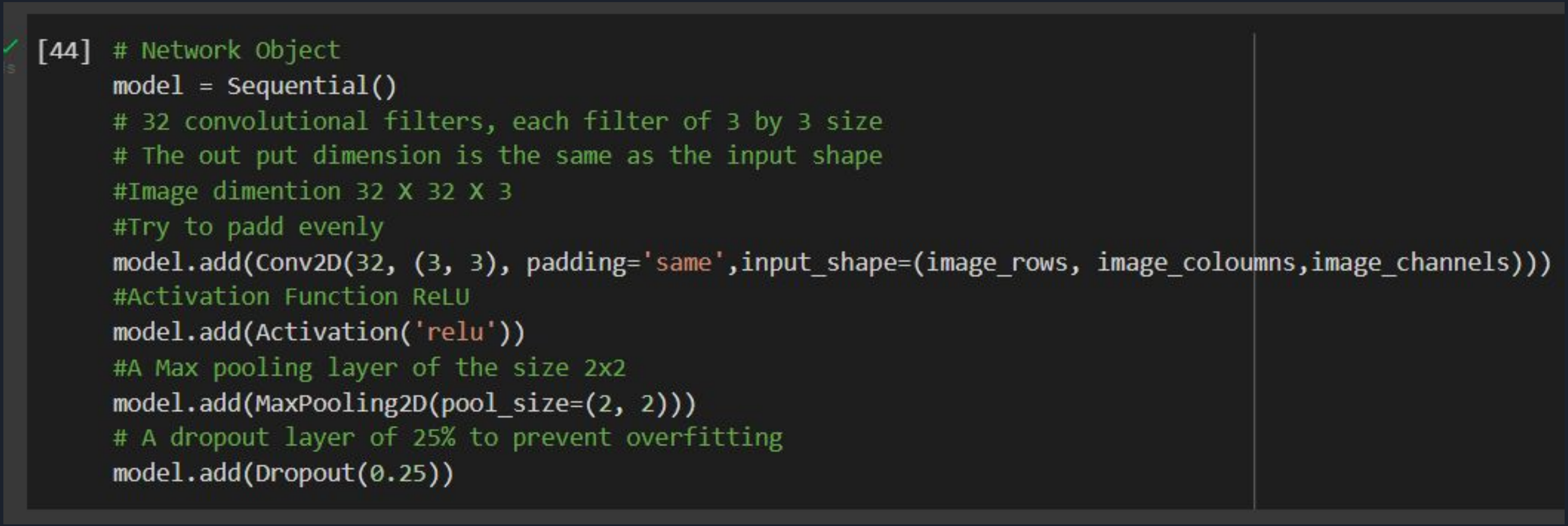
```
[35] #Horse picture
     plt.imshow(X_train[7])
     plt.show()
```

Apply normalisation



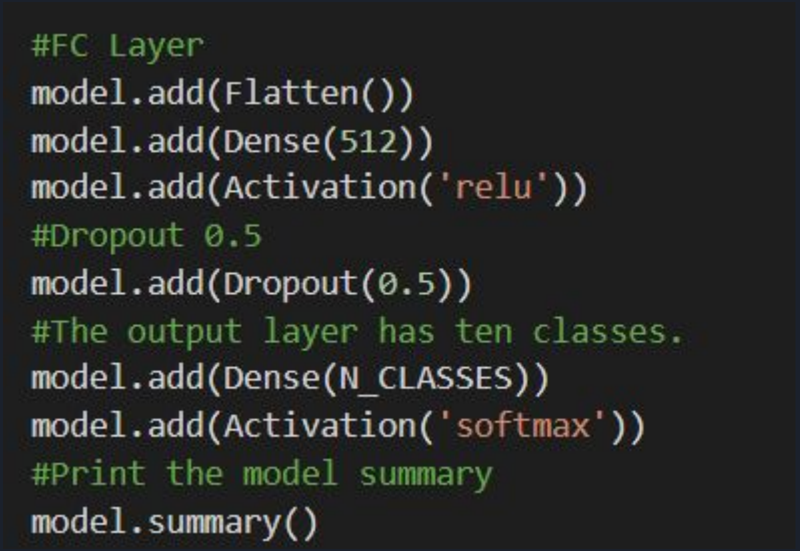
```
# One hot encoing
Y_train = np_utils.to_categorical(y_train, N_CLASSES)
Y_test = np_utils.to_categorical(y_test, N_CLASSES)
# Normalisation
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Building the CNN Network



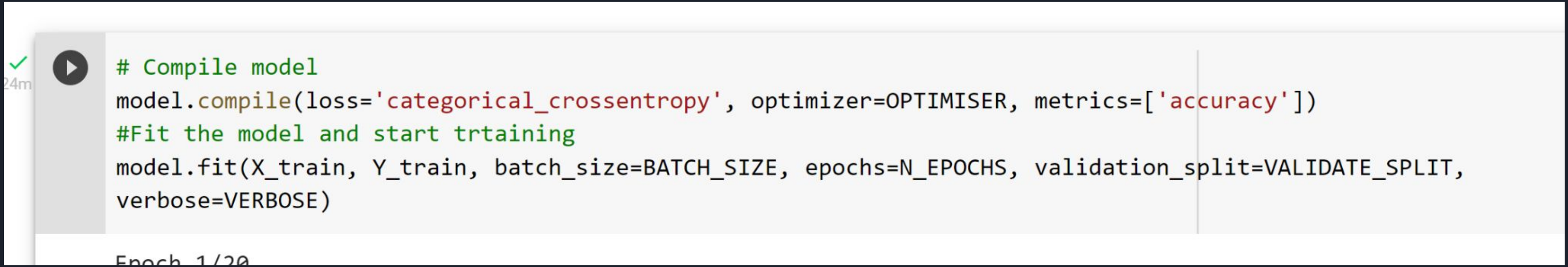
```
[44] # Network Object
     model = Sequential()
     # 32 convolutional filters, each filter of 3 by 3 size
     # The out put dimension is the same as the input shape
     #Image dimention 32 X 32 X 3
     #Try to padd evenly
     model.add(Conv2D(32, (3, 3), padding='same',input_shape=(image_rows, image_coloumns,image_channels)))
     #Activation Function ReLU
     model.add(Activation('relu'))
     #A Max pooling layer of the size 2x2
     model.add(MaxPooling2D(pool_size=(2, 2)))
     # A dropout layer of 25% to prevent overfitting
     model.add(Dropout(0.25))
```

Neural Networks Architecture: FC layer



```
#FC Layer
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
#Dropout 0.5
model.add(Dropout(0.5))
#The output layer has ten classes.
model.add(Dense(N_CLASSES))
model.add(Activation('softmax'))
#Print the model summary
model.summary()
```

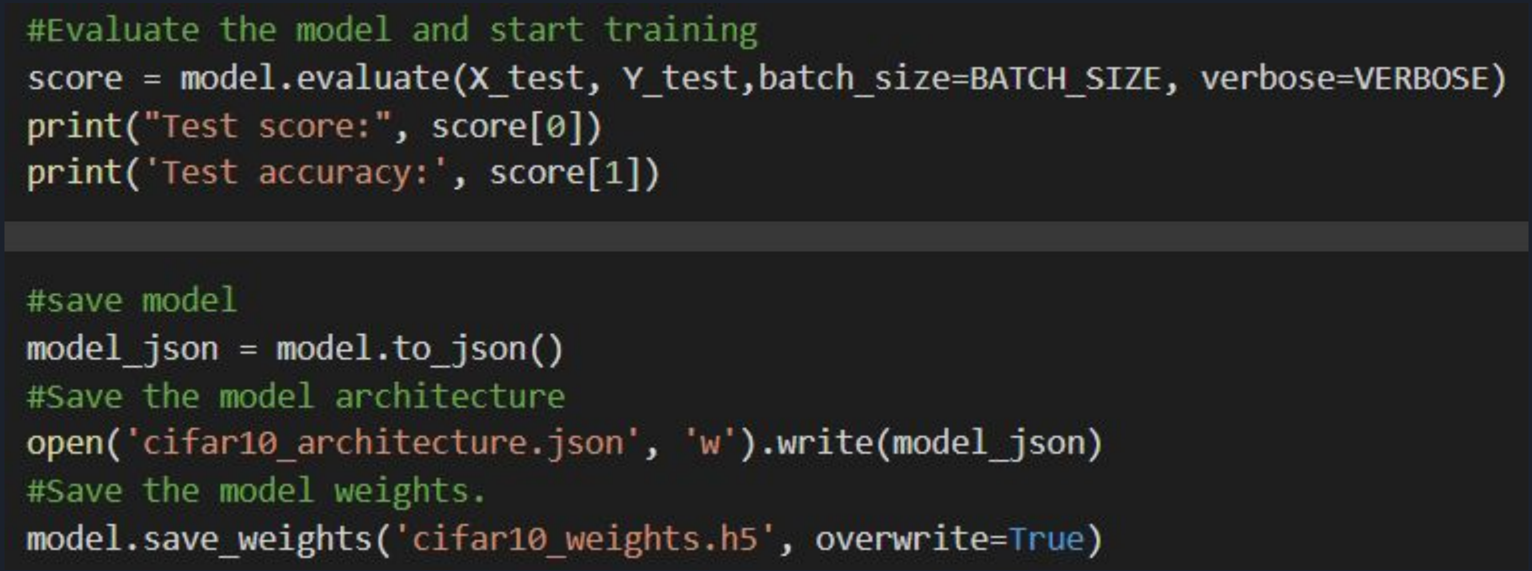
Compile and build the model



```
# Compile model
model.compile(loss='categorical_crossentropy', optimizer=OPTIMISER, metrics=['accuracy'])
#Fit the model and start trtaining
model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=N_EPOCHS, validation_split=VALIDATE_SPLIT,
verbose=VERBOSE)
```

```
Epoch 1/20
```

Save the model



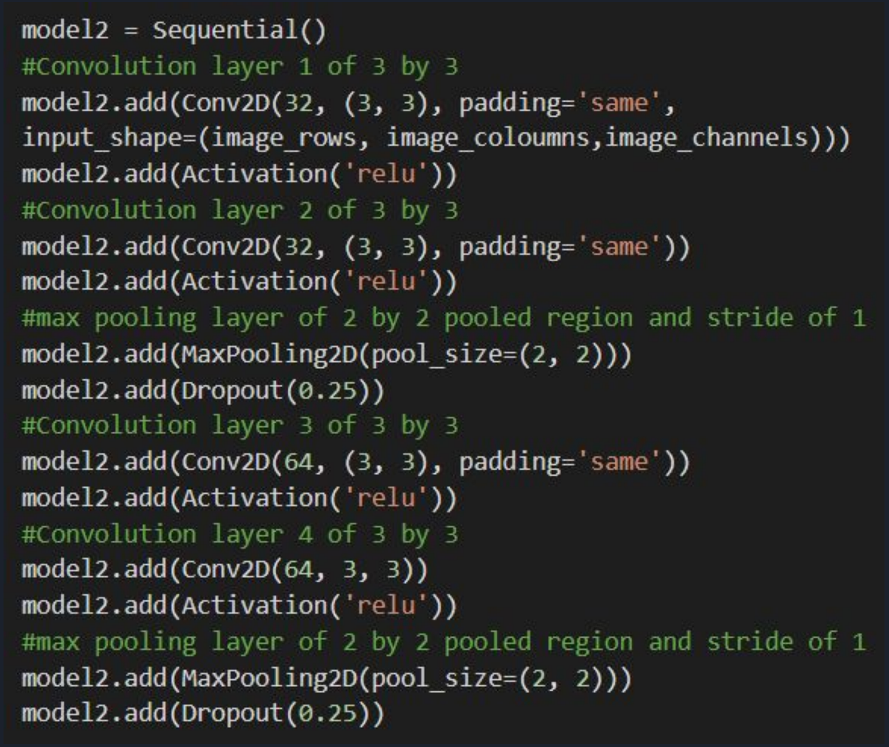
```
#Evaluate the model and start training
score = model.evaluate(X_test, Y_test,batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
#save model
model_json = model.to_json()
#Save the model architecture
open('cifar10_architecture.json', 'w').write(model_json)
#Save the model weights.
model.save_weights('cifar10_weights.h5', overwrite=True)
```

Your turn

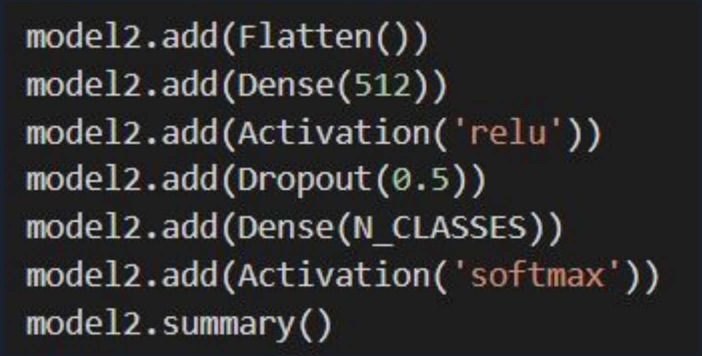
- Change the hyper-parameters (epochs, add layers of Fully connected.)
- Change the kernel size to 2x3 and 4x4
- Change the pooling to 3x3
- Modify the dropout and see if the model improves.
- Visualise the accuracy results

Build a deeper neural network



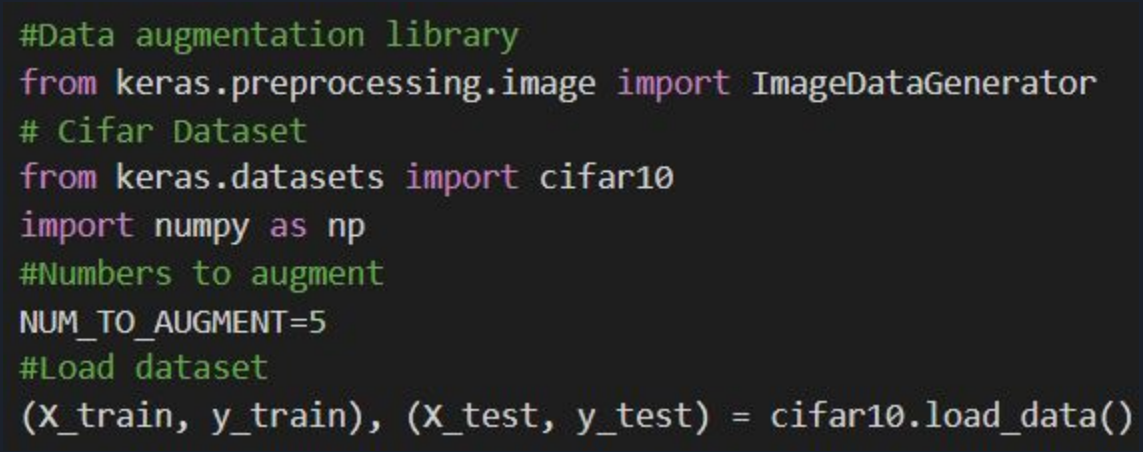
```
model2 = Sequential()
#Convolution layer 1 of 3 by 3
model2.add(Conv2D(32, (3, 3), padding='same',
input_shape=(image_rows, image_coloumns,image_channels)))
model2.add(Activation('relu'))
#Convolution layer 2 of 3 by 3
model2.add(Conv2D(32, (3, 3), padding='same'))
model2.add(Activation('relu'))
#max pooling layer of 2 by 2 pooled region and stride of 1
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))
#Convolution layer 3 of 3 by 3
model2.add(Conv2D(64, (3, 3), padding='same'))
model2.add(Activation('relu'))
#Convolution layer 4 of 3 by 3
model2.add(Conv2D(64, 3, 3))
model2.add(Activation('relu'))
#max pooling layer of 2 by 2 pooled region and stride of 1
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))
```

Fully connected layer



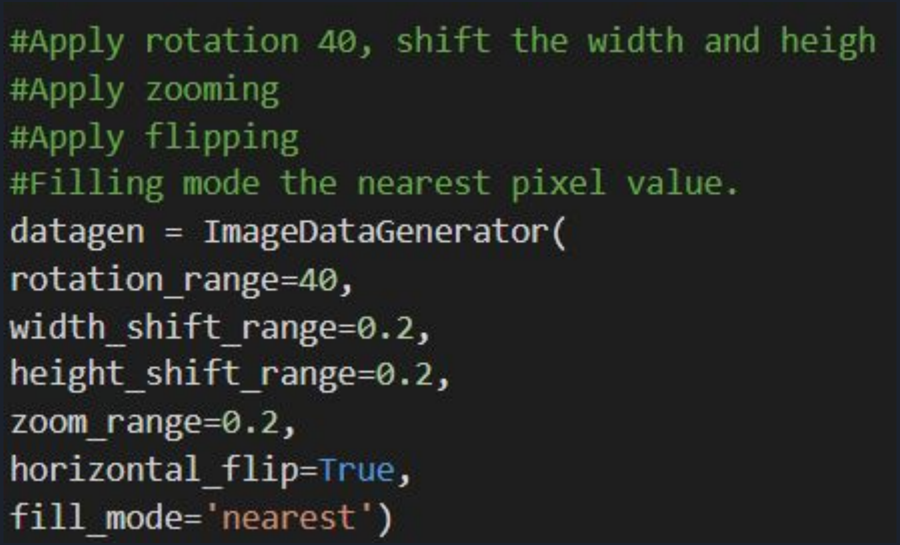
```
model2.add(Flatten())
model2.add(Dense(512))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(N_CLASSES))
model2.add(Activation('softmax'))
model2.summary()
```

Data augmentation



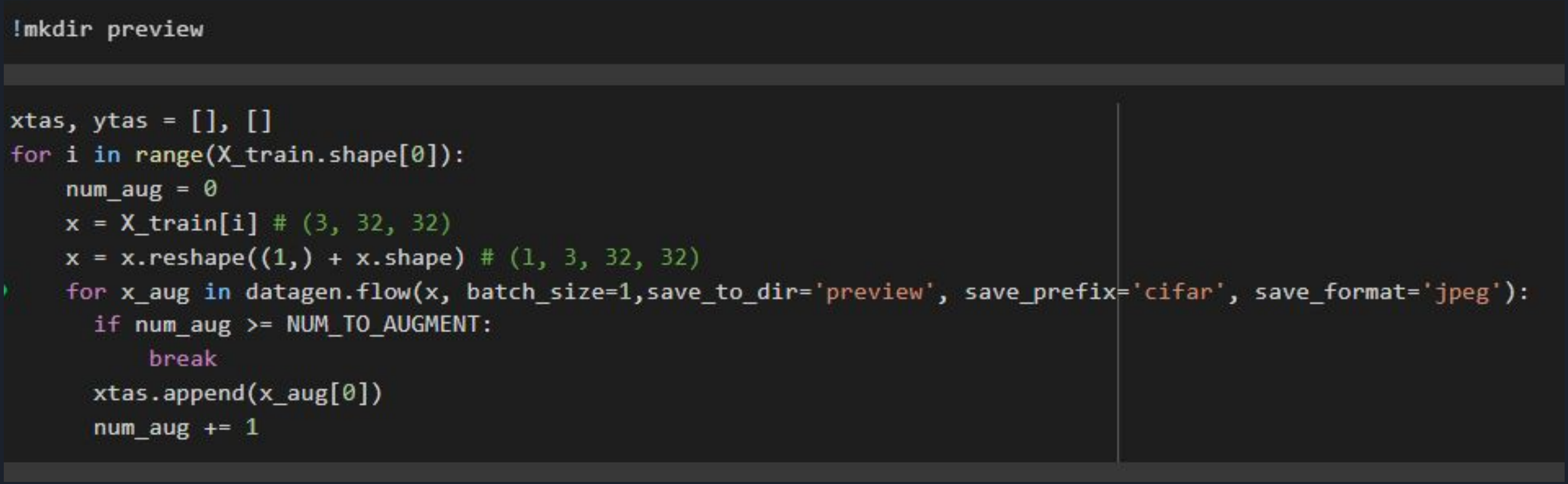
```
#Data augmentation library
from keras.preprocessing.image import ImageDataGenerator
# Cifar Dataset
from keras.datasets import cifar10
import numpy as np
#Numbers to augment
NUM_TO_AUGMENT=5
#Load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

Apply transformation it should take 10-15 minutes make sure you select GPU run time.



```
#Apply rotation 40, shift the width and heigh
#Apply zooming
#Apply flipping
#Filling mode the nearest pixel value.
datagen = ImageDataGenerator(
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')
```

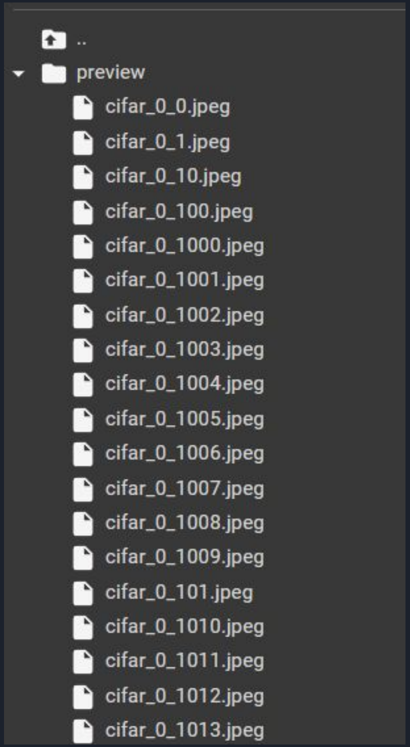
Make a directory and start the data augmentation See on the folder explorer section you will see some new images



```
!mkdir preview
```

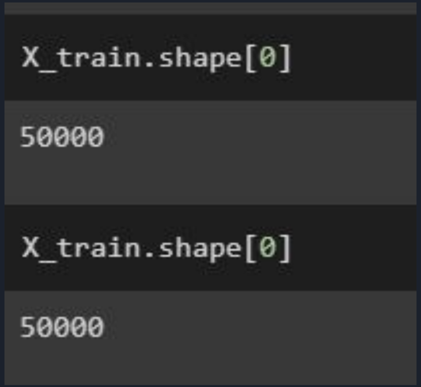
```
xtas, ytas = [], []
for i in range(X_train.shape[0]):
    num_aug = 0
    x = X_train[i] # (3, 32, 32)
    x = x.reshape((1,) + x.shape) # (1, 3, 32, 32)
    for x_aug in datagen.flow(x, batch_size=1,save_to_dir='preview', save_prefix='cifar', save_format='jpeg'):
      if num_aug >= NUM_TO_AUGMENT:
          break
      xtas.append(x_aug[0])
      num_aug += 1
```

On the folder explorer you should see these pictures after 1 minute.



```
..
preview
    cifar_0_0.jpeg
    cifar_0_1.jpeg
    cifar_0_10.jpeg
    cifar_0_100.jpeg
    cifar_0_1000.jpeg
    cifar_0_1001.jpeg
    cifar_0_1002.jpeg
    cifar_0_1003.jpeg
    cifar_0_1004.jpeg
    cifar_0_1005.jpeg
    cifar_0_1006.jpeg
    cifar_0_1007.jpeg
    cifar_0_1008.jpeg
    cifar_0_1009.jpeg
    cifar_0_101.jpeg
    cifar_0_1010.jpeg
    cifar_0_1011.jpeg
    cifar_0_1012.jpeg
    cifar_0_1013.jpeg
```

Check the shape etc.



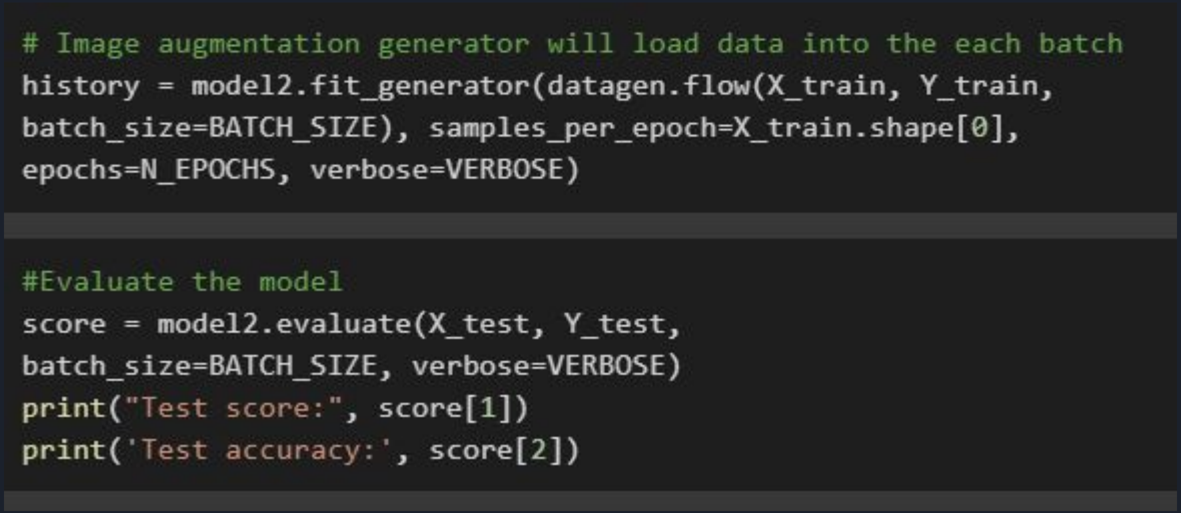
```
X_train.shape[0]
```

```
50000
```

```
X_train.shape[0]
```

```
50000
```

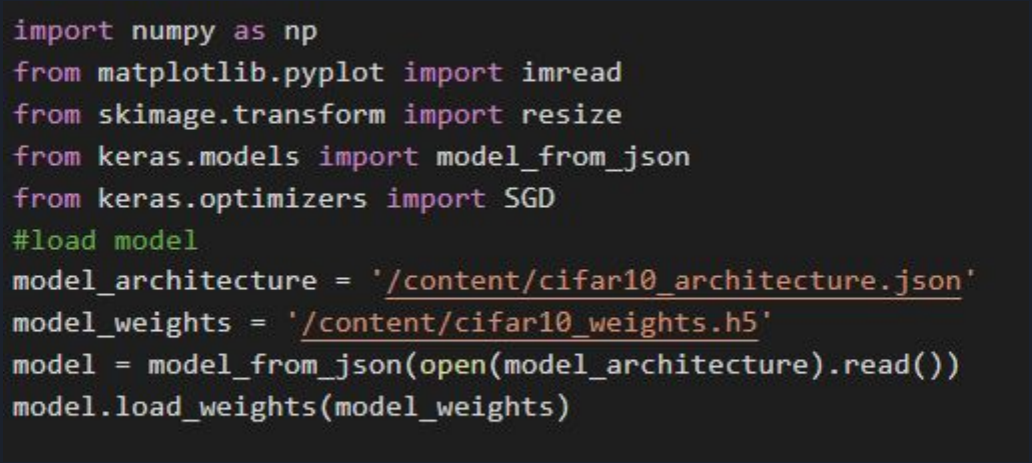
Fit the deep neural network model created in slide 12-13 into the dataset and start training and evaluation



```
# Image augmentation generator will load data into the each batch
history = model2.fit_generator(datagen.flow(X_train, Y_train,
batch_size=BATCH_SIZE), samples_per_epoch=X_train.shape[0],
epochs=N_EPOCHS, verbose=VERBOSE)
```

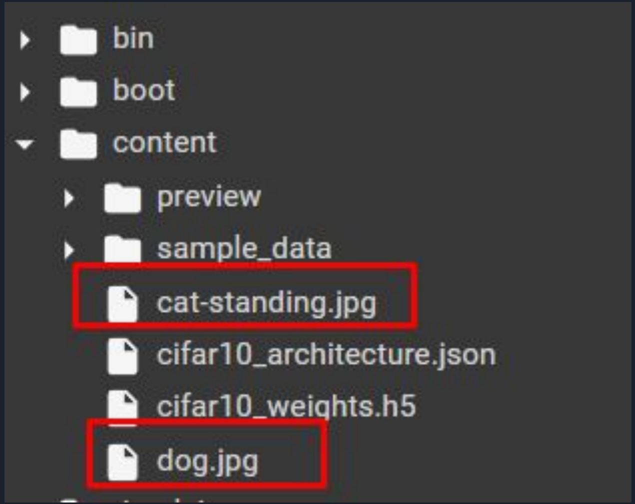
```
#Evaluate the model
score = model2.evaluate(X_test, Y_test,
batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[1])
print('Test accuracy:', score[2])
```

Load the previously trained and saved model.



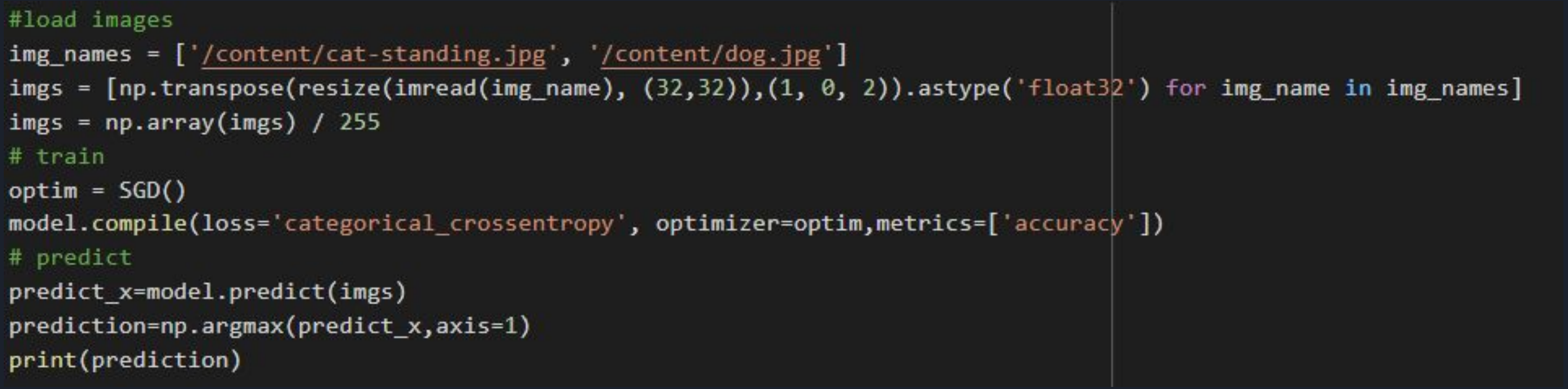
```python
import numpy as np
from matplotlib.pyplot import imread
from skimage.transform import resize
from keras.models import model_from_json
from keras.optimizers import SGD
#load model
model_architecture = '/content/cifar10_architecture.json'
model_weights = '/content/cifar10_weights.h5'
model = model_from_json(open(model_architecture).read())
model.load_weights(model_weights)
```

Upload the images on the vle tutorial under images section onto the content folder in your google colab



- bin
- boot
- content
  - preview
  - sample_data
  - cat-standing.jpg
  - cifar10_architecture.json
  - cifar10_weights.h5
  - dog.jpg

Predict dogs and cats



```
#load images
img_names = ['/content/cat-standing.jpg', '/content/dog.jpg']
imgs = [np.transpose(resize(imread(img_name), (32,32)),(1, 0, 2)).astype('float32') for img_name in img_names]
imgs = np.array(imgs) / 255
# train
optim = SGD()
model.compile(loss='categorical_crossentropy', optimizer=optim,metrics=['accuracy'])
# predict
predict_x=model.predict(imgs)
prediction=np.argmax(predict_x,axis=1)
print(prediction)
```

Tasks for home

- Try implementing CNN to another dataset use Kaggle
- Choose one from here: https://www.kaggle.com/search?q=image+in%3Adatasets+tags%3Aimage
- Explore other data augmentation techniques   follow this tutorial
- Apply various pooling and convolving techniques to the dataset chosen.

End of tutorial