DEEP NEURAL NETWORK

Import TensorFlow

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

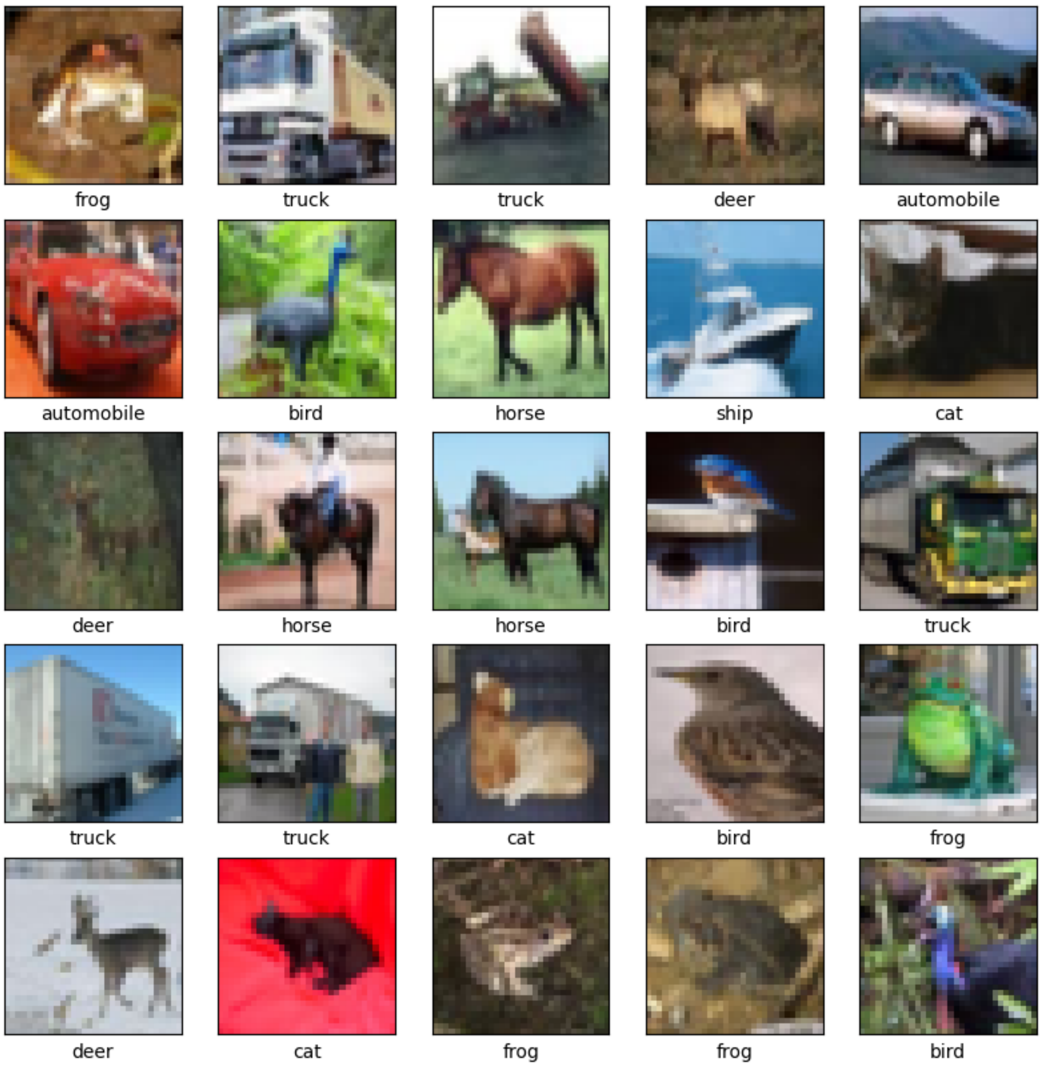
Download and prepare the CIFAR10 dataset

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 14s 0us/step
```

Verify the data

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Create the convolutional base

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Let's display the architecture of your model so far:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Add Dense layers on top

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Here's the complete architecture of your model:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 64)                65600     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compile and train the model

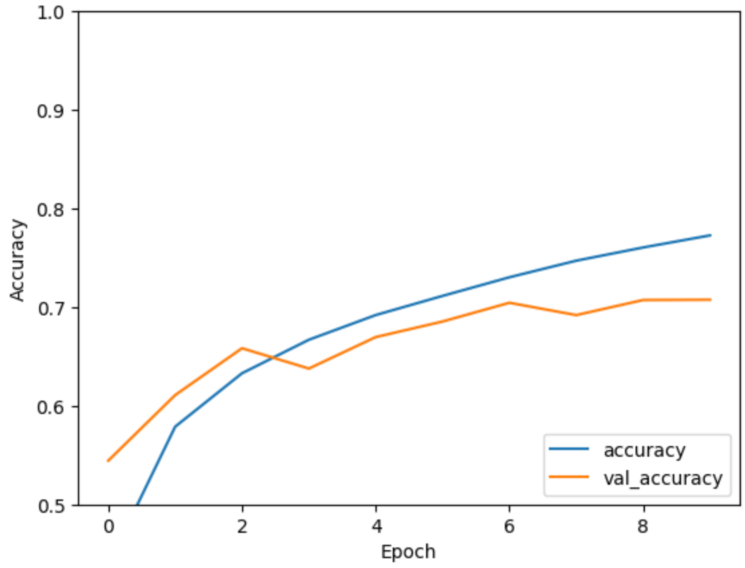
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, 
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 20s 6ms/step - loss: 1.5405 - accuracy: 0.4390 - val_loss: 1.2710 - val_accuracy: 0.5444
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1881 - accuracy: 0.5788 - val_loss: 1.0931 - val_accuracy: 0.6108
Epoch 3/10
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0436 - accuracy: 0.6329 - val_loss: 0.9882 - val_accuracy: 0.6583
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9511 - accuracy: 0.6669 - val_loss: 1.0265 - val_accuracy: 0.6377
Epoch 5/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8768 - accuracy: 0.6919 - val_loss: 0.9480 - val_accuracy: 0.6696
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8200 - accuracy: 0.7114 - val_loss: 0.9161 - val_accuracy: 0.6854
Epoch 7/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7691 - accuracy: 0.7302 - val_loss: 0.8694 - val_accuracy: 0.7044
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7235 - accuracy: 0.7471 - val_loss: 0.8892 - val_accuracy: 0.6919
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6879 - accuracy: 0.7604 - val_loss: 0.8709 - val_accuracy: 0.7072
Epoch 10/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6495 - accuracy: 0.7727 - val_loss: 0.8733 - val_accuracy: 0.7075
```

Evaluate the model

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8733 - accuracy: 0.7075 - 691ms/epoch - 2ms/step
```



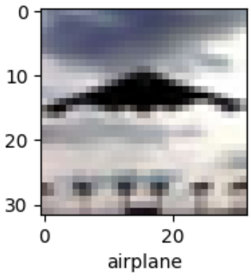
Print accuracy

```
print(test_acc)
```

```
0.7074999809265137
```

Prediction

```
n=111  ##Number of image

plt.figure(figsize=(2,2))
plt.imshow(test_images[n])
plt.xlabel(class_names[test_labels[n][0]])
plt.show()
```



```
import numpy as np


predictions = model.predict(test_images)
print(predictions[n])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(predictions[n])], 10 * np.max(predictions[n]))
)
```

```
313/313 [==============================] - 1s 2ms/step
[ 6.858806   4.0483427  2.6050212 -0.4147426 -3.392217  -7.305912
 -5.7308545 -5.454667   1.9584597 -1.1152903]
This image most likely belongs to airplane with a 68.59 percent confidence.
```