# Acmegrade Artificial Intelligence Project

**Name: Dev Patel** **E-mail id: devpatel0952@gmail.com**

**Topic:** Image recognition using CNN on CIFAR-10 Dataset

## Theory:

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.

## Dataset:

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The 10 classes include- airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

## Steps:

1. Required libraries were imported and TensorFlow version was checked.
2. Data set was directly loaded from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
3. Dataset was split into train and test sets and their image format was checked.
4. The images were processed by reducing pixel values and flattening the label values.
5. The CNN model was developed.
6. The graphs for accuracy and validation accuracy were plotted.

## Results: **The validation accuracy starts to remain stable at about 82.94%.**

```
Epoch 1/15
1563/1563 [==============================] - 195s 125ms/step - loss: 0.7389 - accuracy: 0.7449 - val_loss: 0.7412 - val_accuracy: 0.7456
Epoch 2/15
1563/1563 [==============================] - 213s 136ms/step - loss: 0.6179 - accuracy: 0.7882 - val_loss: 0.6537 - val_accuracy: 0.7802
Epoch 3/15
1563/1563 [==============================] - 201s 128ms/step - loss: 0.5276 - accuracy: 0.8189 - val_loss: 0.6185 - val_accuracy: 0.7985
Epoch 4/15
1563/1563 [==============================] - 196s 125ms/step - loss: 0.4404 - accuracy: 0.8485 - val_loss: 0.6239 - val_accuracy: 0.8002
Epoch 5/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.3750 - accuracy: 0.8699 - val_loss: 0.6479 - val_accuracy: 0.8026
Epoch 6/15
1563/1563 [==============================] - 203s 130ms/step - loss: 0.3195 - accuracy: 0.8895 - val_loss: 0.5767 - val_accuracy: 0.8241
Epoch 7/15
1563/1563 [==============================] - 201s 128ms/step - loss: 0.2746 - accuracy: 0.9043 - val_loss: 0.6336 - val_accuracy: 0.8168
Epoch 8/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.2412 - accuracy: 0.9167 - val_loss: 0.6132 - val_accuracy: 0.8191
Epoch 9/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.2026 - accuracy: 0.9303 - val_loss: 0.6359 - val_accuracy: 0.8228
Epoch 10/15
1563/1563 [==============================] - 200s 128ms/step - loss: 0.1864 - accuracy: 0.9357 - val_loss: 0.6860 - val_accuracy: 0.8111
Epoch 11/15
1563/1563 [==============================] - 199s 127ms/step - loss: 0.1581 - accuracy: 0.9458 - val_loss: 0.7000 - val_accuracy: 0.8328
Epoch 12/15
1563/1563 [==============================] - 202s 129ms/step - loss: 0.1469 - accuracy: 0.9491 - val_loss: 0.7544 - val_accuracy: 0.8225
Epoch 13/15
1563/1563 [==============================] - 203s 130ms/step - loss: 0.1381 - accuracy: 0.9543 - val_loss: 0.7131 - val_accuracy: 0.8286
Epoch 14/15
1563/1563 [==============================] - 201s 129ms/step - loss: 0.1326 - accuracy: 0.9563 - val_loss: 0.7167 - val_accuracy: 0.8299
Epoch 15/15
1563/1563 [==============================] - 206s 132ms/step - loss: 0.1173 - accuracy: 0.9600 - val_loss: 0.7012 - val_accuracy: 0.8294
```

# Importing the libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout, BatchNormalization, MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.models import Model
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.10.0'
```

# Loading the dataset and splitting to train and test set

In [3]:

```
data = tf.keras.datasets.cifar10

(X_train, y_train), (X_test, y_test) = data.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 213s 1us/step
```

In [4]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

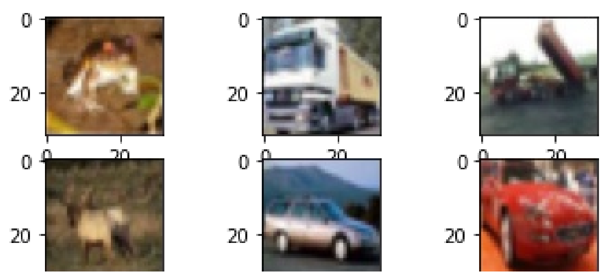
```
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

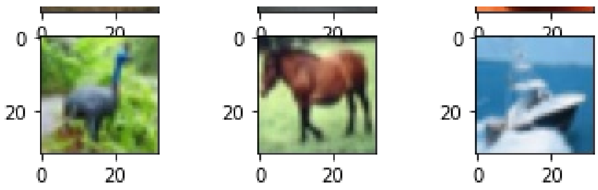
In [5]:

```
# Visualise the dataset

fig, ax = plt.subplots(3, 3)
k = 0

for i in range(3):
    for j in range(3):
        ax[i][j].imshow(X_train[k])
        k += 1
plt.show()
```





## Pre-process the data

In [6]:

```
# Reducing the pixel values and then flattenning the label values
X_train, X_test = X_train / 255.0, X_test / 255.0
y_train, y_test = y_train.flatten(), y_test.flatten()
```

In [7]:

```
# Finding total number of classes
n = len(set(y_train))

print("number of classes:", n)
```

```
number of classes: 10
```

## Build the model using the functional API

In [8]:

```
# Input layer
i = Input(shape=X_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)
```

In [9]:

```
# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
```

In [10]:

```
# Output layer
x = Dense(n, activation='softmax')(x)
```

In [11]:

```
model = Model(i, x)
model.summary()
```

```
Model: "model"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 10)                10250

=================================================================
Total params: 2,397,226
Trainable params: 2,396,330
Non-trainable params: 896
_________________________________________________________________
```

In [12]:

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

# Fitting the model

In [14]:

```
history_cnn = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15)
```

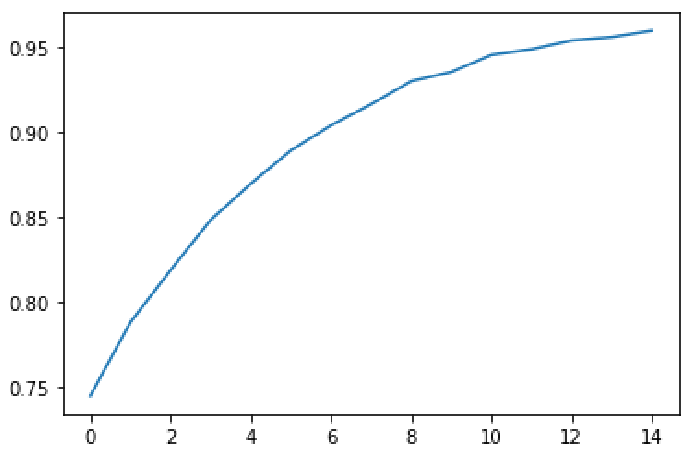
```
Epoch 1/15
1563/1563 [==============================] - 195s 125ms/step - loss: 0.7389 - accuracy: 0
.7449 - val_loss: 0.7412 - val_accuracy: 0.7456
Epoch 2/15
1563/1563 [==============================] - 213s 136ms/step - loss: 0.6179 - accuracy: 0
.7882 - val_loss: 0.6537 - val_accuracy: 0.7802
Epoch 3/15
1563/1563 [==============================] - 201s 128ms/step - loss: 0.5276 - accuracy: 0
.8189 - val_loss: 0.6185 - val_accuracy: 0.7985
Epoch 4/15
1563/1563 [==============================] - 196s 125ms/step - loss: 0.4404 - accuracy: 0
.8485 - val_loss: 0.6239 - val_accuracy: 0.8002
Epoch 5/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.3750 - accuracy: 0
.8699 - val_loss: 0.6479 - val_accuracy: 0.8026
Epoch 6/15
1563/1563 [==============================] - 203s 130ms/step - loss: 0.3195 - accuracy: 0
.8895 - val_loss: 0.5767 - val_accuracy: 0.8241
Epoch 7/15
1563/1563 [==============================] - 201s 128ms/step - loss: 0.2746 - accuracy: 0
.9043 - val_loss: 0.6336 - val_accuracy: 0.8168
Epoch 8/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.2412 - accuracy: 0
.9167 - val_loss: 0.6132 - val_accuracy: 0.8191
Epoch 9/15
1563/1563 [==============================] - 198s 127ms/step - loss: 0.2026 - accuracy: 0
.9303 - val_loss: 0.6359 - val_accuracy: 0.8228
Epoch 10/15
1563/1563 [==============================] - 200s 128ms/step - loss: 0.1864 - accuracy: 0
.9357 - val_loss: 0.6860 - val_accuracy: 0.8111
Epoch 11/15
1563/1563 [==============================] - 199s 127ms/step - loss: 0.1581 - accuracy: 0
.9458 - val_loss: 0.7000 - val_accuracy: 0.8328
Epoch 12/15
1563/1563 [==============================] - 202s 129ms/step - loss: 0.1469 - accuracy: 0
.9491 - val_loss: 0.7544 - val_accuracy: 0.8225
Epoch 13/15
1563/1563 [==============================] - 203s 130ms/step - loss: 0.1381 - accuracy: 0
.9543 - val_loss: 0.7131 - val_accuracy: 0.8286
Epoch 14/15
1563/1563 [==============================] - 201s 129ms/step - loss: 0.1326 - accuracy: 0
.9563 - val_loss: 0.7167 - val_accuracy: 0.8299
Epoch 15/15
1563/1563 [==============================] - 206s 132ms/step - loss: 0.1173 - accuracy: 0
.9600 - val_loss: 0.7012 - val_accuracy: 0.8294
```

In [15]:

```
plt.plot(history_cnn.history['accuracy'])
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0x18ff6397a30>]
```

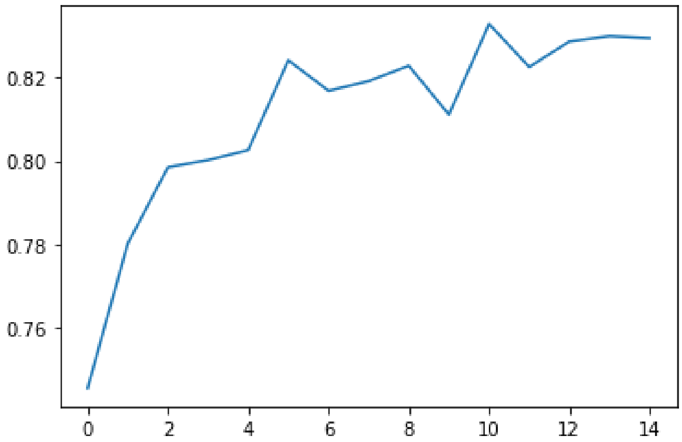



In [16]:

```
plt.plot(history_cnn.history['val_accuracy'])
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x18ff634f490>]
```




In [ ]: