# NAME - PIYUSHA PARDESHI

# BATCH - PGA21 PUNE BATCH

# INSTITUTE - IMARTICUS

## IMPORT LIBRARIES

In [1]:
```
#import tensorflow and keras libs
import tensorflow as tf
import keras
from keras.datasets import cifar10

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


#import CNN libs
from keras.models import Sequential
from keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense
from tensorflow.keras import datasets, layers, models

from sklearn.metrics import confusion_matrix , classification_report
```

## DOWNLOAD THE DATA (CIFAR10)

In [2]:
```
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

In [3]:
```
print("Train shape:","Train X:",x_train.shape,"Train Y:",y_train.shape)
print("Test shape:","Test X:",x_test.shape,"Test Y:",y_test.shape)
```

```
Train shape: Train X: (50000, 32, 32, 3) Train Y: (50000, 1)
Test shape: Test X: (10000, 32, 32, 3) Test Y: (10000, 1)
```

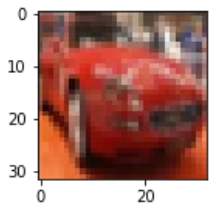
In [4]:
```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
50000 train samples
10000 test samples
```

## LETS VISUALIZE SOME IMAGES : EDA

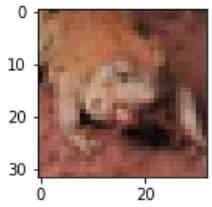
In [5]:
```
plt.figure(figsize=(15,2))
plt.imshow(x_train[5])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7fba50f14110>
```



In [6]:
```
plt.figure(figsize=(15,2))
plt.imshow(x_test[5])
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7fba50a5c550>
```




## CONVERTING 2D DATA TO 1D

In [7]:
```
y_train= y_train.reshape(-1,)
pd.unique(y_train)
```

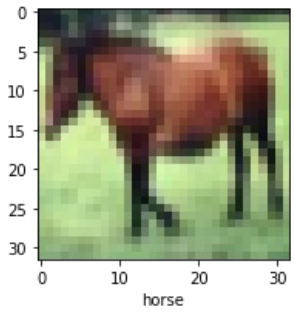
Out[7]:
```
array([6, 9, 4, 1, 2, 7, 8, 3, 5, 0], dtype=uint8)
```

## LETS PLOT SOME RANDOM IMAGES

In [8]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
num_classes = 10

def plot_image(x,y,index):
  fig = plt.figure(figsize=(15,3))
  plt.imshow(x[index])
  plt.xlabel(class_names[y[index]])
```

In [9]:
```
plot_image(x_train,y_train,7)
```




In [10]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
num_classes = 10

fig = plt.figure(figsize=(10,5))
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    plt.imshow(x_train[i])
    ax.set_title(class_names[y_train[i]])
plt.show()
```

## LETS RESCALE THE VALUES

In [11]:
```
x_train = x_train/255
x_test = x_test/255
```

In [12]:
```
x_train= x_train.astype('float32')
x_test= x_test.astype('float32')
```

## LETS BUILD A SIMPLE ANN

### BUILD MODEL BY CREATING LAYERS

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='softmax')
    ])
```

### COMPILE ANN MODEL

In [14]:
```
ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

### FIT DATA ON ANN MODEL

In [15]:
```
history = ann.fit(x_train, y_train, epochs=15,validation_data=(x_test, y_test))
```

```
Epoch 1/15
1563/1563 [==============================] - 103s 66ms/step - loss: 1.8069 - accuracy: 0.3550 - val_loss: 1.6804
- val_accuracy: 0.4078
Epoch 2/15
1563/1563 [==============================] - 103s 66ms/step - loss: 1.6193 - accuracy: 0.4279 - val_loss: 1.6255
- val_accuracy: 0.4172
Epoch 3/15
1563/1563 [==============================] - 103s 66ms/step - loss: 1.5408 - accuracy: 0.4566 - val_loss: 1.6081
- val_accuracy: 0.4239
Epoch 4/15
1563/1563 [==============================] - 103s 66ms/step - loss: 1.4807 - accuracy: 0.4780 - val_loss: 1.4932
- val_accuracy: 0.4686
Epoch 5/15
1563/1563 [==============================] - 105s 67ms/step - loss: 1.4323 - accuracy: 0.4943 - val_loss: 1.4950
- val_accuracy: 0.4768
Epoch 6/15
1563/1563 [==============================] - 106s 68ms/step - loss: 1.3885 - accuracy: 0.5106 - val_loss: 1.5358
- val_accuracy: 0.4497
Epoch 7/15
1563/1563 [==============================] - 107s 68ms/step - loss: 1.3524 - accuracy: 0.5258 - val_loss: 1.5554
- val_accuracy: 0.4645
Epoch 8/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.3156 - accuracy: 0.5371 - val_loss: 1.3973
- val_accuracy: 0.5038
Epoch 9/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.2856 - accuracy: 0.5479 - val_loss: 1.3904
- val_accuracy: 0.5112
Epoch 10/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.2527 - accuracy: 0.5611 - val_loss: 1.4058
- val_accuracy: 0.5067
Epoch 11/15
1563/1563 [==============================] - 110s 70ms/step - loss: 1.2232 - accuracy: 0.5714 - val_loss: 1.4669
- val_accuracy: 0.4851
```

```
Epoch 12/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.1966 - accuracy: 0.5805 - val_loss: 1.4753
- val_accuracy: 0.4751
Epoch 13/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.1672 - accuracy: 0.5915 - val_loss: 1.3609
- val_accuracy: 0.5122
Epoch 14/15
1563/1563 [==============================] - 108s 69ms/step - loss: 1.1409 - accuracy: 0.6030 - val_loss: 1.3372
- val_accuracy: 0.5284
Epoch 15/15
1563/1563 [==============================] - 107s 69ms/step - loss: 1.1124 - accuracy: 0.6136 - val_loss: 1.3563
- val_accuracy: 0.5189
```

## ANN : EVALUATION ON TEST DATA

In [16]:
```
ann.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 8s 24ms/step - loss: 1.3563 - accuracy: 0.5189
```

Out[16]:
```
[1.3562846183776855, 0.5188999772071838]
```

## CLASSIFICATION REPORT OF ANN MODEL

In [17]:
```
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       0.56      0.61      0.58      1000
           1       0.78      0.47      0.59      1000
           2       0.37      0.47      0.41      1000
           3       0.42      0.31      0.36      1000
           4       0.43      0.51      0.47      1000
           5       0.54      0.31      0.39      1000
           6       0.48      0.72      0.57      1000
           7       0.71      0.48      0.57      1000
           8       0.53      0.76      0.62      1000
           9       0.61      0.56      0.58      1000

    accuracy                           0.52     10000
   macro avg       0.54      0.52      0.51     10000
weighted avg       0.54      0.52      0.51     10000
```

## PREDICTION OF DATA ON ANN

In [18]:
```
# Storing Result in a Data Frame
df =pd.DataFrame({'Actual':y_test.reshape(-1,),'Predicted':y_pred_classes})
df.head(10)
```

Out[18]:

| | Actual | Predicted |
|---|---|---|
| **0** | 3 | 3 |
| **1** | 8 | 8 |
| **2** | 8 | 8 |
| **3** | 0 | 0 |
| **4** | 6 | 4 |
| **5** | 6 | 6 |
| **6** | 1 | 1 |
| **7** | 6 | 6 |
| **8** | 3 | 5 |
| **9** | 1 | 1 |

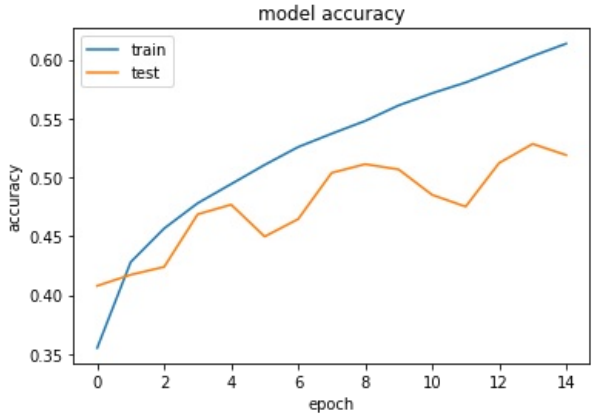
In [19]:
```
y_test= y_test.reshape(-1,)
y_test
```

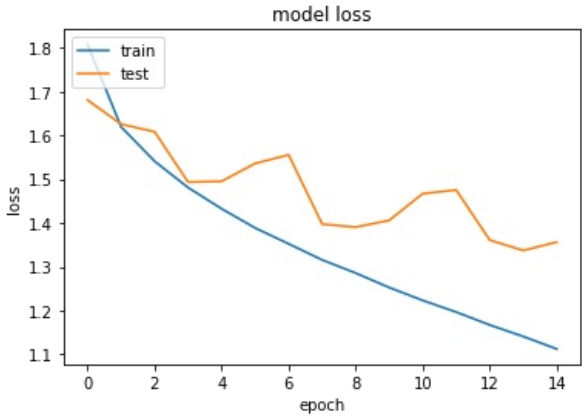
Out[19]:
```
array([3, 8, 8, ..., 5, 1, 7], dtype=uint8)
```

## TRAINING AND VALIDATION CURVES FOR CNN

In [20]:
```
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




## LETS BUILD A CNN ARCHITECHTURE WITH KERAS AND STACK AN ANN ON TOP OF CNN

## BUILD CNN MODEL

In [21]:
```
# creating multiple layers like convolution 2d ,max pooling then add dense layers to the model
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')
])
```

## COMPILE CNN MODEL

In [22]:

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

## FIT CNN MODEL

In [23]:

```
history = cnn.fit(x_train, y_train, epochs=20,validation_data=(x_test, y_test))
```

```
Epoch 1/20
1563/1563 [==============================] - 66s 42ms/step - loss: 1.4833 - accuracy: 0.4654 - val_loss: 1.2139 -
val_accuracy: 0.5745
Epoch 2/20
1563/1563 [==============================] - 65s 41ms/step - loss: 1.1447 - accuracy: 0.5986 - val_loss: 1.0907 -
val_accuracy: 0.6131
Epoch 3/20
1563/1563 [==============================] - 64s 41ms/step - loss: 1.0071 - accuracy: 0.6494 - val_loss: 1.0131 -
val_accuracy: 0.6473
Epoch 4/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.9210 - accuracy: 0.6827 - val_loss: 1.0042 -
val_accuracy: 0.6491
Epoch 5/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8537 - accuracy: 0.7042 - val_loss: 0.9637 -
val_accuracy: 0.6663
Epoch 6/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8015 - accuracy: 0.7220 - val_loss: 0.9067 -
val_accuracy: 0.6891
Epoch 7/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7486 - accuracy: 0.7401 - val_loss: 0.9734 -
val_accuracy: 0.6710
Epoch 8/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7127 - accuracy: 0.7528 - val_loss: 0.8981 -
val_accuracy: 0.6985
Epoch 9/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6748 - accuracy: 0.7668 - val_loss: 0.9212 -
val_accuracy: 0.6945
Epoch 10/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6411 - accuracy: 0.7764 - val_loss: 0.9437 -
val_accuracy: 0.6843
Epoch 11/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6107 - accuracy: 0.7859 - val_loss: 0.9265 -
val_accuracy: 0.6970
Epoch 12/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5826 - accuracy: 0.7960 - val_loss: 0.9865 -
val_accuracy: 0.6861
Epoch 13/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5536 - accuracy: 0.8073 - val_loss: 0.9504 -
val_accuracy: 0.6989
Epoch 14/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5303 - accuracy: 0.8141 - val_loss: 1.0220 -
val_accuracy: 0.6864
Epoch 15/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5036 - accuracy: 0.8234 - val_loss: 1.0864 -
val_accuracy: 0.6793
Epoch 16/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.4773 - accuracy: 0.8324 - val_loss: 1.0519 -
val_accuracy: 0.6913
Epoch 17/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.4636 - accuracy: 0.8384 - val_loss: 1.1298 -
val_accuracy: 0.6779
Epoch 18/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.4364 - accuracy: 0.8450 - val_loss: 1.1287 -
val_accuracy: 0.6826
Epoch 19/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.4132 - accuracy: 0.8551 - val_loss: 1.1661 -
val_accuracy: 0.6838
Epoch 20/20
1563/1563 [==============================] - 64s 41ms/step - loss: 0.3961 - accuracy: 0.8606 - val_loss: 1.1904 -
val_accuracy: 0.6815
```

# EVALUATION ON TEST DATA : CNN

In [24]:
```
cnn.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 4s 12ms/step - loss: 1.1904 - accuracy: 0.6815
```

Out[24]:
```
[1.1904023885726929, 0.6815000176429749]
```

In [25]:
```
y_pred = cnn.predict(x_test)
y_pred[:5]
```

Out[25]:
```
array([[8.4963575e-04, 1.4090250e-04, 1.0169011e-05, 1.5265533e-01,
        2.3785881e-04, 2.3303512e-03, 7.0449775e-03, 7.2383036e-06,
        8.3670717e-01, 1.6404621e-05],
       [1.2921882e-03, 6.6815257e-02, 3.7760461e-09, 9.3280299e-09,
        9.2885943e-10, 1.7046051e-10, 2.7966945e-12, 9.8067013e-11,
        9.2992717e-01, 1.9653121e-03],
       [6.0373440e-02, 1.5120710e-01, 4.7338582e-04, 1.8537648e-03,
        3.3062755e-04, 1.1910455e-03, 3.3537890e-05, 3.5853189e-04,
        6.2651652e-01, 1.5766205e-01],
       [9.4306952e-01, 9.5333653e-03, 2.8318915e-05, 1.1187425e-05,
        9.3208087e-08, 1.7037218e-08, 5.3859145e-07, 2.4140143e-09,
        4.7352973e-02, 3.9398496e-06],
       [1.1938706e-10, 7.8104176e-06, 3.7399858e-02, 9.6684863e-04,
        6.1001847e-03, 6.5603024e-05, 9.5532256e-01, 3.1421126e-07,
        1.3086357e-04, 6.0319817e-06]], dtype=float32)
```

In [26]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[26]:
```
[8, 8, 8, 0, 6]
```

In [27]:
```
y_test[:5]
```

Out[27]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [28]:
```
#plot_image(x_test, y_test,3)
```

In [29]:
```
class_names[y_classes[3]]
```

Out[29]:
```
'airplane'
```

# CLASSIFICATION ON CNN MODEL

In [30]:
```
y_pred = cnn.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       0.73      0.73      0.73      1000
           1       0.74      0.82      0.78      1000
           2       0.55      0.61      0.58      1000
           3       0.54      0.41      0.47      1000
           4       0.66      0.62      0.64      1000
           5       0.59      0.58      0.58      1000
           6       0.73      0.76      0.74      1000
           7       0.75      0.72      0.74      1000
```

```
           8       0.76      0.81      0.78      1000
           9       0.73      0.77      0.75      1000

    accuracy                           0.68     10000
   macro avg       0.68      0.68      0.68     10000
weighted avg       0.68      0.68      0.68     10000
```

## PREDICTIO OF DATA ON CNN

In [31]:
```
# Storing Result in a Data Frame
df =pd.DataFrame({'Actual':y_test.reshape(-1,),'Predicted':y_pred_classes})
df
```

Out[31]:

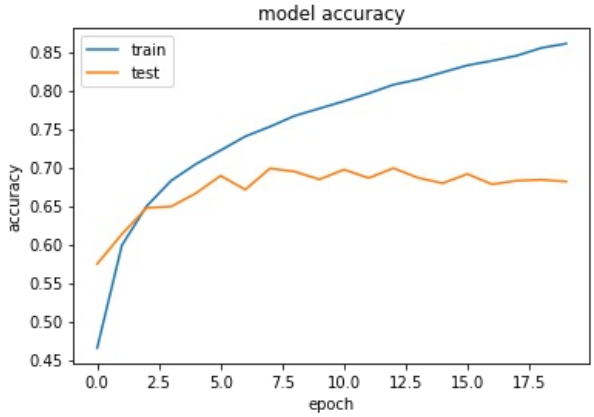
| | Actual | Predicted |
|---|---|---|
| **0** | 3 | 8 |
| **1** | 8 | 8 |
| **2** | 8 | 8 |
| **3** | 0 | 0 |
| **4** | 6 | 6 |
| **...** | ... | ... |
| **9995** | 8 | 3 |
| **9996** | 3 | 3 |
| **9997** | 5 | 2 |
| **9998** | 1 | 1 |
| **9999** | 7 | 7 |

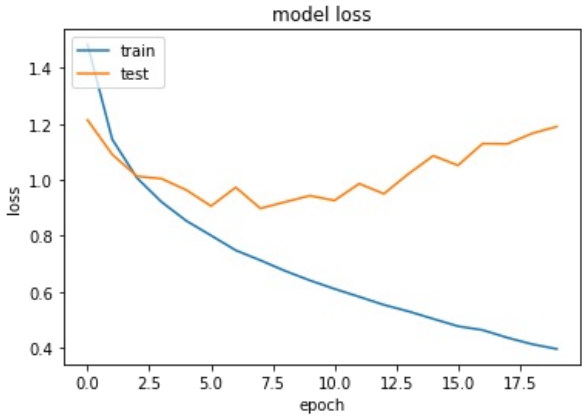
10000 rows × 2 columns

## TRAINING AND VALIDATION CURVES FOR CNN

In [32]:
```
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```






# CONCLUSION

IMAGE CLASSIFICATION IN DEEP LEARNING HAS HELP US PREDICT THE IMAGE NAME BY BUILDING MODELS USING ANN AND CNN TECHNIQUES THIS SHOWED US THAT THE ADDITION OF MULTIPLE LAYERS IN MODEL MAKES THE COMPUTATION SPEED FASTER AS WELL AS THE PERFORMANCE ALSO INCREASES WHEN MODELS ARE STACKED ON EACH OTHER A MODEL WITH BETTER PREDICTION CAN BE USED FURTHER WHICH IS CNN WHICH HAS GIVEN ACCURACY UPTO 86 % WHICH SEEMS QUITE GOOD.

In [32]: