NAME: VEENA T.G.S ROLL NO: 225229145

step-1

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import keras
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.backend import categorical_crossentropy
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
```

step-2

```
(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
170498071/170498071 [==============================] - 3s 0us/step
```

```
print('Shape of X_train is {}'.format(X_train.shape))
print('Shape of X_test is {}'.format(X_test.shape))
print('Shape of y_train is {}'.format(y_train.shape))
print('Shape of y_test is {}'.format(y_test.shape))
```

```
Shape of X_train is (50000, 32, 32, 3)
Shape of X_test is (10000, 32, 32, 3)
Shape of y_train is (50000, 1)
Shape of y_test is (10000, 1)
```

step-3

```
num_classes =10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

step-4

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

```
print('Shape of one sample of X_train is {}'.format(X_train[37].shape))
print('Shape of one sample of y_train is {}'.format(y_train[37].shape))
```

```
Shape of one sample of X_train is (32, 32, 3)
Shape of one sample of y_train is (10,)
```

```
X_train[37]
```

```
array([[[0.37254903, 0.4117647 , 0.49803922],
        [0.34509805, 0.38039216, 0.47058824],
        [0.3372549 , 0.3764706 , 0.4627451 ],
        ...,
        [0.39607844, 0.45490196, 0.5647059 ],
        [0.35686275, 0.42352942, 0.53333336],
        [0.4117647 , 0.4862745 , 0.6156863 ]],

       [[0.32156864, 0.3529412 , 0.43137255],
        [0.29411766, 0.3254902 , 0.40784314],
        [0.29803923, 0.32941177, 0.40784314],
        ...,
        [0.36862746, 0.4       , 0.48235294],
        [0.2       , 0.23921569, 0.3137255 ],
        [0.32941177, 0.38039216, 0.47843137]],

       [[0.3019608 , 0.33333334, 0.40392157],
        [0.2901961 , 0.31764707, 0.38431373],
        [0.2784314 , 0.30588236, 0.37254903],
        ...,
        [0.2784314 , 0.2901961 , 0.3372549 ],
        [0.18431373, 0.20392157, 0.24705882],
        [0.34509805, 0.37254903, 0.43529412]],

       ...,

       [[0.38039216, 0.37254903, 0.28235295],
        [0.36078432, 0.36078432, 0.27058825],
        [0.38039216, 0.3647059 , 0.27450982],
        ...,
        [0.3372549 , 0.35686275, 0.25490198],
        [0.36862746, 0.38039216, 0.28235295],
        [0.3529412 , 0.38039216, 0.2784314 ]],

       [[0.37254903, 0.3529412 , 0.25490198],
        [0.32941177, 0.3372549 , 0.23137255],
        [0.34901962, 0.34901962, 0.24313726],
        ...,
        [0.3764706 , 0.38039216, 0.29803923],
        [0.4       , 0.3764706 , 0.3019608 ],
        [0.38039216, 0.36862746, 0.28627452]],

       [[0.35686275, 0.32941177, 0.24705882],
        [0.3254902 , 0.31764707, 0.22352941],
        [0.32156864, 0.31764707, 0.21568628],
        ...,
        [0.39215687, 0.3764706 , 0.30588236],
        [0.4117647 , 0.38039216, 0.3137255 ],
        [0.42352942, 0.4       , 0.3254902 ]]]], dtype=float32)
```

```
y_train[37]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
model = Sequential()
model.add(Conv2D(32, (5,5), strides=(2,2), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (5,5), strides=(2,2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 32)        2432

 activation (Activation)     (None, 16, 16, 32)        0

 conv2d_1 (Conv2D)           (None, 6, 6, 32)          25632

 activation_1 (Activation)   (None, 6, 6, 32)          0

 max_pooling2d (MaxPooling2  (None, 3, 3, 32)          0
 D)

 dropout (Dropout)           (None, 3, 3, 32)          0

 flatten (Flatten)           (None, 288)               0

 dense (Dense)               (None, 512)               147968

 activation_2 (Activation)   (None, 512)               0

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

 activation_3 (Activation)   (None, 10)                0

=================================================================
Total params: 181162 (707.66 KB)
Trainable params: 181162 (707.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

step-5

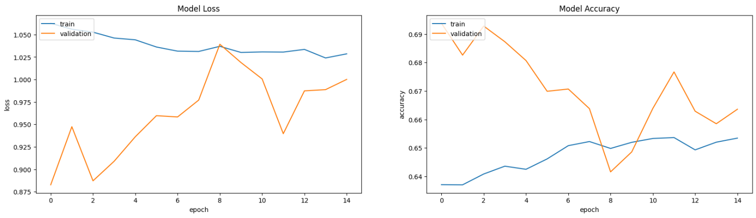
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
batch_size=32
opt = tf.keras.optimizers.RMSprop(learning_rate=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, batch_size=32, epochs=15, verbose=2, validation_data=(X_val, y_val))
```

```
    Epoch 1/15
    1250/1250 - 30s - loss: 1.0626 - accuracy: 0.6371 - val_loss: 0.8827 - val_accuracy: 0.6938 - 30s/epoch - 24ms/step
    Epoch 2/15
    1250/1250 - 27s - loss: 1.0560 - accuracy: 0.6370 - val_loss: 0.9474 - val_accuracy: 0.6826 - 27s/epoch - 21ms/step
    Epoch 3/15
    1250/1250 - 25s - loss: 1.0527 - accuracy: 0.6409 - val_loss: 0.8871 - val_accuracy: 0.6928 - 25s/epoch - 20ms/step
    Epoch 4/15
    1250/1250 - 32s - loss: 1.0461 - accuracy: 0.6436 - val_loss: 0.9089 - val_accuracy: 0.6873 - 32s/epoch - 25ms/step
    Epoch 5/15
    1250/1250 - 27s - loss: 1.0442 - accuracy: 0.6425 - val_loss: 0.9362 - val_accuracy: 0.6807 - 27s/epoch - 22ms/step
    Epoch 6/15
    1250/1250 - 25s - loss: 1.0362 - accuracy: 0.6461 - val_loss: 0.9597 - val_accuracy: 0.6699 - 25s/epoch - 20ms/step
    Epoch 7/15
    1250/1250 - 26s - loss: 1.0316 - accuracy: 0.6508 - val_loss: 0.9582 - val_accuracy: 0.6707 - 26s/epoch - 20ms/step
    Epoch 8/15
    1250/1250 - 27s - loss: 1.0312 - accuracy: 0.6522 - val_loss: 0.9770 - val_accuracy: 0.6638 - 27s/epoch - 22ms/step
    Epoch 9/15
    1250/1250 - 26s - loss: 1.0369 - accuracy: 0.6498 - val_loss: 1.0392 - val_accuracy: 0.6416 - 26s/epoch - 21ms/step
    Epoch 10/15
    1250/1250 - 25s - loss: 1.0301 - accuracy: 0.6520 - val_loss: 1.0188 - val_accuracy: 0.6486 - 25s/epoch - 20ms/step
    Epoch 11/15
    1250/1250 - 26s - loss: 1.0307 - accuracy: 0.6533 - val_loss: 1.0006 - val_accuracy: 0.6639 - 26s/epoch - 21ms/step
    Epoch 12/15
    1250/1250 - 25s - loss: 1.0305 - accuracy: 0.6536 - val_loss: 0.9396 - val_accuracy: 0.6767 - 25s/epoch - 20ms/step
    Epoch 13/15
    1250/1250 - 25s - loss: 1.0335 - accuracy: 0.6493 - val_loss: 0.9873 - val_accuracy: 0.6629 - 25s/epoch - 20ms/step
    Epoch 14/15
    1250/1250 - 26s - loss: 1.0239 - accuracy: 0.6520 - val_loss: 0.9887 - val_accuracy: 0.6585 - 26s/epoch - 21ms/step
    Epoch 15/15
    1250/1250 - 25s - loss: 1.0286 - accuracy: 0.6535 - val_loss: 1.0001 - val_accuracy: 0.6636 - 25s/epoch - 20ms/step
```

```
fig = plt.figure(figsize=(20, 5))
fig.add_subplot(1,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
fig.add_subplot(1,2,2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



step-6

```
model1 = Sequential()
model1.add(Conv2D(32, (5,5), strides=(2,2), padding='same', input_shape=X_train.shape[1:]))
model1.add(Activation('relu'))
model1.add(Conv2D(32, (5,5), strides=(2,2)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Dropout(0.25))
model1.add(Flatten())
model1.add(Dense(512))
model1.add(Activation('relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes))
model1.add(Activation('softmax'))
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 16, 16, 32)        2432

 activation_4 (Activation)   (None, 16, 16, 32)        0

 conv2d_3 (Conv2D)           (None, 6, 6, 32)          25632

 activation_5 (Activation)   (None, 6, 6, 32)          0

 max_pooling2d_1 (MaxPoolin  (None, 3, 3, 32)          0
 g2D)

 dropout_2 (Dropout)         (None, 3, 3, 32)          0

 flatten_1 (Flatten)         (None, 288)               0

 dense_2 (Dense)             (None, 512)               147968

 activation_6 (Activation)   (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

 activation_7 (Activation)   (None, 10)                0

=================================================================
Total params: 181162 (707.66 KB)
Trainable params: 181162 (707.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

step-7

```
model1 = Sequential()
model1.add(Conv2D(32, (5,5), strides=(2,2), padding='same', input_shape=X_train.shape[1:]))
model1.add(Activation('relu'))
model1.add(Conv2D(32, (5,5), strides=(2,2)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Dropout(0.25))
model1.add(Flatten())
model1.add(Dense(512))
model1.add(Activation('relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes))
model1.add(Activation('softmax'))
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 16, 16, 32)        2432

 activation_8 (Activation)   (None, 16, 16, 32)        0

 conv2d_5 (Conv2D)           (None, 6, 6, 32)          25632

 activation_9 (Activation)   (None, 6, 6, 32)          0

 max_pooling2d_2 (MaxPoolin  (None, 3, 3, 32)          0
 g2D)
 dropout_4 (Dropout)         (None, 3, 3, 32)          0         
                                                                 
 flatten_2 (Flatten)         (None, 288)               0         
                                                                 
 dense_4 (Dense)             (None, 512)               147968    
                                                                 
 activation_10 (Activation)  (None, 512)               0         
                                                                 
 dropout_5 (Dropout)         (None, 512)               0         
                                                                 
 dense_5 (Dense)             (None, 10)                5130      
                                                                 
 activation_11 (Activation)  (None, 10)                0         
                                                                 
=================================================================
Total params: 181162 (707.66 KB)
Trainable params: 181162 (707.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Double-click (or enter) to edit

```
datagen = ImageDataGenerator(featurewise_center=False,
samplewise_center=False,

featurewise_std_normalization=False,
samplewise_std_normalization=False,
zca_whitening=False,
rotation_range=0,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
vertical_flip=False)

datagen.fit(X_train)
```

```
batch_size=32
opt = RMSprop(learning_rate=0.0005)
model1.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```