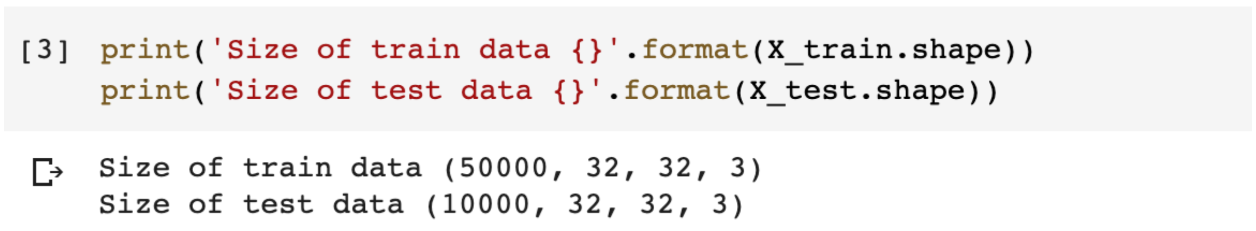
BigData Analytics - ICP05
Created by - Prabhanjan Trivedi

Task - Use the same data (that we used in ICP4
from keras.datasets import cifar10)) and use the model provided in ICP5 to perform image classification. You must change 4 hyper parameters in the source code. Report your findings in detail.
Note: please indicate in your reports which 4 hyperparameters you changed in the source code and why in your opinion these changes are logical.

Process - In this example, we will be using the famous CIFAR-10 dataset. CIFAR-10 is a large image dataset containing over 60,000 images representing 10 different classes of objects like cats, planes, and cars.

The images are full-color RGB, but they are fairly small, only 32 x 32. One great thing about the CIFAR-10 dataset is that it comes prepackaged with Keras, so it is very easy to load up the dataset and the images need very little preprocessing.

We have train data features of 50K image and for testing we have 10K.

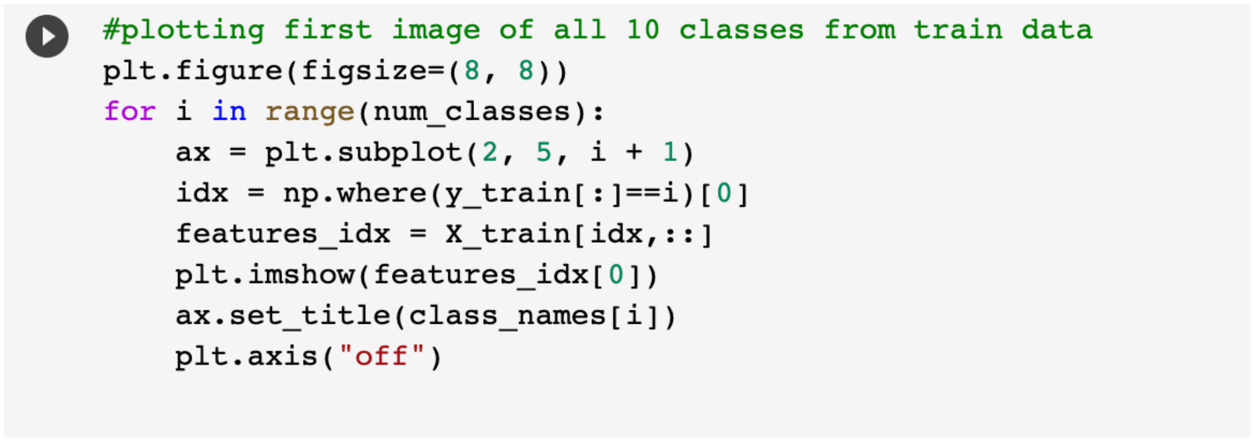


```
[3] print('Size of train data {}'.format(X_train.shape))
    print('Size of test data {}'.format(X_test.shape))
```

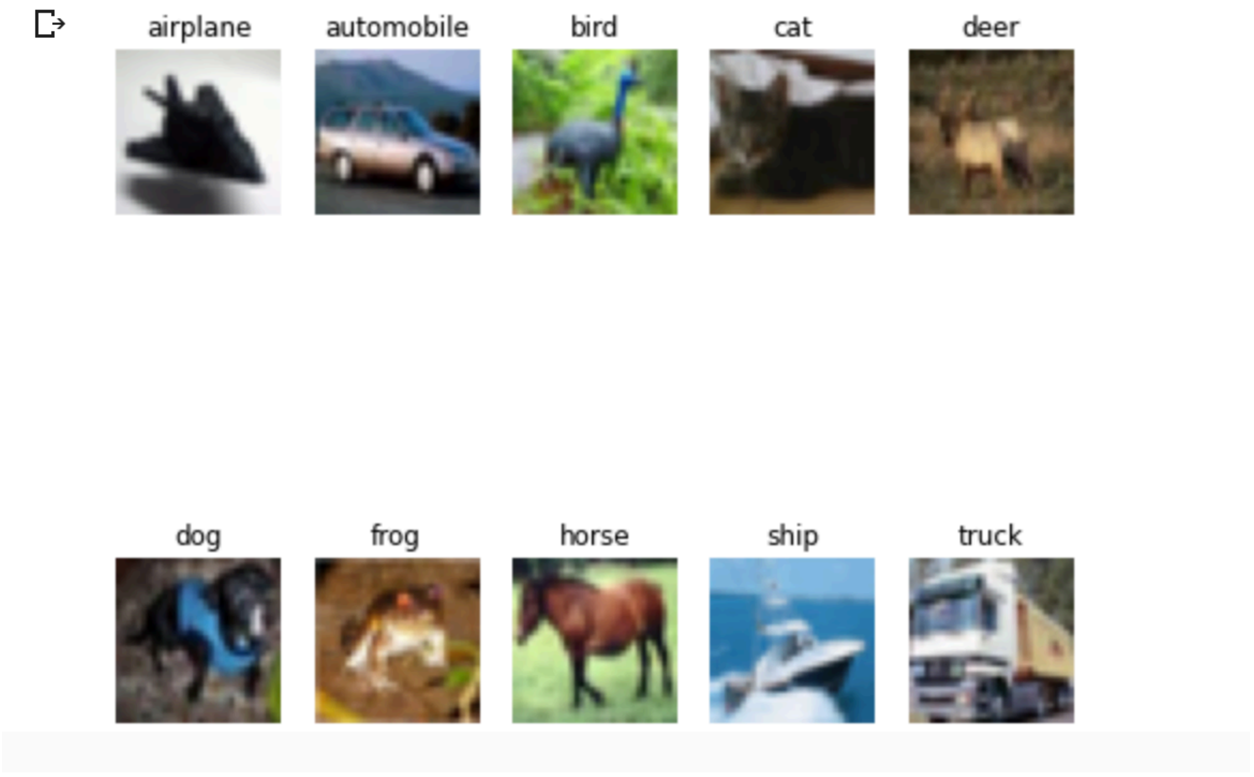
```
Size of train data (50000, 32, 32, 3)
Size of test data (10000, 32, 32, 3)
```

In this Dataset, we have following 10 classes Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.

Here we plotted first image of each class from training data set.



```
#plotting first image of all 10 classes from train data
plt.figure(figsize=(8, 8))
for i in range(num_classes):
    ax = plt.subplot(2, 5, i + 1)
    idx = np.where(y_train[:]==i)[0]
    features_idx = X_train[idx,::]
    plt.imshow(features_idx[0])
    ax.set_title(class_names[i])
    plt.axis("off")
```



In most cases we will need to do some preprocessing of our data to get it ready for use, but since we are using a prepackaged dataset, very little preprocessing needs to be done. One thing we want to do is normalize the input data.

If the values of the input data are in too wide a range it can negatively impact how the network performs. In this case, the input values are the pixels in the image, which have a value between 0 to 255.

So in order to normalize the data we can simply divide the image values by 255. To do this we first need to make the data a float type, since they are currently integers. We can do this by using the astype() Numpy command and then declared as 'float32'.

```
[8]  # normalize the inputs from 0-255 to between 0 and 1 by dividing by 255
     X_train = X_train.astype('float32')
     X_test = X_test.astype('float32')
     X_train = X_train / 255.0
     X_test = X_test / 255.0
```

We changed the output data in matrix form using one hot-encode.A one hot encoding is a representation of categorical variables as binary vectors.

This first requires that the categorical values be mapped to integer values.Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1. We used to_categorical() command from bumpy utils function.

The Numpy command to_categorical() is used to one-hot encode. This is why we imported the np_utils function from Keras, as it contains to_categorical().

```
[9]  # one hot encode outputs
     y_train = np_utils.to_categorical(y_train)
     y_test = np_utils.to_categorical(y_test)
```

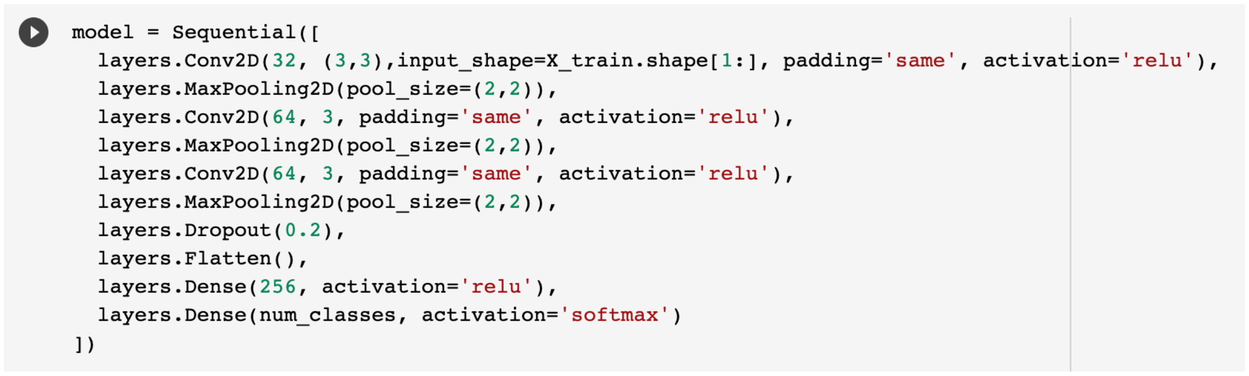
Model Creation -

The model consists of three convolution blocks with a max pool layer in each of them.

- In the first stage, Our net will learn 32 convolutional filters, each of which with a 3 x 3 size. Activation is relu, which is a simple way of introducing non-linearity. After that we have a max-pooling operation with pool size 2 x 2 followed by two another 64 convolutional filters, each of which with a 3 x 3 size and activation is also relu.
- In the next stage in the deep pipeline, Our model will have a dropout at 20% and then flatten the layers. The Final stage in the deep pipeline is a dense network with 256 units and relu activation followed by another dense layer with softmax activation layer with 10 classes as output, one for each category.

Following hyperparametrs are changed from ICP05 in class model.

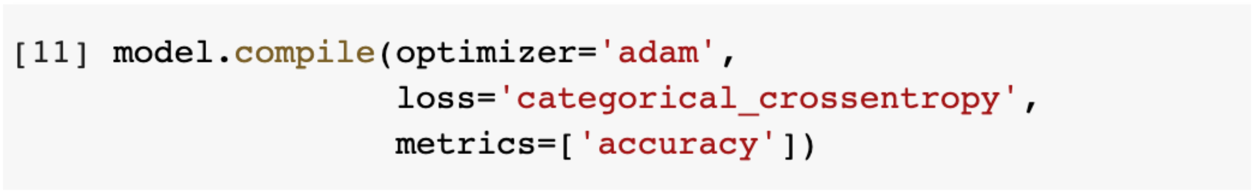
1. First CNN has filter of 32 (In class model, it was 16).
2. Second CNN has filter of 64(In class model, it was 32).
3. First dense layer has dimension of 256 (In class model, it was 128).
4. Last dense layer has activation function as softmax (In class model it was not there).
5. We use loss function as ‘categorical_crossentropy’ while in class model, it was sparse categorical cross entropy.



```
model = Sequential([
  layers.Conv2D(32, (3,3),input_shape=X_train.shape[1:], padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2,2)),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2,2)),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2,2)),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dense(num_classes, activation='softmax')
])
```

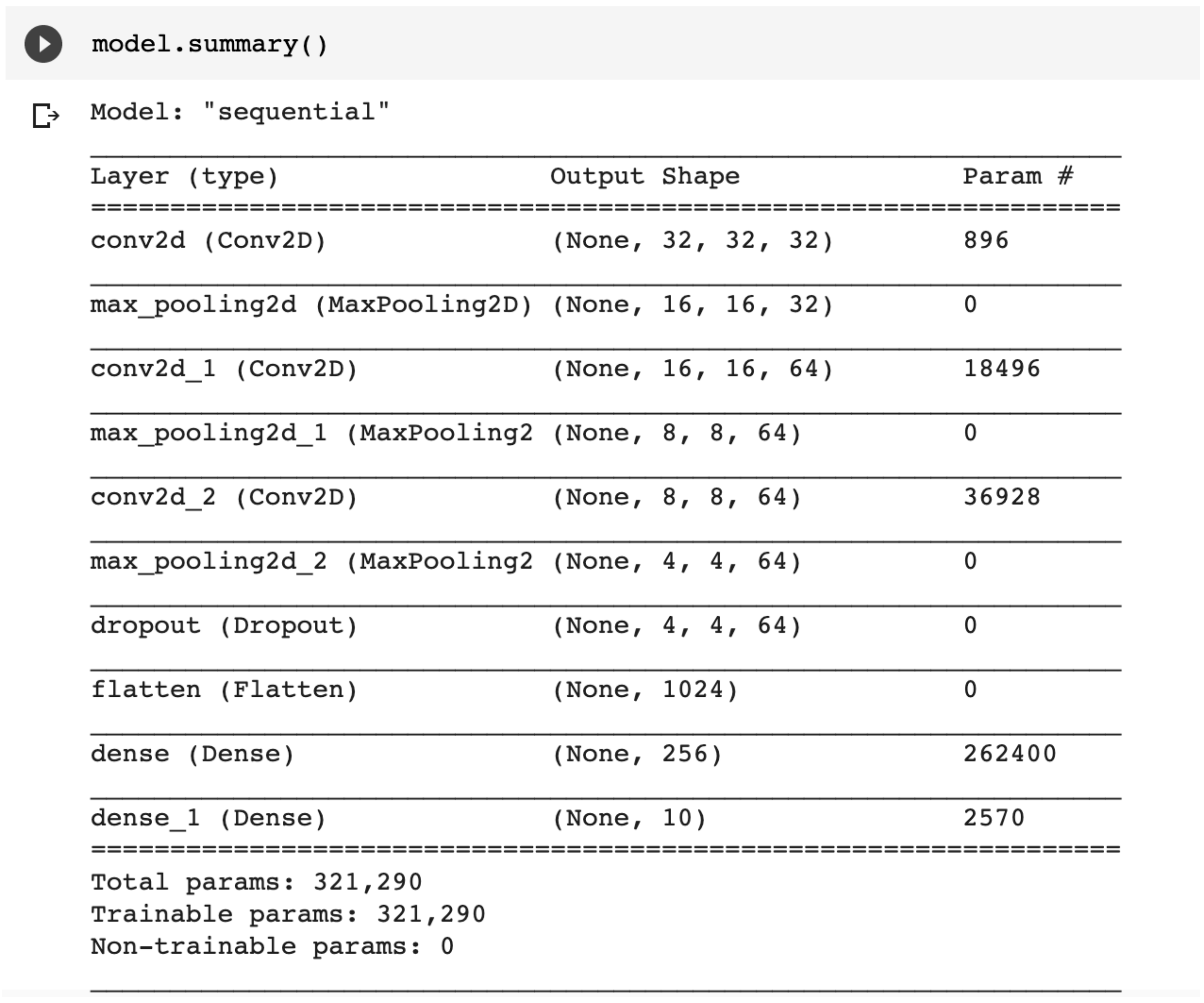
Compile the model

Choose the optimizers as Adam optimizer and losses as Categorical_Crossentropy loss function beacuse we have hot encoding for categorical value y_test and y_train.



```
[11] model.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

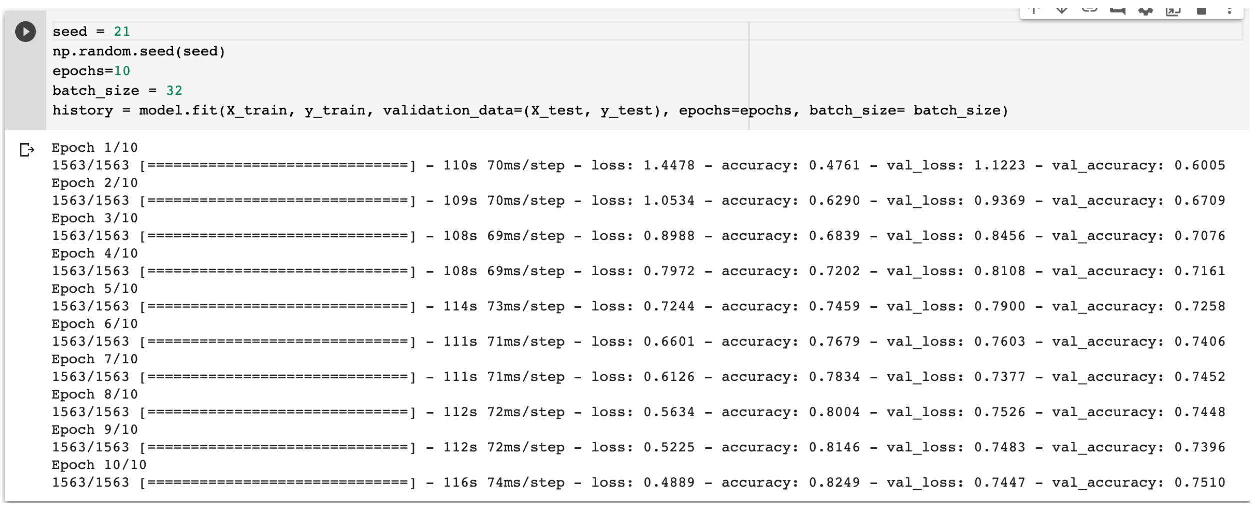
Here we have model summary. As per model, we have 321,290 parameters to train.



```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0         
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 256)               262400    
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570      
=================================================================
Total params: 321,290
Trainable params: 321,290
Non-trainable params: 0
_________________________________________________________________
```

We train the model with seed value of 21, batch size of 32 and epochs 10.



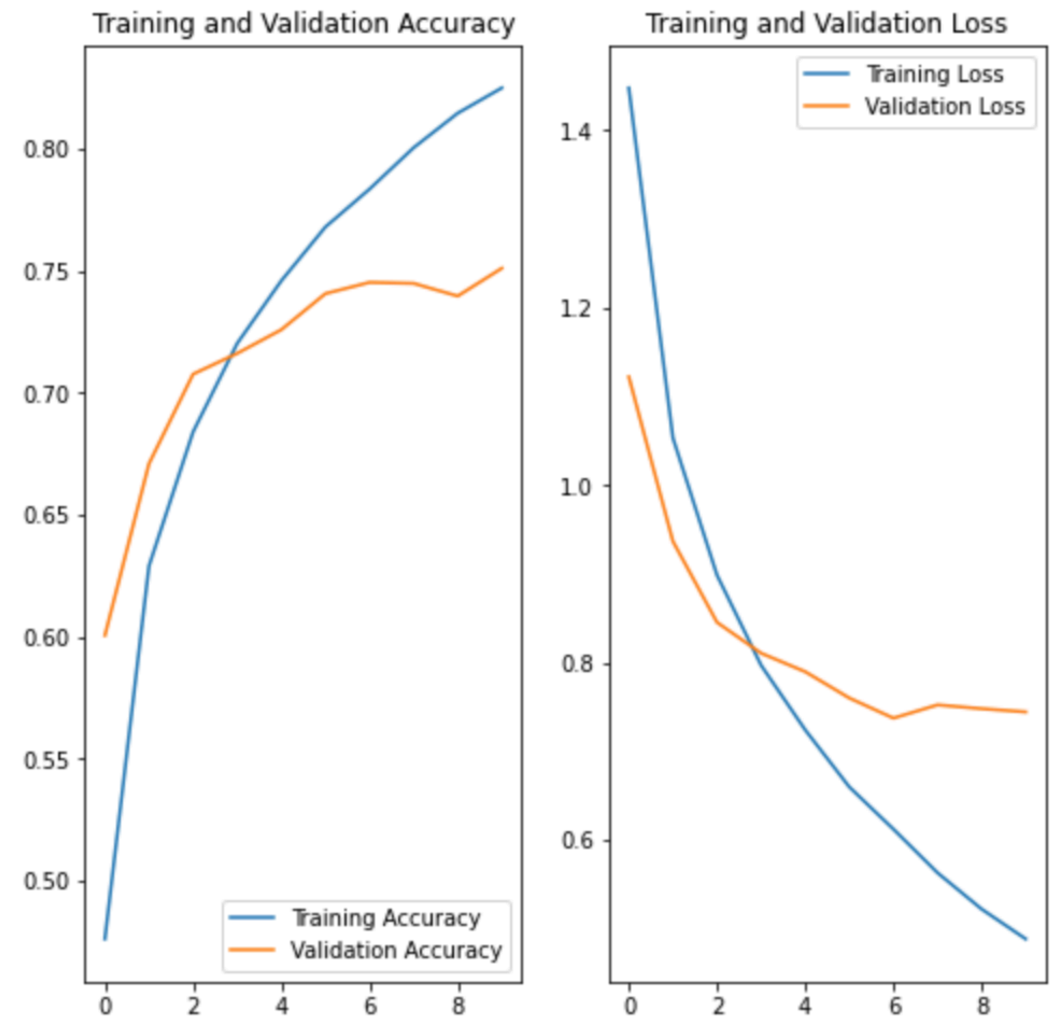
```
seed = 21
np.random.seed(seed)
epochs=10
batch_size = 32
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size= batch_size)
```

```
Epoch 1/10
1563/1563 [==============================] - 110s 70ms/step - loss: 1.4478 - accuracy: 0.4761 - val_loss: 1.1223 - val_accuracy: 0.6005
Epoch 2/10
1563/1563 [==============================] - 109s 70ms/step - loss: 1.0534 - accuracy: 0.6290 - val_loss: 0.9369 - val_accuracy: 0.6709
Epoch 3/10
1563/1563 [==============================] - 108s 69ms/step - loss: 0.8988 - accuracy: 0.6839 - val_loss: 0.8456 - val_accuracy: 0.7076
Epoch 4/10
1563/1563 [==============================] - 108s 69ms/step - loss: 0.7972 - accuracy: 0.7202 - val_loss: 0.8108 - val_accuracy: 0.7161
Epoch 5/10
1563/1563 [==============================] - 114s 73ms/step - loss: 0.7244 - accuracy: 0.7459 - val_loss: 0.7900 - val_accuracy: 0.7258
Epoch 6/10
1563/1563 [==============================] - 111s 71ms/step - loss: 0.6601 - accuracy: 0.7679 - val_loss: 0.7603 - val_accuracy: 0.7406
Epoch 7/10
1563/1563 [==============================] - 111s 71ms/step - loss: 0.6126 - accuracy: 0.7834 - val_loss: 0.7377 - val_accuracy: 0.7452
Epoch 8/10
1563/1563 [==============================] - 112s 72ms/step - loss: 0.5634 - accuracy: 0.8004 - val_loss: 0.7526 - val_accuracy: 0.7448
Epoch 9/10
1563/1563 [==============================] - 112s 72ms/step - loss: 0.5225 - accuracy: 0.8146 - val_loss: 0.7483 - val_accuracy: 0.7396
Epoch 10/10
1563/1563 [==============================] - 116s 74ms/step - loss: 0.4889 - accuracy: 0.8249 - val_loss: 0.7447 - val_accuracy: 0.7510
```

As we can see that accuracy on training and validation data is increasing per epoch and loss is decreasing.

We plotted graph for accuracy and loss between training and validation data.

[14]



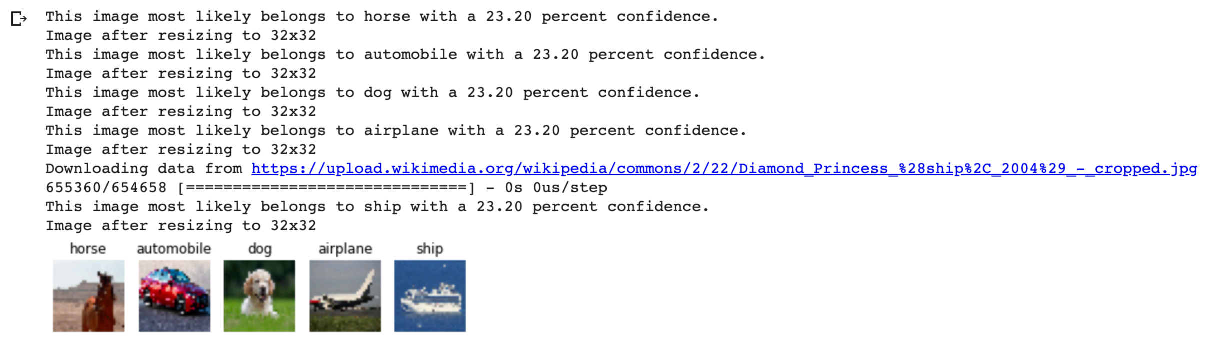
We downloaded 5 different images from internet and check the model by feeding them.

Here is input links and image type.

1. Horse, url - https://scx2.b-cdn.net/gfx/news/2020/1-geneticstudy.jpg'
2. Car, url - https://www.extremetech.com/wp-content/uploads/2019/12/SONATA-hero-option1-764A5360-edit.jpg
3. Dog, url - https://i.insider.com/5df126b679d7570ad2044f3e?width=1800&format=jpeg&auto=webp
4. plane, url - https://www.netpaths.net/wp-content/uploads/google-airplane1.jpg
5. ship, url - https://upload.wikimedia.org/wikipedia/commons/2/22/Diamond_Princess_%28ship%2C_2004%29_-_cropped.jpg

I saved this links in dictionary to iterate over and see the result.

Here is the output of the model.



```
This image most likely belongs to horse with a 23.20 percent confidence.
Image after resizing to 32x32
This image most likely belongs to automobile with a 23.20 percent confidence.
Image after resizing to 32x32
This image most likely belongs to dog with a 23.20 percent confidence.
Image after resizing to 32x32
This image most likely belongs to airplane with a 23.20 percent confidence.
Image after resizing to 32x32
Downloading data from https://upload.wikimedia.org/wikipedia/commons/2/22/Diamond_Princess_%28ship%2C_2004%29_-_cropped.jpg
655360/654658 [==============================] - 0s 0us/step
This image most likely belongs to ship with a 23.20 percent confidence.
Image after resizing to 32x32
```