Image Classification Using Convolutional Neural Network (CNN)

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

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
```

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)
```

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

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

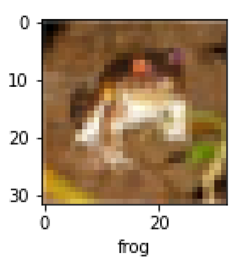
In [7]:
```
y_test = y_test.reshape(-1,)
```

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

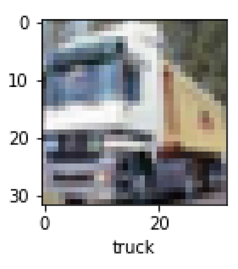
Let's plot some images to see what they are

In [19]:
```
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 [11]:
```
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

In [12]:
```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

In [13]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

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

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

```
Epoch 1/5
1563/1563 [==============================] - 51s 32ms/step - loss: 1.8545 - accuracy: 0.3375
Epoch 2/5
1563/1563 [==============================] - 54s 34ms/step - loss: 1.6554 - accuracy: 0.4134
Epoch 3/5
1563/1563 [==============================] - 56s 36ms/step - loss: 1.5677 - accuracy: 0.4453
Epoch 4/5
1563/1563 [==============================] - 58s 37ms/step - loss: 1.5060 - accuracy: 0.4679
Epoch 5/5
1563/1563 [==============================] - 56s 36ms/step - loss: 1.4543 - accuracy: 0.4857
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x1553ea70100>
```

You can see that at the end of 5 epochs, accuracy is at around 48.48%

In [14]:
```
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))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.69      0.36      0.48      1000
           1       0.67      0.47      0.55      1000
           2       0.31      0.46      0.37      1000
           3       0.41      0.18      0.25      1000
           4       0.58      0.19      0.29      1000
           5       0.35      0.48      0.40      1000
           6       0.42      0.69      0.52      1000
           7       0.50      0.60      0.55      1000
           8       0.60      0.65      0.62      1000
           9       0.52      0.63      0.57      1000

    accuracy                           0.47     10000
   macro avg       0.50      0.47      0.46     10000
weighted avg       0.50      0.47      0.46     10000
```

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

In [15]:
```
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 [16]:
```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [17]:
```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 28s 18ms/step - loss: 1.5493 - accuracy: 0.4347
Epoch 2/10
1563/1563 [==============================] - 29s 18ms/step - loss: 1.1902 - accuracy: 0.5804
Epoch 3/10
1563/1563 [==============================] - 29s 18ms/step - loss: 1.0403 - accuracy: 0.6365
Epoch 4/10
1563/1563 [==============================] - 28s 18ms/step - loss: 0.9540 - accuracy: 0.6657
Epoch 5/10
1563/1563 [==============================] - 28s 18ms/step - loss: 0.8957 - accuracy: 0.6895
Epoch 6/10
1563/1563 [==============================] - 28s 18ms/step - loss: 0.8458 - accuracy: 0.7054
Epoch 7/10
1563/1563 [==============================] - 29s 18ms/step - loss: 0.7996 - accuracy: 0.7233
Epoch 8/10
1563/1563 [==============================] - 29s 18ms/step - loss: 0.7618 - accuracy: 0.7357
Epoch 9/10
1563/1563 [==============================] - 28s 18ms/step - loss: 0.7297 - accuracy: 0.7472
Epoch 10/10
1563/1563 [==============================] - 29s 18ms/step - loss: 0.6960 - accuracy: 0.7577
```

Out[17]:
```
<tensorflow.python.keras.callbacks.History at 0x15542680940>
```

With CNN, at the end 5 epochs, accuracy was at around 70.28% 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 [18]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.8994 - accuracy: 0.6999
```

Out[18]:
```
[0.8994082808494568, 0.6998999714851379]
```

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

Out[20]:
```
array([[4.42408746e-05, 1.06172280e-04, 8.05528325e-05, 8.93471360e-01,
        4.60884658e-05, 1.71734765e-02, 2.40301574e-03, 5.91494518e-06,
        8.66578966e-02, 1.12836124e-05],
       [2.69579527e-04, 6.17878395e-04, 4.70100076e-07, 3.47208538e-06,
        1.63589590e-07, 5.12187590e-08, 4.46268693e-08, 1.21474519e-09,
        9.98918056e-01, 1.90344072e-04],
       [1.81118101e-02, 5.99340200e-02, 9.22011022e-05, 3.77155794e-03,
        1.79968003e-04, 3.02464003e-04, 7.47733357e-05, 2.80438398e-04,
        8.77298295e-01, 3.99544500e-02],
       [6.84610546e-01, 1.85087021e-03, 2.37693498e-03, 8.73076438e-04,
        7.01270439e-03, 5.81853128e-05, 2.94850534e-03, 1.31227702e-04,
        2.99838215e-01, 2.99813953e-04],
       [2.46152172e-06, 2.06649402e-05, 2.84170592e-03, 3.15380283e-02,
        1.09281674e-01, 3.80036957e-03, 8.52199316e-01, 3.70471230e-06,
        3.10928473e-04, 1.09269570e-06]], dtype=float32)
```

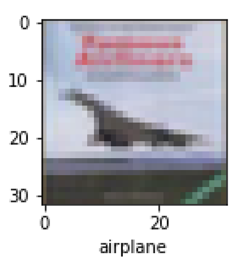
In [21]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[21]:
```
[3, 8, 8, 0, 6]
```

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

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

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



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

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

In [ ]:
```

```