Manik Daya
FCI
2017140008
BE_IT

Experiment 2: Convolutional Neural Network

1 OBJECTIVE

Introduction to various Convolutional Neural Network Architectures and performing image classification through CNN

2 CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

4 SUBMISSION

Dataset: The dataset used here is CIFAR dataset which consists 60000 images and each image is 32*32 and 3 is the RGB channel.
In test we have 10000 images and in train we have 50000 images and x_train and x_test consists of images and y_train and y_test will have all the labels. The dataset is classified into 10 categories labelled as following:

airplane
automobile
bird
cat

deer
dog
frog
horse
ship
truck

Screenshots (code and implementation)

In [1]:
```
import tensorflow as tf
```

In [2]:
```
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout, GlobalMaxPooling2D
from tensorflow.keras.models import Model
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
cifar10 = tf.keras.datasets.cifar10
```

In [5]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = y_train.flatten(), y_test.flatten()
print("x_train.shape:", x_train.shape)
print("y_train.shape", y_train.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 281s 2us/step
x_train.shape: (50000, 32, 32, 3)
y_train.shape (50000,)
```

In [38]:
```
x_train.shape
```

Out[38]:
```
(50000, 32, 32, 3)
```

In [17]:
```
x_test.shape
```

Out[17]:
```
(10000, 32, 32, 3)
```

In [6]:
```
K = len(set(y_train))
print("number of classes:", K)
```

```
number of classes: 10
```

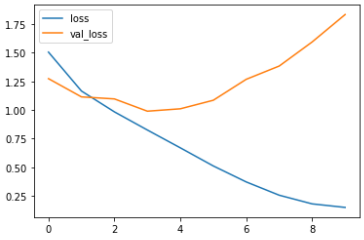
In [36]:
```
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)

x = Flatten()(x)

x = Dense(1024, activation='relu')(x)

x = Dense(K, activation='softmax')(x)

model = Model(i, x)
```

In [80]:
```
# Compile and fit
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
r = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10)
```

```
Epoch 1/10
1250/1250 [==============================] - 36s 28ms/step - loss: 1.5059 - accuracy: 0.4541 - val_loss: 1.2743 - val_accuracy:
0.5392
Epoch 2/10
1250/1250 [==============================] - 34s 28ms/step - loss: 1.1664 - accuracy: 0.5822 - val_loss: 1.1152 - val_accuracy:
0.6024
Epoch 3/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.9843 - accuracy: 0.6517 - val_loss: 1.0974 - val_accuracy:
0.6182
Epoch 4/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.8258 - accuracy: 0.7081 - val_loss: 0.9902 - val_accuracy:
0.6573
Epoch 5/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.6702 - accuracy: 0.7627 - val_loss: 1.0111 - val_accuracy:
0.6645
Epoch 6/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.5120 - accuracy: 0.8212 - val_loss: 1.0857 - val_accuracy:
0.6573
Epoch 7/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.3723 - accuracy: 0.8700 - val_loss: 1.2688 - val_accuracy:
0.6575
Epoch 8/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.2565 - accuracy: 0.9097 - val_loss: 1.3845 - val_accuracy:
0.6660
Epoch 9/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.1808 - accuracy: 0.9373 - val_loss: 1.5949 - val_accuracy:
0.6529
Epoch 10/10
1250/1250 [==============================] - 34s 27ms/step - loss: 0.1499 - accuracy: 0.9487 - val_loss: 1.8342 - val_accuracy:
0.6558
```

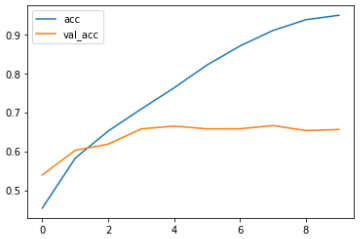
In [81]:
```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

Out[81]:
```
<matplotlib.legend.Legend at 0x134d19aa340>
```



In [82]:
```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc')
plt.plot(r.history['val_accuracy'], label='val_acc')
plt.legend()
```

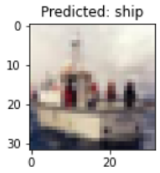
Out[82]:
```
<matplotlib.legend.Legend at 0x134d1c0abe0>
```



In [21]:
```
labels = '''airplane
            automobile
            bird
            cat
            deer
            dog
            frog
            horse
            ship
            truck'''.split()
```

In [28]:

```
selected = 88


res = model.predict(x_test[selected:selected+1])

arr = np.array(res[0])
result = np.where(arr == np.amax(arr))

# print(result[0])
plt.figure(figsize=(12,2))
plt.imshow(x_test[selected], cmap='gray')

plt.title("Predicted: %s" % (labels[int(result[0])]));
```




```
The model gives Accuracy of about 96% for categorising the testing samples. The concept is followed for CNN starting with
convolution layer and then dense layer, converting to a vector and finally using activation function to classify the
categories.
```

Conclusion: A CNN network is built using keras and tesnorflow library in python for classifying labels of thousands of pictures in different categories. The dataset consists 10000 images in test and 50000 images in train whereas x_train and x_test consists images and y_train and y_test consists labels of the images. A CNN network is built with first 3 layers as Convo2D layers which deals with input images. 32 in first 64 in second and 128 in the third layer are the number of filters used in each layer respectively. Kernel size is 3 here which means the filter matrix is of 3*3 order. Between Convo2D and Dense layer there is flattened layer which provides a 1D array of the filtered matrix and it serves as a connection between convolution layer and dense layer.

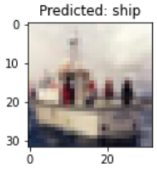
Compiling model takes three parameters i.e optimizer, loss and metrics of accuracy. After first epoch the accuracy was 45.41% and validation accuracy came out to be 53.92, the accuracy increases with every epoch and after 10 epochs the accuracy came out to be 94.87% whereas the validation accuracy after 10^{th} epoch is 65.58%.

The difference between accuracy and validation accuracy tells us that the model is overfitting.

Most of the labels are correctly predicted but there are some cases where the prediction is wrong as the model in not 100% accurate.

Output with correct prediction and wrong prediction is shown below:

```
selected = 88


res = model.predict(x_test[selected:selected+1])

arr = np.array(res[0])
result = np.where(arr == np.amax(arr))

# print(result[0])
plt.figure(figsize=(12,2))
plt.imshow(x_test[selected], cmap='gray')

plt.title("Predicted: %s" % (labels[int(result[0])]));
```




```
selected = 4


res = model.predict(x_test[selected:selected+1])

arr = np.array(res[0])
result = np.where(arr == np.amax(arr))

# print(result[0])
plt.figure(figsize=(12,2))
plt.imshow(x_test[selected], cmap='gray')

plt.title("Predicted: %s" % (labels[int(result[0])]));
```