NAME : C . ASHIKA

ROLL NO : 225229105

LAB - 9 IMAGE CLASSIFICATION USING CNN FOR CIFAR-10 DATA

STEPS

PART - I BASELINE MODEL

1 . IMPORT LIBRARIES

In [1]:

```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D,MaxPooling2D
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator
```

2 . LOAD YOUR DATA AND PRINT THE SHAPE OF TRAINING AND TEST SAMPLES

In [3]:

```

(x_train,y_train),(x_test,y_test)=cifar10.load_data()
print('x_train shape:',x_train.shape)
print(x_train.shape[0],'train samples')
print(x_test.shape[0],'test samples')
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

3 . PRINT THE SHAPE OF ONE IMAGE

In [4]:

```
x_train[5000].shape
```

Out[4]:

```
(32, 32, 3)
```

4 . DISPLAY ONE IMAGE USING imshow() Function

In [5]:

```
print(y_train[444])
```

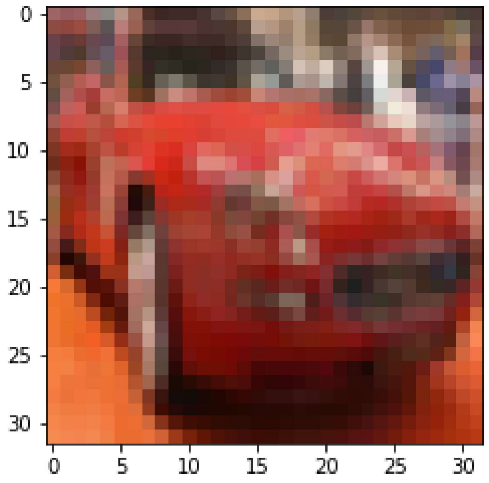
```
[9]
```

In [21]:

```
plt.imshow(x_train[5])
```

Out[21]:

```
<matplotlib.image.AxesImage at 0x2680269c940>
```



5 . CONVERT Y_train AND Y_test INTO CATEGORICAL VALUES

In [7]:

```
from tensorflow.keras.utils import to_categorical
```

In [8]:

```
num_classes=10
y_train=keras.utils.to_categorical (y_train,num_classes)
y_test=keras.utils.to_categorical (y_test,num_classes)
```

In [9]:

```
y_train[4]
```

Out[9]:

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

6 . CONVERT TRAIN DATA INTO FLOAT AND SCALE

In [10]:

```
x_train=x_train.astype('float32')

x_test=x_test.astype('float32')
x_train /=255
x_test /=255
```

7 . BUILD YOUR FIRST CNN

In [11]:

```
# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Define the CNN architecture
model = Sequential()

model.add(Conv2D(32, (5, 5), strides=(2, 2), activation='relu', padding='same', input_sh
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), strides=(2, 2), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

In [12]:

```
import tensorflow as tf
print(help(tf.keras.optimizers.Adam))
```

```
 |   gradients, and is well suited for problems that are large in terms
of
 |   data/parameters*".
 |
 |   Args:
 |     learning_rate: A `tf.Tensor`, floating point value, a schedule t
hat is a
 |       `tf.keras.optimizers.schedules.LearningRateSchedule`, or a cal
lable
 |       that takes no arguments and returns the actual value to use. T
he
 |       learning rate. Defaults to `0.001`.
 |     beta_1: A float value or a constant float tensor, or a callable
 |       that takes no arguments and returns the actual value to use. T
he
 |       exponential decay rate for the 1st moment estimates. Defaults
to `0.9`.
 |     beta_2: A float value or a constant float tensor, or a callable
 |       that takes no arguments and returns the actual value to use. T
he
```

In [16]:

```
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import legacy
```

8 . PRINT THE SUMMARY AND VERIFY YOUR CONFIGURATION

9 . COMPILE AND FIT AND VALIDATE YOUR MODEL WITH THE FOLLOWING PARAMETERS

In [18]:

```
# Compile the model with RMSprop optimizer and categorical_crossentropy loss
from keras.optimizers import RMSprop
optimizer =tf.keras.optimizers.legacy.RMSprop(learning_rate=0.0005, decay=1e-6)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy']

# Display the model summary
model.summary()

# Train the model
batch_size = 32
epochs = 15
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1,

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy}")
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 32)        2432      
                                                                 
 max_pooling2d (MaxPooling2  (None, 8, 8, 32)          0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 4, 4, 64)          18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 2, 2, 64)          0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 256)               0         
                                                                 
 dropout (Dropout)           (None, 256)               0         
                                                                 
 dense (Dense)               (None, 128)               32896     
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 512)               66048     
                                                                 
 dense_2 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 125002 (488.29 KB)
Trainable params: 125002 (488.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/15
1407/1407 [==============================] - 12s 8ms/step - loss: 1.8652
- accuracy: 0.3056 - val_loss: 1.5959 - val_accuracy: 0.4062
Epoch 2/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.5943
- accuracy: 0.4138 - val_loss: 1.5952 - val_accuracy: 0.4182
Epoch 3/15
1407/1407 [==============================] - 10s 7ms/step - loss: 1.4871
- accuracy: 0.4558 - val_loss: 1.4518 - val_accuracy: 0.4642
Epoch 4/15
1407/1407 [==============================] - 12s 8ms/step - loss: 1.4255
- accuracy: 0.4788 - val_loss: 1.3353 - val_accuracy: 0.5058
Epoch 5/15
1407/1407 [==============================] - 13s 9ms/step - loss: 1.3718
- accuracy: 0.5015 - val_loss: 1.2577 - val_accuracy: 0.5438
Epoch 6/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.3298
- accuracy: 0.5237 - val_loss: 1.2230 - val_accuracy: 0.5598
Epoch 7/15
1407/1407 [==============================] - 12s 9ms/step - loss: 1.3022
- accuracy: 0.5322 - val_loss: 1.1855 - val_accuracy: 0.5814
Epoch 8/15
1407/1407 [==============================] - 13s 9ms/step - loss: 1.2767
- accuracy: 0.5438 - val_loss: 1.2642 - val_accuracy: 0.5476
Epoch 9/15
1407/1407 [==============================] - 12s 9ms/step - loss: 1.2564
- accuracy: 0.5547 - val_loss: 1.3083 - val_accuracy: 0.5400
Epoch 10/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.2330
- accuracy: 0.5646 - val_loss: 1.1098 - val_accuracy: 0.6108
```

```
Epoch 11/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.2183
- accuracy: 0.5715 - val_loss: 1.2023 - val_accuracy: 0.5766
Epoch 12/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.2080
- accuracy: 0.5740 - val_loss: 1.1322 - val_accuracy: 0.6080
Epoch 13/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.1917
- accuracy: 0.5807 - val_loss: 1.1663 - val_accuracy: 0.5800
Epoch 14/15
1407/1407 [==============================] - 13s 9ms/step - loss: 1.1849
- accuracy: 0.5844 - val_loss: 1.0440 - val_accuracy: 0.6312
Epoch 15/15
1407/1407 [==============================] - 11s 8ms/step - loss: 1.1738
- accuracy: 0.5903 - val_loss: 1.0610 - val_accuracy: 0.6220
313/313 [==============================] - 1s 4ms/step - loss: 1.0903 - a
ccuracy: 0.6201
Test accuracy: 0.6201000213623047
```

PART - II MODEL IMPROVEMENTS

In [20]:

```
# Define the more complicated CNN architecture
model = Sequential()

model.add(Conv2D(32, (3, 3), strides=(1, 1), activation='relu', padding='same', input_sh
model.add(Conv2D(64, (3, 3), strides=(1, 1), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), strides=(1, 1), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), strides=(1, 1), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
optimizer = tf.keras.optimizers.legacy.RMSprop(learning_rate=0.0005, decay=1e-6)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy']

# Display the model summary and parameter count
model.summary()
print("Total Parameters:", model.count_params())

# Train the model for 5 epochs
batch_size = 64
epochs = 5
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_s

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy}")
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896       
                                                                 
 conv2d_3 (Conv2D)           (None, 32, 32, 64)        18496     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_4 (Conv2D)           (None, 16, 16, 128)       73856     
                                                                 
 conv2d_5 (Conv2D)           (None, 16, 16, 256)       295168    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 256)         0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 16384)             0         
                                                                 
 dense_3 (Dense)             (None, 512)               8389120   
                                                                 
 dense_4 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 8782666 (33.50 MB)
Trainable params: 8782666 (33.50 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Total Parameters: 8782666
Epoch 1/5
704/704 [==============================] - 347s 491ms/step - loss: 1.4756
- accuracy: 0.4732 - val_loss: 1.2042 - val_accuracy: 0.5512
Epoch 2/5
704/704 [==============================] - 349s 496ms/step - loss: 0.9249
- accuracy: 0.6767 - val_loss: 0.9552 - val_accuracy: 0.6662
Epoch 3/5
704/704 [==============================] - 373s 530ms/step - loss: 0.6821
- accuracy: 0.7643 - val_loss: 1.1602 - val_accuracy: 0.6126
Epoch 4/5
704/704 [==============================] - 369s 524ms/step - loss: 0.5010
- accuracy: 0.8267 - val_loss: 1.0734 - val_accuracy: 0.6558
Epoch 5/5
704/704 [==============================] - 367s 521ms/step - loss: 0.3460
- accuracy: 0.8810 - val_loss: 0.7470 - val_accuracy: 0.7682
313/313 [==============================] - 21s 68ms/step - loss: 0.8069 -
accuracy: 0.7452
Test accuracy: 0.745199978351593
```