# Assessment 3

### Problem 2: Write a program to implement the new CNN model. The model should contains following things (use any grayscale dataset with the 10 classes).

In [ ]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from keras.datasets import cifar10
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from keras.models import Sequential
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship
', 'truck']
```

In [ ]:

```
fig, axes = plt.subplots(ncols=5, nrows=2, figsize=(17, 8))
index = 0
for i in range(2):
    for j in range(5):
        axes[i,j].set_title(labels[y_train[index][0]])
        axes[i,j].imshow(X_train[index])
        axes[i,j].get_xaxis().set_visible(False)
        axes[i,j].get_yaxis().set_visible(False)
        index += 1
plt.show()
```

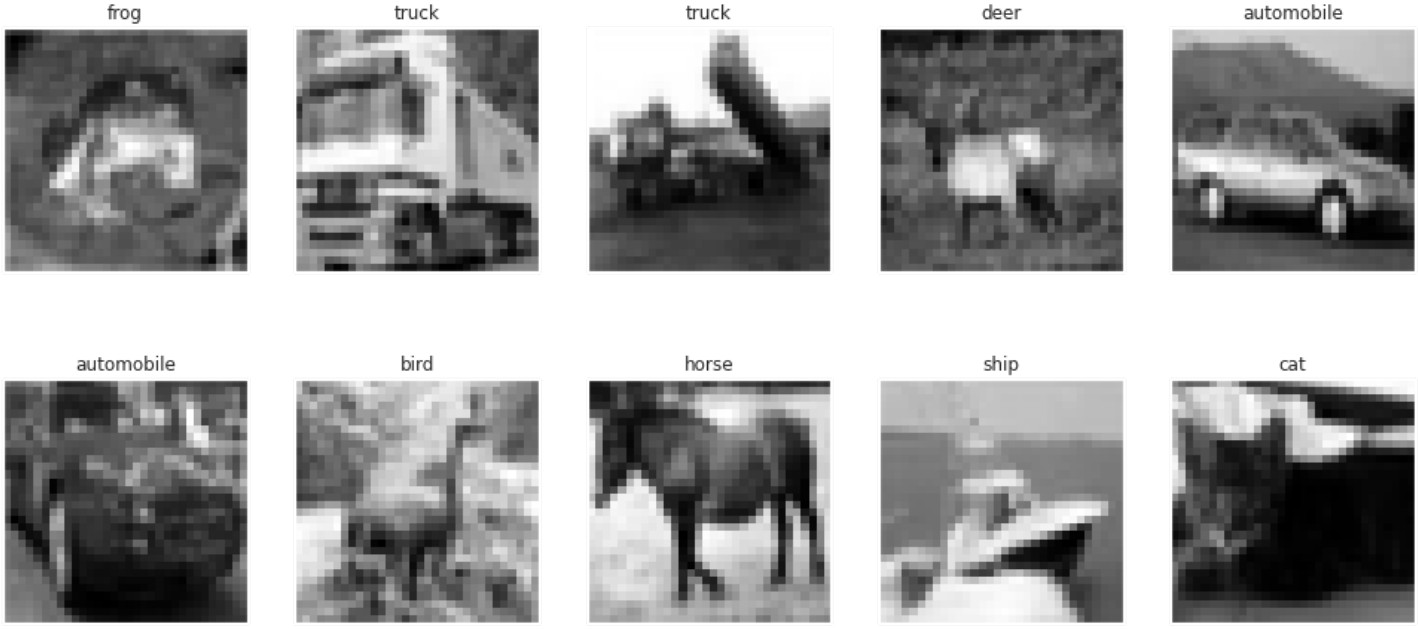

In [ ]:

```
import cv2
X_train = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_train])
X_test = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_test])
```

In [ ]:

```
fig, axes = plt.subplots(ncols=5, nrows=2, figsize=(17, 8))
```

```
fig, axes = plt.subplots(ncols=5, nrows=2, figsize=(17, 8))
index = 0
for i in range(2):
    for j in range(5):
        axes[i,j].set_title(labels[y_train[index][0]])
        axes[i,j].imshow(X_train[index], cmap='gray')
        axes[i,j].get_xaxis().set_visible(False)
        axes[i,j].get_yaxis().set_visible(False)
        index += 1
plt.show()
```




In [ ]:

```
X_train  = X_train/255
X_test  = X_test/255
```

In [ ]:

```
one_hot_encoder = OneHotEncoder(sparse=False)
one_hot_encoder.fit(y_train)
```

Out[ ]:

```
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,
              handle_unknown='error', sparse=False)
```

In [ ]:

```
y_train = one_hot_encoder.transform(y_train)
y_test = one_hot_encoder.transform(y_test)
```

In [ ]:

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
X_train.shape
```

Out[ ]:

```
(50000, 32, 32, 1)
```

In [ ]:

```
input_shape = (X_train.shape[1], X_train.shape[2], 1)
input_shape
```

Out[ ]:

```
(32, 32, 1)
```

## Model

## Model

***Convolution 1 --->input(32, 32, 1) --->no. of filters = 6 ---> filter(5, 5) --->strides = 1 --->activation(relu) --->output(28, 28, 6)***

***SubSampling 1 Averagepooling2D --->input(28, 28, 6) ---> filter(2, 2) --->strides = 2 --->output(14, 14, 6)***

***Convolution 2 --->input(14, 14, 6) --->no. of filters = 16 ---> filter(5, 5) --->strides = 1 --->activation(relu) --->output(10, 10, 16)***

***SubSampling 2 Averagepooling2D --->input(10, 10, 16) ---> filter(2, 2) --->strides = 2 --->output(5, 5, 16)***

***FullyConnected 1 --->input(5, 5, 16) --->activation(relu) --->output(120)***

***FullyConnected 2 --->input(120) --->activation(relu) --->output(84)***

***Output --->input(84) --->activation(softmax) --->output(10)***

In [ ]:

```
model = Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), padding='valid', activation='relu', input_shape=
input_shape))
model.add(AveragePooling2D(pool_size=(2, 2), strides = 2))
model.add(Conv2D(16, kernel_size=(5, 5), padding='valid', activation='relu'))
model.add(AveragePooling2D(pool_size=(2, 2), strides = 2))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

In [ ]:

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test,
y_test))
```

```
Epoch 1/20
1563/1563 [==============================] - 32s 20ms/step - loss: 1.9776 - acc: 0.2752 -
val_loss: 1.5926 - val_acc: 0.4180
```

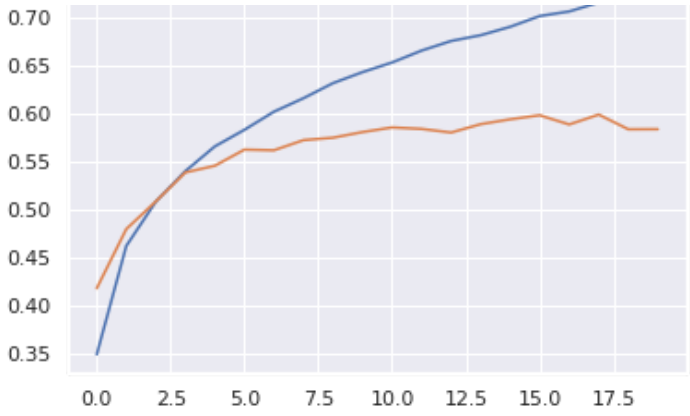
```
Epoch 2/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.5403 - acc: 0.4458 -
val_loss: 1.4377 - val_acc: 0.4797
Epoch 3/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.3977 - acc: 0.5026 -
val_loss: 1.3832 - val_acc: 0.5085
Epoch 4/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.3016 - acc: 0.5407 -
val_loss: 1.3103 - val_acc: 0.5387
Epoch 5/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.2289 - acc: 0.5652 -
val_loss: 1.3023 - val_acc: 0.5457
Epoch 6/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.1711 - acc: 0.5858 -
val_loss: 1.2634 - val_acc: 0.5625
Epoch 7/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.1269 - acc: 0.6033 -
val_loss: 1.2593 - val_acc: 0.5618
Epoch 8/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.0876 - acc: 0.6161 -
val_loss: 1.2390 - val_acc: 0.5724
Epoch 9/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.0402 - acc: 0.6373 -
val_loss: 1.2409 - val_acc: 0.5749
Epoch 10/20
1563/1563 [==============================] - 31s 20ms/step - loss: 1.0142 - acc: 0.6441 -
val_loss: 1.2160 - val_acc: 0.5808
Epoch 11/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.9802 - acc: 0.6552 -
val_loss: 1.2194 - val_acc: 0.5856
Epoch 12/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.9436 - acc: 0.6692 -
val_loss: 1.2296 - val_acc: 0.5841
Epoch 13/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.9158 - acc: 0.6801 -
val_loss: 1.2512 - val_acc: 0.5803
Epoch 14/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.8887 - acc: 0.6871 -
val_loss: 1.2124 - val_acc: 0.5890
Epoch 15/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.8611 - acc: 0.6962 -
val_loss: 1.2370 - val_acc: 0.5941
Epoch 16/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.8278 - acc: 0.7094 -
val_loss: 1.2104 - val_acc: 0.5983
Epoch 17/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.8099 - acc: 0.7120 -
val_loss: 1.2405 - val_acc: 0.5887
Epoch 18/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.7849 - acc: 0.7225 -
val_loss: 1.2414 - val_acc: 0.5990
Epoch 19/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.7624 - acc: 0.7320 -
val_loss: 1.2979 - val_acc: 0.5837
Epoch 20/20
1563/1563 [==============================] - 31s 20ms/step - loss: 0.7524 - acc: 0.7348 -
val_loss: 1.3099 - val_acc: 0.5838
```

In [ ]:

```
model.save('CustomCNNusingCIFAR_10.h5')
```

In [ ]:

```
plt.title('Training Accuracy')
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.show()
```




In [ ]:

```
predictions = model.predict(X_test)
predictions = one_hot_encoder.inverse_transform(predictions)
y_test = one_hot_encoder.inverse_transform(y_test)
cm = confusion_matrix(y_test, predictions)
plt.figure(figsize=(9,9))
sns.heatmap(cm, cbar=False, xticklabels=labels, yticklabels=labels, fmt='d', annot=True,
cmap=plt.cm.Blues)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```