```
# Case Study - Image Classification using Deep CNN in Keras.
```

In [ ]:
```
!pip install opencv-python
#OpenCV is a Python library that allows you to perform image processing and
#It provides a wide range of features, including object detection, face reco
```

In [1]:
```
# Import necessary modules.

import cv2
import numpy as np
import matplotlib.pyplot as plt


from tensorflow.keras import datasets, models, layers, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
WARNING:tensorflow:From C:\Users\SIMRAN\anaconda3\lib\site-packages\keras
\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is de
precated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy inst
ead.
```

In [2]:
```
# Set the batch size, number of epochs.
batch_size = 32
num_classes = 10
epochs = 40
num_predictions = 20
#this can be set at the later stage after trying different iterartions, this
```

In [3]:
```
#lets import data sets from kears
from tensorflow.keras.datasets import cifar10
```

In [4]:
```

#if in case data set doesnt run you can use this:
import requests
import ssl

# Bypass SSL verification
ssl._create_default_https_context = ssl._create_unverified_context

# Download CIFAR-10 dataset
url = "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
response = requests.get(url)
```

In [5]:
```
# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [6]:
```
# Print the shape of dataset.
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
```

In [7]:
```
# Print the shape of dataset.
print('y_train shape:', y_train.shape)
print(y_train.shape[0], 'train samples')
print(y_test.shape[0], 'test samples')
```

```
y_train shape: (50000, 1)
50000 train samples
10000 test samples
```

- The training set contains 50000 images.
- The size of each image is 32x32 pixels.
- Each image has 3 color channels.

In [8]:
```
x_train[0, :, :, :].shape  #checking shape of the x_train by index
```

Out[8]:
```
(32, 32, 3)
```

Highlights:

- How to select the 10th image?
- How to get the red pixels only?
- What is the shape of resulting array?

In [9]:
```
y_train #the output of the training sets
```

Out[9]:
```
array([[6],
       [9],
       [9],
       ...,
       [9],
       [1],
       [1]], dtype=uint8)
```

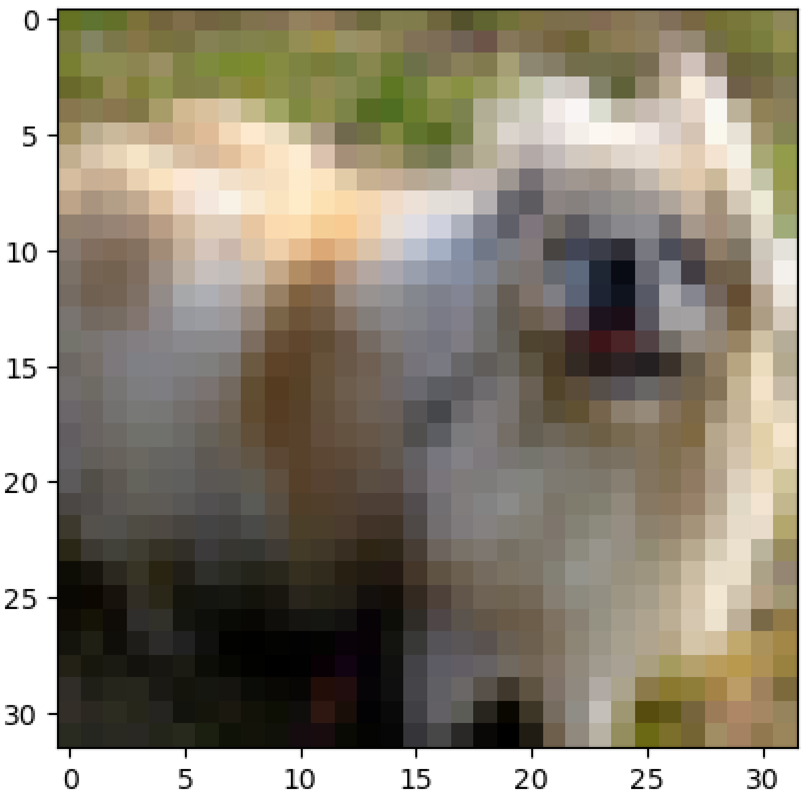
In [10]:
```
y_train[2][0] #output of the the 2nd index in the training set
```

Out[10]:
```
9
```

In [11]:
```
#lets label the index for better understanding
label_dict =  {0:'airplane', 1:'automobile', 2:'bird', 3:'cat', 4:'deer', 5
```

In [12]:
```
#lets check the image and the label for the 40th data from x_train and y_tr
i = 40
image = x_train[i]
label = y_train[i][0]
print(f'Label \n Label Id: {label} \n Name: {label_dict[label]}')
plt.imshow(image);
#pixel is less, hence the quality of image seems bad,
#as this code is just for learning purspose, no problem with the the traini
```

```
Label
 Label Id: 5
 Name: dog
```



- As the image quality is not good, the edges are not so good. But still we can visualize that there are edges.

In [13]:
```
y_test
```

Out[13]:
```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

Please Note:

There are many tools to one-hot encode and they differ in syntax, but the keras one is probably best implemented.

- `keras.utils.to_categorical`

- sklearn.preprocessing.OneHotEncoder
- pandas get_dummies

In [14]:
```
y_train.shape #lets reshape it using onehot encoder, below shown are two way
```

Out[14]:
```
(50000, 1)
```

In [15]:
```
# Convert labels to one hot vectors.

from sklearn.preprocessing import LabelBinarizer
enc = LabelBinarizer()
y_train = enc.fit_transform(y_train)
y_test = enc.fit_transform(y_test)
```

In [16]:
```
print(y_train.shape)
print(y_test.shape)
```

```
(50000, 10)
(10000, 10)
```

In [21]:
```
#y_train, it can be seen now as, 1 where it is positive with the image
y_train[0]
```

Out[21]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
```

In [ ]:
```
#second method for the same
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

pd.get_dummies(y)
```

```
Create the Model:

- Convolutional input layer, 32 feature maps with a size of 5×5 and a
rectifier activation function.
- Batch Normalization Layer.
- Convolutional layer, 32 feature maps with a size of 5×5 and a rectifier
activation function.
- Batch Normalization layer.
- Max Pool layer with size 2×2.
- Dropout layer at 25%.
---
- Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier
activation function.
- Batch Normalization layer.
- Dropout layer at 25%.
- Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier
activation function.
- Batch Normalization layer.
- Max Pool layer with size 2×2.
- Dropout layer at 25%.
---
- GlobalMaxPooling2D layer.
- Fully connected layer with 256 units and a rectifier activation
function.
- Dropout layer at 50%.
- Fully connected output layer with 10 units and a softmax activation
function.
```

In [18]:

```
# Set the CNN model

batch_size = None

model = models.Sequential()
model.add(layers.Conv2D(32, (5, 5), padding='same', activation="relu", inpu
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (5, 5), padding='same', activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(64, (3, 3), padding='same', activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

model.add(layers.Conv2D(64, (3, 3), padding='same', activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.5))

model.add(layers.Flatten())
model.add(layers.Dense(256, activation="relu")) #one hidden layer with 256 
model.add(layers.Dropout(0.5))

# softmax
model.add(layers.Dense(10, activation="softmax"))

model.summary()
```

```
WARNING:tensorflow:From C:\Users\SIMRAN\anaconda3\lib\site-packages\keras
\src\backend.py:873: The name tf.get_default_graph is deprecated. Please u
se tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\SIMRAN\anaconda3\lib\site-packages\keras
\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused
_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm ins
tead.

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

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        51264

 batch_normalization_1 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          36928

 batch_normalization_2 (Bat  (None, 8, 8, 64)          256
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 64)          0

 conv2d_3 (Conv2D)           (None, 4, 4, 64)          36928

 batch_normalization_3 (Bat  (None, 4, 4, 64)          256
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 2, 2, 64)          0
 g2D)

 dropout_3 (Dropout)         (None, 2, 2, 64)          0

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 256)               65792

 dropout_4 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570
```

```
=================================================================
Total params: 196810 (768.79 KB)
Trainable params: 196362 (767.04 KB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

When training the network, what you want is minimize the cost by applying a algorithm of your choice. It could be SGD, AdamOptimizer, AdagradOptimizer, or something. You have to study how each algorithm works to choose what to use, but AdamOptimizer works find for most cases in general.

In [19]:
```
# initiate Adam optimizer
opt = optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilo
```

In [20]:
```
# Let's train the model
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

Standarizing the data

- The pixel values are in the range of 0 to 255 for each of the red, green and blue channels.

- It is good practice to work with normalized data. Because the input values are well understood, we can easily normalize to the range 0 to 1 by dividing each value by the maximum observation which is 255.

- Note, the data is loaded as integers, so we must cast it to floating point values in order to perform the division.

In [21]:
```
#standarizing the data
x_train = x_train.astype('float32') # Conversion to float type from integer
x_test = x_test.astype('float32')
x_train /= 255.0 # Division by 255
x_test /= 255.0
```

In [22]:
```
#Adding Early stopping callback to the fit function is going to stop the tra
#if the val_loss is not going to change even '0.001' for more than 10 conti

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.001, patienc

#Adding Model Checkpoint callback to the fit function is going to save the w
#Hence saving the best weights occurred during training

model_checkpoint =  ModelCheckpoint('cifar_cnn_checkpoint_{epoch:02d}_loss{v
                                    monitor='val_los
                                    verbose=1,
                                    save_best_only=Tr
                                    save_weights_onl
                                    mode='auto',
                                    period=1)
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batches seen.
```

In [23]:
```
x_train.shape[0]
```
Out[23]:
```
50000
```

In [24]:
```
x_train = x_train.reshape(x_train.shape[0], 32, 32, 3)
x_test = x_test.reshape(x_test.shape[0], 32, 32, 3)

print(x_train.shape)
print(x_test.shape)
```
```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

In [25]:
```
history = model.fit(x_train,
                    y_train,
                    batch_size=batch_size,
                    epochs=100,
                    validation_data=(x_test, y_test),
                    shuffle=True,
                    verbose=1,
                    callbacks=[early_stopping,model_checkpoint])

# plot training history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

#for learning purpose less epochs is choosen, you should tune this hyperpara
```
```
Epoch 30: val_loss did not improve from 0.66675
1563/1563 [==============================] - 104s 66ms/step - loss: 0.73
17 - accuracy: 0.7463 - val_loss: 0.8866 - val_accuracy: 0.7029
Epoch 31/100
1563/1563 [==============================] - ETA: 0s - loss: 0.7213 - ac
curacy: 0.7529
Epoch 31: val_loss did not improve from 0.66675
1563/1563 [==============================] - 105s 67ms/step - loss: 0.72
13 - accuracy: 0.7529 - val_loss: 0.6793 - val_accuracy: 0.7679
Epoch 32/100
1563/1563 [==============================] - ETA: 0s - loss: 0.7234 - ac
curacy: 0.7518
Epoch 32: val_loss improved from 0.66675 to 0.62610, saving model to cif
ar_cnn_checkpoint_32_loss0.6261.h5
1563/1563 [==============================] - 105s 67ms/step - loss: 0.72
34 - accuracy: 0.7518 - val_loss: 0.6261 - val_accuracy: 0.7874
Epoch 33/100
1563/1563 [==============================] - ETA: 0s - loss: 0.7126 - ac
curacy: 0.7557
Epoch 33: val_loss did not improve from 0.62610
```

In [26]:
```python
# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
# sigmoid
```

```
313/313 [==============================] - 7s 22ms/step - loss: 0.6071 - a
ccuracy: 0.7912
Test loss: 0.6071414351463318
Test accuracy: 0.7911999821662903
```

```
#hyperparameter tuning can increase the accuracy also the pixel of the
image is less which can be the reason for the
# low accuracy for now, but the score is good for the prediction
```

In [27]:
```python
predictions = model.predict(x_test)
```

```
313/313 [==============================] - 9s 22ms/step
```

In [28]:
```python
preds = pd.DataFrame(predictions)
preds
```

Out[28]:

	0	1	2	3	4	5	6	7	
0	1.085919e-04	1.776379e-04	0.000682	0.582963	0.000526	0.314984	0.099586	1.553694e-04	5
1	6.605163e-03	2.580067e-01	0.000073	0.000268	0.000003	0.000003	0.000027	2.000701e-06	7
2	1.644520e-02	1.034624e-02	0.000529	0.003659	0.000160	0.000023	0.000211	9.543000e-05	9
3	9.325504e-01	4.512614e-04	0.011549	0.002303	0.002365	0.000132	0.000329	9.004054e-05	4
4	7.867246e-06	1.356194e-05	0.004512	0.001868	0.001040	0.000052	0.992479	9.727671e-07	2
...	...	...	...	...	...	...	...	...	
9995	4.390072e-01	9.073644e-04	0.022465	0.112056	0.018362	0.016399	0.006648	2.214676e-03	3
9996	2.422249e-06	2.352338e-07	0.000811	0.529121	0.001928	0.411737	0.056266	1.334896e-04	2
9997	3.970541e-08	8.770414e-09	0.000059	0.004642	0.000003	0.995146	0.000013	1.380672e-04	6
9998	5.757232e-02	8.837342e-01	0.006984	0.009757	0.011011	0.003464	0.016252	4.518481e-04	5
9999	1.212057e-06	1.047841e-08	0.000035	0.000012	0.005108	0.000182	0.000001	9.946608e-01	4

10000 rows × 10 columns

In [ ]: