# Small Image Classification Using Artificial Neural Network (ANN)

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 ANN for classification

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

## Load the dataset

```
(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 [==============================] - 2s 0us/step
(50000, 32, 32, 3)
```

What is the percentage of data division? 80:20? or something else? calculate

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

what is the interpretation of 32 * 31 * 3?

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

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[: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

```
y_train = y_train.reshape(-1,)
y_train
```

```
array([6, 9, 9, ..., 9, 1, 1], dtype=uint8)
```

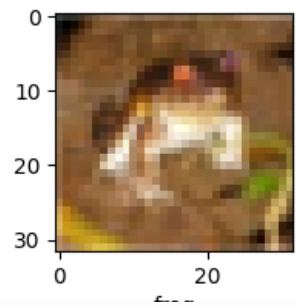
```
y_test = y_test.reshape(-1,)
```

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

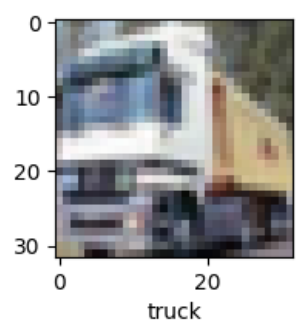
Let's plot some images to see what they are

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

```
plot_sample(X_train, y_train, 0)
```



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

### Normalizing the training data

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

### Build simple artificial neural network for image classification

```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(100, activation='relu'),
        layers.Dense(100, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

```
# Parameter Calculations
3072*100
100*100
10000*10
307200 + 10000 +1000 +100+ 100+ 10
```

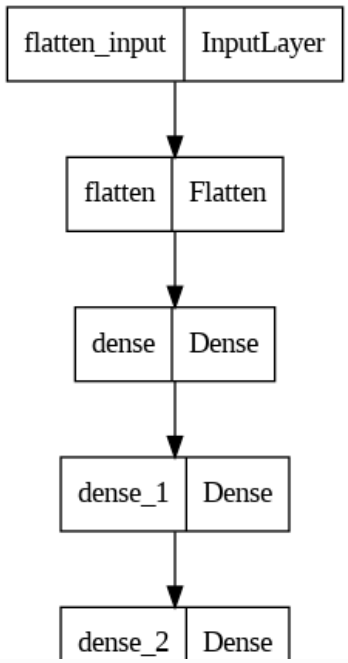
```
318410
```

```
ann.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 100)               307300

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 318410 (1.21 MB)
```

```
Trainable params: 318410 (1.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import pydot
from tensorflow import keras
keras.utils.plot_model(ann)
```




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

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

```
Epoch 1/5
1563/1563 [==============================] - 14s 8ms/step - loss: 1.8985 - accuracy: 0.3169
Epoch 2/5
1563/1563 [==============================] - 15s 10ms/step - loss: 1.7177 - accuracy: 0.3922
Epoch 3/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.6386 - accuracy: 0.4174
Epoch 4/5
1563/1563 [==============================] - 7s 4ms/step - loss: 1.5828 - accuracy: 0.4371
Epoch 5/5
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5468 - accuracy: 0.4530
<keras.src.callbacks.History at 0x7cf1426227a0>
```

**You can see that at the end of 5 epochs, accuracy is at around 45.3%**

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]
print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
313/313 [==============================] - 2s 5ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.47      0.58      0.52      1000
           1       0.55      0.55      0.55      1000
           2       0.35      0.29      0.32      1000
           3       0.41      0.17      0.24      1000
           4       0.44      0.31      0.37      1000
           5       0.33      0.43      0.37      1000
           6       0.55      0.38      0.45      1000
           7       0.49      0.48      0.49      1000
           8       0.62      0.47      0.54      1000
           9       0.36      0.73      0.48      1000
```

```
              accuracy                           0.44     10000
             macro avg       0.46      0.44      0.43     10000
          weighted avg       0.46      0.44      0.43     10000
```

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.

The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y_test.