Annapoorima S

225229101

PDL Lab 9_Image Classification using CNN for CIFAR-10 Data

Part-I: Baseline Model

Import Libraries

In [44]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [47]:

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import Adam
from keras.backend import categorical_crossentropy
```

In [48]:

```
import tensorflow as tf
```

Load the data

In [49]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [50]:

```
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
```

Print the shape of one image

In [51]:

```
X_train[444].shape
```

Out[51]:

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

Display one image

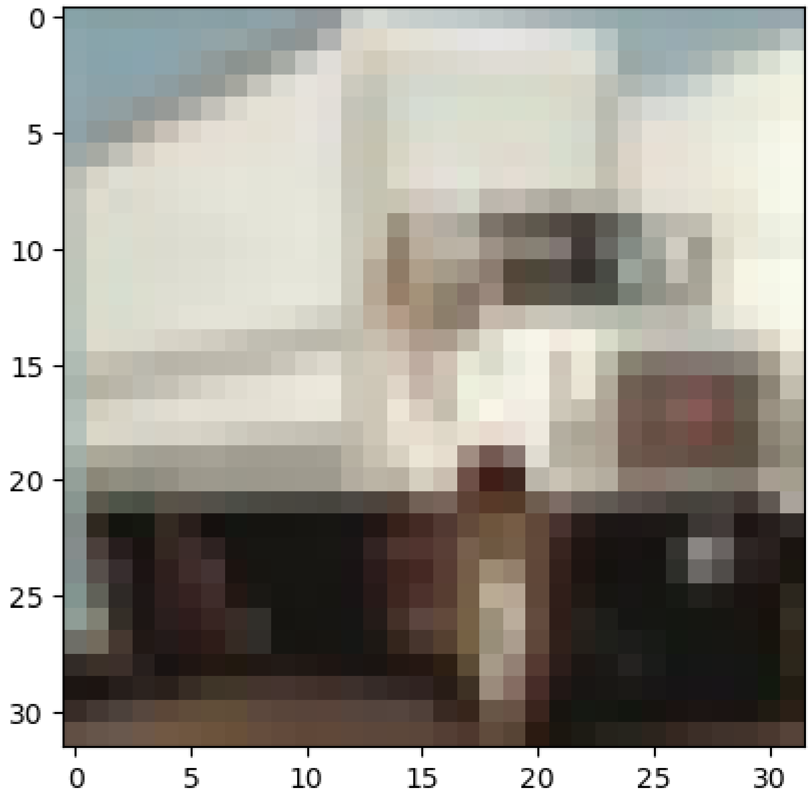
In [52]:

```
print(y_train[444])
plt.imshow(X_train[444])
```

```
[9]
```

Out[52]:

```
<matplotlib.image.AxesImage at 0x27027af7bd0>
```



Convert y_train and y_test into categorical values

In [53]:

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

In [54]:

```
y_train[444]
```

Out[54]:

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

Convert train data into float and sclae

In [55]:

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

Build CNN

In [56]:

```
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 [57]:

```
from tensorflow.keras.optimizers import legacy as legacy_optimizers
```

In [58]:

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

In [59]:

```
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy']
```

Print summary and verify configuration

In [60]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 16, 16, 32)        2432      
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 8, 8, 32)          0         
 g2D)                                                            
                                                                 
 conv2d_7 (Conv2D)           (None, 4, 4, 64)          18496     
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 2, 2, 64)          0         
 g2D)                                                            
                                                                 
 flatten_6 (Flatten)         (None, 256)               0         
                                                                 
 dropout_12 (Dropout)        (None, 256)               0         
                                                                 
 dense_12 (Dense)            (None, 128)               32896     
                                                                 
 dropout_13 (Dropout)        (None, 128)               0         
                                                                 
 dense_13 (Dense)            (None, 512)               66048     
                                                                 
 dense_14 (Dense)            (None, 10)                5130      
                                                                 
=================================================================
Total params: 125002 (488.29 KB)
Trainable params: 125002 (488.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [63]:

```
batch_size = 32
epochs = 15
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1,
```

```
Epoch 1/15
1407/1407 [==============================] - 34s 23ms/step - loss: 1.8767
- accuracy: 0.3032 - val_loss: 1.6192 - val_accuracy: 0.4028
Epoch 2/15
1407/1407 [==============================] - 28s 20ms/step - loss: 1.6027
- accuracy: 0.4115 - val_loss: 1.5894 - val_accuracy: 0.4108
Epoch 3/15
1407/1407 [==============================] - 30s 21ms/step - loss: 1.4953
- accuracy: 0.4503 - val_loss: 1.3957 - val_accuracy: 0.4858
Epoch 4/15
1407/1407 [==============================] - 31s 22ms/step - loss: 1.4263
- accuracy: 0.4800 - val_loss: 1.4371 - val_accuracy: 0.4844
Epoch 5/15
1407/1407 [==============================] - 32s 23ms/step - loss: 1.3786
- accuracy: 0.5022 - val_loss: 1.2753 - val_accuracy: 0.5308
Epoch 6/15
1407/1407 [==============================] - 27s 19ms/step - loss: 1.3384
- accuracy: 0.5186 - val_loss: 1.2360 - val_accuracy: 0.5418
Epoch 7/15
1407/1407 [==============================] - 26s 18ms/step - loss: 1.3084
- accuracy: 0.5297 - val_loss: 1.1904 - val_accuracy: 0.5790
Epoch 8/15
1407/1407 [==============================] - 24s 17ms/step - loss: 1.2763
- accuracy: 0.5435 - val_loss: 1.2051 - val_accuracy: 0.5664
Epoch 9/15
1407/1407 [==============================] - 26s 19ms/step - loss: 1.2576
- accuracy: 0.5488 - val_loss: 1.1819 - val_accuracy: 0.5782
Epoch 10/15
1407/1407 [==============================] - 29s 21ms/step - loss: 1.2396
- accuracy: 0.5578 - val_loss: 1.2156 - val_accuracy: 0.5714
Epoch 11/15
1407/1407 [==============================] - 40s 28ms/step - loss: 1.2189
- accuracy: 0.5677 - val_loss: 1.1412 - val_accuracy: 0.5898
Epoch 12/15
1407/1407 [==============================] - 31s 22ms/step - loss: 1.2087
- accuracy: 0.5748 - val_loss: 1.2205 - val_accuracy: 0.5708
Epoch 13/15
1407/1407 [==============================] - 32s 22ms/step - loss: 1.1997
- accuracy: 0.5782 - val_loss: 1.1096 - val_accuracy: 0.6108
Epoch 14/15
1407/1407 [==============================] - 34s 24ms/step - loss: 1.1891
- accuracy: 0.5839 - val_loss: 1.0962 - val_accuracy: 0.6158
Epoch 15/15
1407/1407 [==============================] - 30s 21ms/step - loss: 1.1813
- accuracy: 0.5858 - val_loss: 1.1127 - val_accuracy: 0.6078
```

Out[63]:

```
<keras.src.callbacks.History at 0x27027b6c450>
```

In [64]:

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy}")
```

```
313/313 [==============================] - 2s 8ms/step - loss: 1.1242 - a
ccuracy: 0.6021
Test accuracy: 0.6021000146865845
```

Compile and fit and validate

Part-II: Model Improvements

In [67]:

```
model1 = Sequential()
model1.add(Conv2D(filters=32, kernel_size=(5,5), strides=1, padding='same', activation='
model1.add(Conv2D(filters=32, kernel_size=(5,5), strides=1, padding='same', activation='
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Conv2D(filters=64, kernel_size=(5,5), strides=1, padding='same', activation='
model1.add(Conv2D(filters=64, kernel_size=(5,5), strides=1, padding='same', activation='
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Dropout(0.25))
model1.add(Flatten())
model1.add(Dense(512, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(10, activation='softmax'))
model1.compile(optimizer='adam', loss=loss, metrics=met)
history1 = model1.fit(X_train, y_train, shuffle=True, epochs=5, batch_size=32, validatic
score1 = model1.evaluate(X_test, y_test, verbose=0)
```

```
Epoch 1/5
1250/1250 [==============================] - 799s 635ms/step - loss: 1.62
64 - accuracy: 0.3997 - val_loss: 1.2617 - val_accuracy: 0.5451
Epoch 2/5
1250/1250 [==============================] - 860s 688ms/step - loss: 1.25
30 - accuracy: 0.5505 - val_loss: 1.0829 - val_accuracy: 0.6216
Epoch 3/5
1250/1250 [==============================] - 696s 557ms/step - loss: 1.07
39 - accuracy: 0.6211 - val_loss: 0.9444 - val_accuracy: 0.6634
Epoch 4/5
1250/1250 [==============================] - 730s 584ms/step - loss: 0.94
89 - accuracy: 0.6658 - val_loss: 0.9217 - val_accuracy: 0.6710
Epoch 5/5
1250/1250 [==============================] - 724s 579ms/step - loss: 0.85
85 - accuracy: 0.6976 - val_loss: 0.8650 - val_accuracy: 0.6949
```

In [68]:

```
print('Test loss:', score1[0])
print('Test accuracy:', score1[1]*100)
```

```
Test loss: 0.8830985426902771
Test accuracy: 69.22000050544739
```