Assignment No.3

Name : Anuradha Ajinath Gite

Roll-No : 4401027

Problem Statement

Build the Image classification model by dividing the model into following 4 stages

1. Loading and preprocessing the image data
2. Defining the model's architecture
3. Training the model
4. Estimating the model's performance

Solution Expected

Build the image classification model and improve model generalisation by achieving increased accuracy and descresesd loss where model gains good confidence with the prediction.

Objectives to be achieved

The objective of image classification is to identify and portray, as a unique gray level (or color), the features occurring in an image in terms of the object or type of land cover these features actually represent on the ground.

Methodology to be used

1. Deep Learning
2. Convolution Neural Network

Justification with Theory/Literature

Deep Learning

Deep learning attempts to mimic the human brain—albeit far from matching its ability—enabling systems to cluster data and make predictions with incredible accuracy. Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability — allowing it to “learn” from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

Image Classification

Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos.Image classification is probably the most important part of digital image analysis. It uses AI-based deep learning models to analyze images with results that for specific tasks already surpass human-level accuracy (for example, in face recognition). Image classification is where a computer can analyse an image and identify the ‘class’ the image falls under. (Or a probability of the image being part of a ‘class’.) A class is essentially a label, for instance, ‘car’, ‘animal’, ‘building’ and so on.

For example, you input an image of a sheep. Image classification is the process of the computer analysing the image and telling you it’s a sheep. (Or the probability that it’s a sheep.)

Process of Model Building

Before we deep dive into the Python code, let’s take a moment to understand how an image classification model is typically designed. We can divide this process broadly into 4 stages. Each stage requires a certain amount of time to execute:

Stage 1: Loading and pre-processing the data

Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a good amount of images in the training set. Also, the shape of the data varies according to the architecture/framework that we use.

Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the ‘Basics of Image Processing in Python’ to understand more about how pre-processing works with image data.

But we are not quite there yet. In order to see how our model performs on unseen data (and before exposing it to the test set), we need to create a validation set. This is done by partitioning the training set data.

In short, we train the model on the training data and validate it on the validation data. Once we are satisfied with the model’s performance on the validation set, we can use it for making predictions on the test data.

Stage 2: Defining the model’s architecture

This is another crucial step in our deep learning model building process. We have to define how our model will look and that requires answering questions like:

How many convolutional layers do we want? What should be the activation function for each layer? How many hidden units should each layer have? And many more. These are essentially the hyperparameters of the model which play a MASSIVE part in deciding how good the predictions will be.

How do we decide these values? Excellent question! A good idea is to pick these values based on existing research/studies. Another idea is to keep experimenting with the values until you find the best match but this can be quite a time consuming process.

Stage 3: Training the model

For training the model, we require:

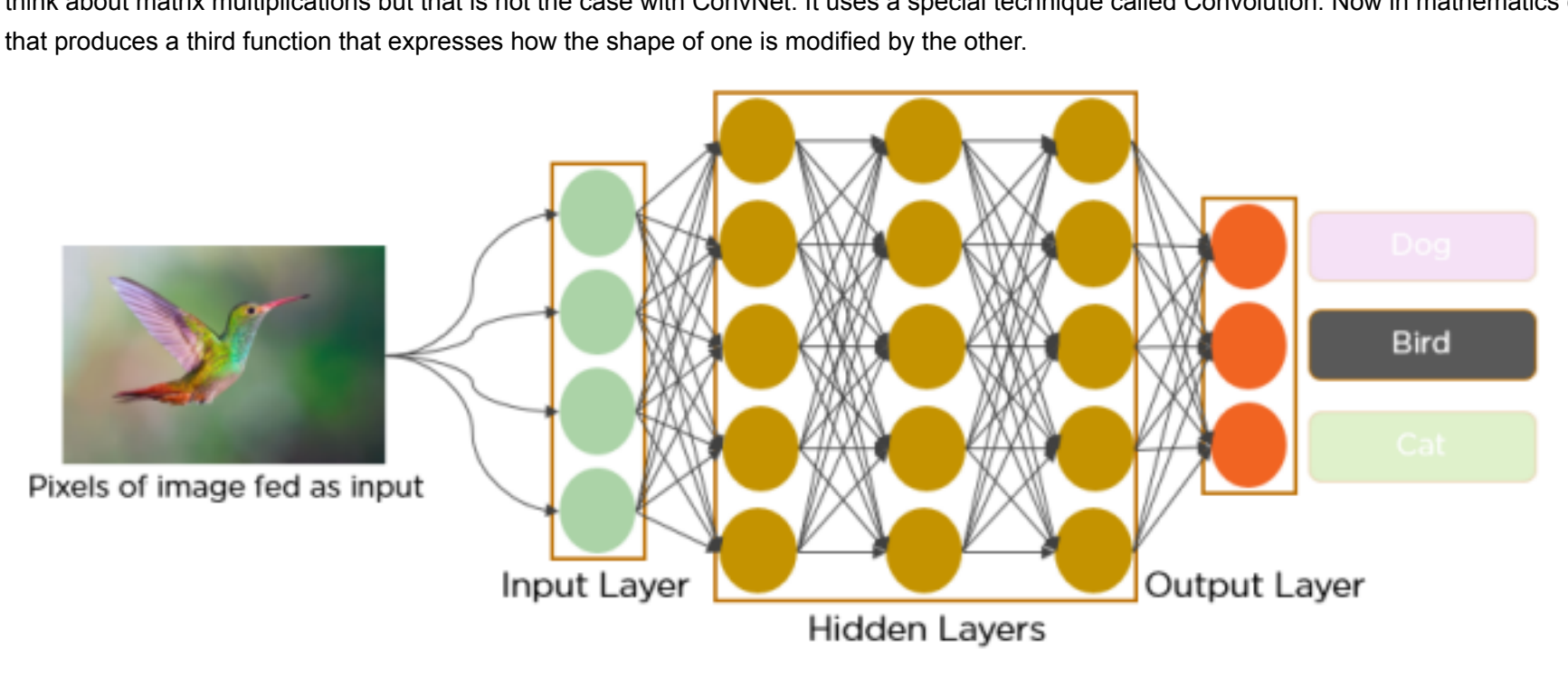
Training images and their corresponding true labels Validation images and their corresponding true labels (we use these labels only to validate the model and not during the training phase) We also define the number of epochs in this step. For starters, we will run the model for 10 epochs (you can change the number of epochs later).

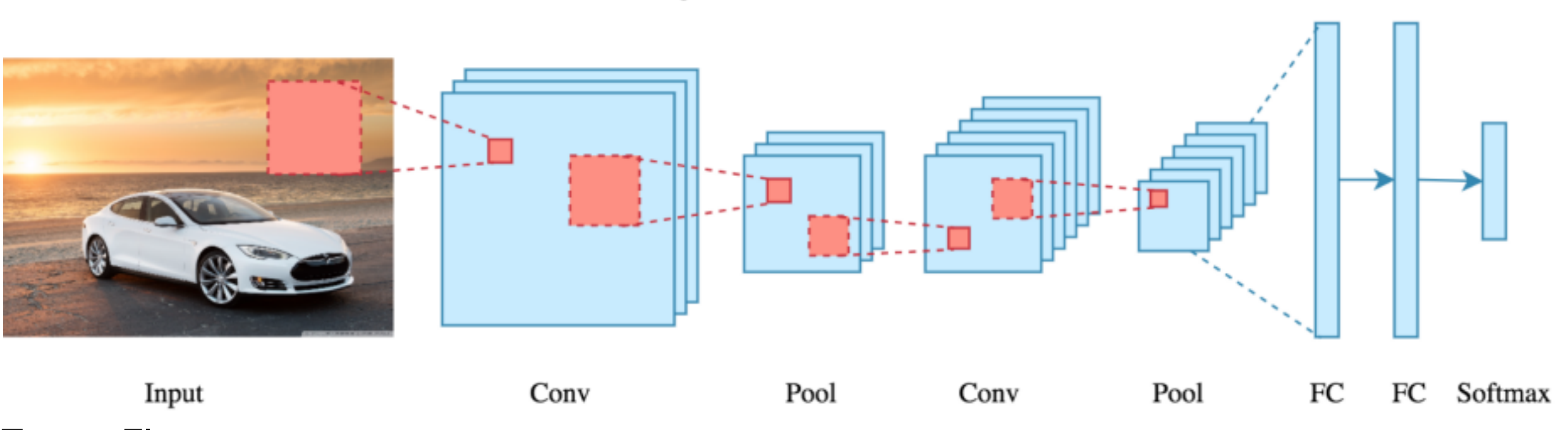
Stage 4: Estimating the model’s performance

Finally, we load the test data (images) and go through the pre-processing step here as well. We then predict the classes for these images using the trained model.

Convolutional Neural Networks

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.





Tensor Flow

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind.

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. Keras is: Simple -- but not simplistic. Keras reduces developer cognitive load to free you to focus on the parts of the problem that really matter. Flexible -- Keras adopts the principle of progressive disclosure of complexity: simple workflows should be quick and easy, while arbitrarily advanced workflows should be possible via a clear path Loading [MathJax]/extensions/Safe.js that builds upon what you've already learned. Powerful -- Keras provides industry-strength performance and scalability: it is used by organizations and companies including NASA, YouTube, or Waymo.

CIFAR-10 Dataset

CIFAR is an acronym that stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute.

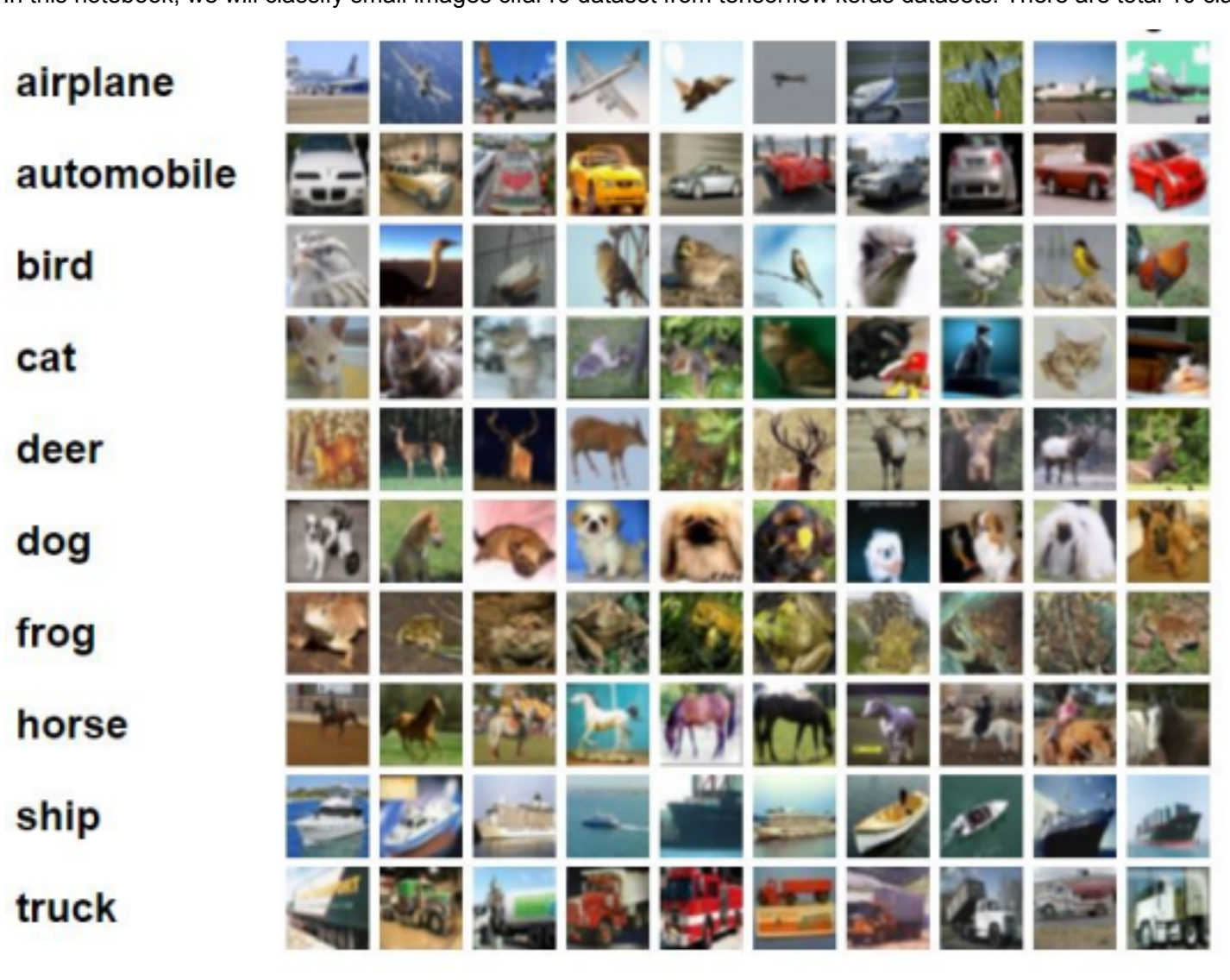
The dataset is comprised of 60,000 32×32 pixel color photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

1. airplane
2. automobile
3. bird
4. cat
5. deer
6. dog
7. frog
8. horse
9. ship
10. truck These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

CIFAR-10 is a well-understood dataset and widely used for benchmarking computer vision algorithms in the field of machine learning.The problem is “solved.” It is relatively straightforward to achieve 80% classification accuracy. Top performance on the problem is achieved by deep learning convolutional neural networks with a classification accuracy above 90% on the test dataset

Code & Output

In this notebook, we will classify small images cifar10 dataset from tensorflow keras datasets. There are total 10 classes as shown below. We will use CNN for classification



In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

Load the dataset

In [2]:
```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 7779s 46us/step
```

Out[2]:
```
(50000, 32, 32, 3)
```

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

Here we see there are 50000 training images and 1000 test images

In [4]:
```
y_train.shape
```

Out[4]:
```
(50000, 1)
```

In [5]:
```
y_train[:5]
```

Out[5]:
```
array([[6],
 [9],
 [9],
 [4],
 [1]], dtype=uint8)
```

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

In [6]:
```
y_train = y_train.reshape(-1,) y_train[:5]
```

Out[6]:
```
array([6, 9, 9, 4, 1],
dtype=uint8)
```

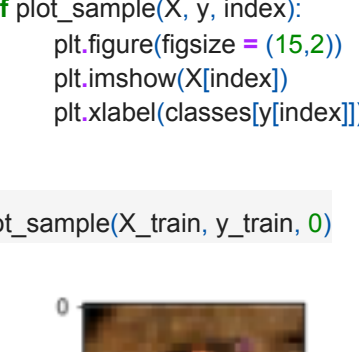
In [7]:
```
y_test = y_test.reshape(-1,)
```

In [8]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

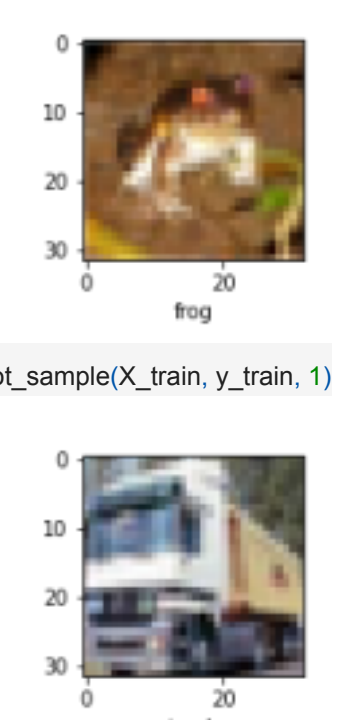
Let's plot some images to see what they are

In [9]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [10]:
```
plot_sample(X_train, y_train, 0)
```



In [12]:
```
plot_sample(X_train, y_train, 1)
```



Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to

1. Hence to normalize in 0-->1 range, we need to divide it by 255

Normalizing the training data

In [13]:
```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

In [14]:
```
ann = models.Sequential([
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(3000, activation='relu'),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 71s 44ms/step - loss: 1.8121 - accuracy: 0.3566 Epoch 2/5
1563/1563 [==============================] - 69s 44ms/step - loss: 1.6260 - accuracy: 0.4264 Epoch 3/5
1563/1563 [==============================] - 69s 44ms/step - loss: 1.5425 - accuracy: 0.4568 Epoch 4/5
1563/1563 [==============================] - 72s 46ms/step - loss: 1.4821 - accuracy: 0.4775 Epoch 5/5
1563/1563 [==============================] - 71s 45ms/step - loss: 1.4320 - accuracy: 0.4965
```

Out[14]:
```
<keras.callbacks.History at 0x1eab88c0b20>
```

You can see that at the end of 5 epochs, accuracy is at around 49%

Now let us build a convolutional neural network to train our images

In [16]:
```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),  layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
    ])
```

In [17]:
```
cnn.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
```

In [18]:
```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 26s 17ms/step - loss: 1.4344 - accuracy: 0.4887 Epoch 2/10
1563/1563 [==============================] - 24s 15ms/step - loss: 1.0976 - accuracy: 0.6162 Epoch 3/10
1563/1563 [==============================] - 23s 14ms/step - loss: 0.9728 - accuracy: 0.6627 Epoch 4/10
1563/1563 [==============================] - 23s 14ms/step - loss: 0.8888 - accuracy: 0.6909 Epoch 5/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8230 - accuracy: 0.7153 Epoch 6/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.7629 - accuracy: 0.7338 Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7118 - accuracy: 0.7519 Epoch 8/10
1563/1563 [==============================] - 26s 17ms/step - loss: 0.6648 - accuracy: 0.7669 Epoch 9/10
1563/1563 [==============================] - 26s 17ms/step - loss: 0.6219 - accuracy: 0.7824 Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.5860 - accuracy: 0.7950
```

Out[18]:
```
<keras.callbacks.History at 0x1eab851de50>
```

With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

In [19]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 2s 4ms/step - loss: 0.9481 - accuracy: 0.6971
```

Out[19]:
```
[0.9480656385421753,
0.6970999836921692]
```

In [20]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 1s 4ms/step
```

Out[20]:
```
array([[3.4695051e-03, 7.4790127e-04,
1.0699183e-03, 6.3420224e-01,  7.0188974e-04,
3.1539711e-01, 1.2989420e-03, 1.1233686e-02,
3.0647002e-02, 1.2317749e-03],
 [6.4961502e-04, 1.9408781e-02, 1.2785280e-09,
5.6119926e-09,  4.3968877e-09, 2.4492278e-11,
1.3069035e-10, 1.3648950e-10,  9.7991943e-01,
2.2179836e-05],
 [4.5123866e-01, 4.5114130e-02, 9.4693722e-05,
1.5559462e-03,  4.4963305e-04, 2.0922555e-05,
2.4143192e-04, 2.2208314e-04,  4.7349823e-01,
2.7564226e-02],
 [9.3615222e-01, 1.0264226e-03, 3.8821992e-02,
1.3830122e-03,  1.2950689e-02, 5.8792564e-05,
2.3970899e-05, 3.0163734e-04,  9.2014894e-03,
7.9743666e-05],
 [3.8150662e-05, 1.3042828e-04, 1.6568774e-02,
1.5169306e-02,  7.1870703e-01, 3.8427464e-03,
2.4537726e-01, 1.3322149e-06,  1.5945302e-04,
5.5806522e-06]], dtype=float32)
```

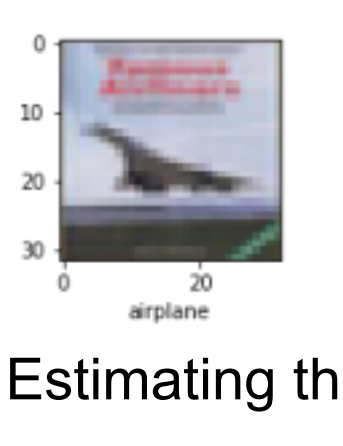
In [21]:
```
y_classes = [np.argmax(element) for element in y_pred] y_classes[:5]
```

Out[21]:
```
[3, 8, 8, 0, 4]
```

In [22]:
```
y_test[:5]
```

Out[22]:
```
array([3, 8, 8, 0, 6],
dtype=uint8)
```

In [23]:
```
plot_sample(X_test, y_test,3)
```



Estimating the model’s performance

In [24]:
```
classes[y_classes[3]]
```

Out[24]:
```
'airplane'
```

In [25]:
```
classes[y_classes[3]]
```

Out[25]:
```
'airplane'
```

Conclusion

Thus we have succesfully builded the Image classification model by dividing the model into following 4 stages Loading and preprocessing the image data,Defining the model's architecture,Training the model and Estimating the model's performance

References

In [ ]:
```
https://www.ibm.com/cloud/learn/deep-learning
https://keras.io/about/
https://github.com/codebasics/deep-learning-keras-tf-tutorial
https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
https://www.analyticsvidhya.com/blog/2019/01/build-image-classification-model-10-minutes/
```