Experiment No. 8 - Implement and test CNN for object recognition

Convolutional Neural Network

A CNN is composed of a series of layers, where each layer defines a specific computation. Layers are the basic building blocks of neural networks in Keras. A layer consists of a tensor-in tensor-out computation function (the layer's call method) and some state, held in TensorFlow variables (the layer's weights).

In this example, the Keras layers are used to create a CNN:

- Conv2D(Convolution layer) - This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs
- Dense(Core layer) - Just your regular densely-connected NN layer
- Flatten(Reshaping layer) - Flattens the input. Does not affect the batch size.
- Dropout(Regularization layer) - The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.
- MaxPooling2D(Pooling layer) - Downsamples the input representation by taking the maximum value over the window defined by pool_size for each dimension along the features axis. The window is shifted by strides in each dimension.

Algorithm

1. Import necessary library
2. Load the data
3. Normalisation of data
4. CNN Model
5. Training the model
6. Training loss vs Validation loss
7. Training accuracy vs Validation accuracy

1. Import Library

In [1]:
```
# baseline model with dropout on the cifar10 dataset
import sys
from tensorflow.python import keras
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, Dense, Flatten, Dropout, MaxPooling2D
import numpy as np
from IPython.display import Image
```

```
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])

Bad key "text.kerning_factor" on line 4 in
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
or from the matplotlib source distribution
Using TensorFlow backend.
```

2. Load the data

In [2]:
```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = cifar10.load_data()
    X = np.vstack((trainX, testX))
    Y = np.vstack((trainY, testY))
    # one hot encode target values
    Y = to_categorical(Y)
    return X, Y

# load dataset
X, Y = load_dataset()
```

3. Normalisation of data

In [3]:
```
# scale pixels
def prep_pixels(data):
    # convert from integers to floats
    data_norm = data.astype('float32')
    # normalize to range 0-1
    data_norm = data_norm / 255.0
    # return normalized images
    return data_norm

# prepare pixel data
X = prep_pixels(X)
print('Preprocessing Completed')
```

```
Preprocessing Completed
```

4. CNN Model

In [4]:
```
# define cnn model
def define_model():

    img_rows = 32
    img_cols = 32
    dim = 3
    num_classes = 10

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',input_shape=(img_rows, img_cols, dim)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(num_classes, activation='softmax'))
    # compile model
    model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer='adam',
              metrics=['accuracy'])
    return model


# define model
model = define_model()
print('Define Model Completed')
```

```
WARNING:tensorflow:From c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Define Model Completed
```

5. Training the model

In [5]:
```
# fit model
#history = model.fit(trainX, trainY, epochs=10, batch_size=64, validation_data=(testX, testY), verbose=0)
history = model.fit(X, Y, epochs=4, batch_size=32, validation_split=0.2)
print('Model Fit Completed')
```

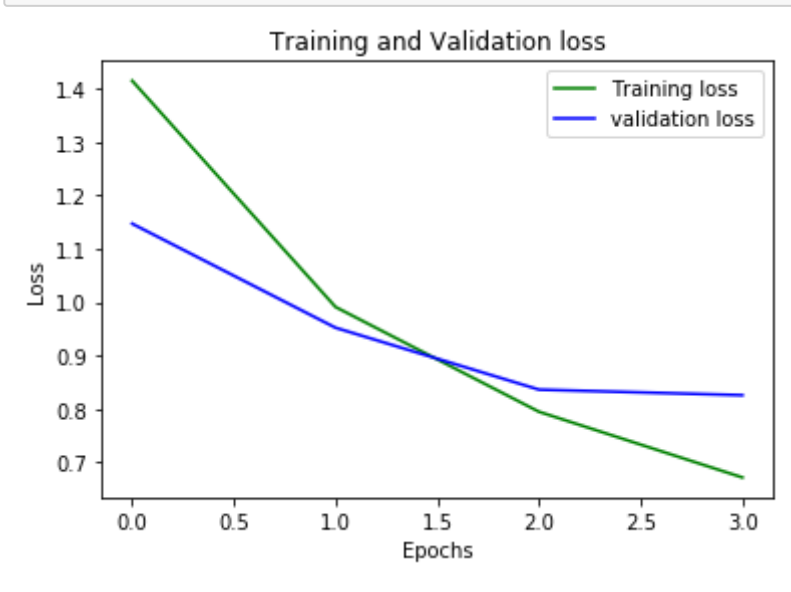
```
WARNING:tensorflow:From c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 48000 samples, validate on 12000 samples
Epoch 1/4
48000/48000 [==============================] - 13s 269us/step - loss: 1.4150 - accuracy: 0.4882 - val_loss: 1.1473 - val_accuracy: 0.5926
Epoch 2/4
48000/48000 [==============================] - 12s 247us/step - loss: 0.9912 - accuracy: 0.6513 - val_loss: 0.9524 - val_accuracy: 0.6669
Epoch 3/4
48000/48000 [==============================] - 12s 249us/step - loss: 0.7951 - accuracy: 0.7209 - val_loss: 0.8365 - val_accuracy: 0.7158
Epoch 4/4
48000/48000 [==============================] - 12s 245us/step - loss: 0.6720 - accuracy: 0.7648 - val_loss: 0.8261 - val_accuracy: 0.7256
Model Fit Completed
```

6. Training loss vs Validation loss

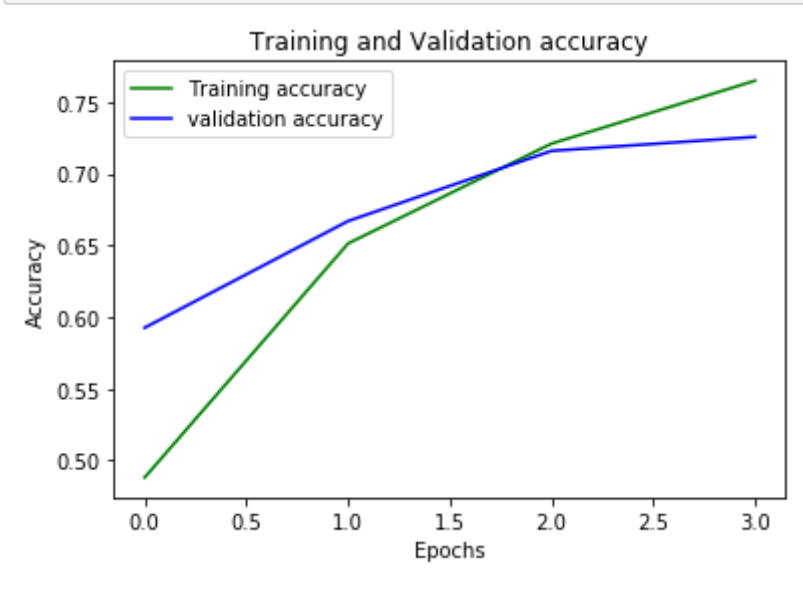
In [6]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], 'g', label='Training loss')
plt.plot(history.history['val_loss'], 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



7. Training accuracy vs Validation accuracy

In [7]:
```
plt.plot(history.history['accuracy'], 'g', label='Training accuracy')
plt.plot(history.history['val_accuracy'], 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Questions

1. What are advantages of CNN over ANN?

Ans:-

- Data of CNN is Image data whereas tabular data is of ANN. So CNN is higher application in the field of Medical Image Analysis, Computer Vision etc.
- CNN has parameter sharing but ANN don't.
- CNN has Spatial relationship and ANN don't.

2. What are the building blocks of CNN?

Ans:- The basic building blocks of CNN are:

Convolution layer - a "filter", sometimes called a "kernel", is passed over the image, viewing a few pixels at a time (for example, 3X3 or 5X5). The convolution operation is a dot product of the original pixel values with weights defined in the filter. The results are summed up into one number that represents all the pixels the filter observed.

Activation layer - the convolution layer generates a matrix that is much smaller in size than the original image. This matrix is run through an activation layer, which introduces non-linearity to allow the network to train itself via backpropagation. The activation function is typically ReLu.

Pooling layer - "pooling" is the process of further downsampling and reducing the size of the matrix. A filter is passed over the results of the previous layer and selects one number out of each group of values (typically the maximum, this is called max pooling). This allows the network to train much faster, focusing on the most important information in each feature of the image.

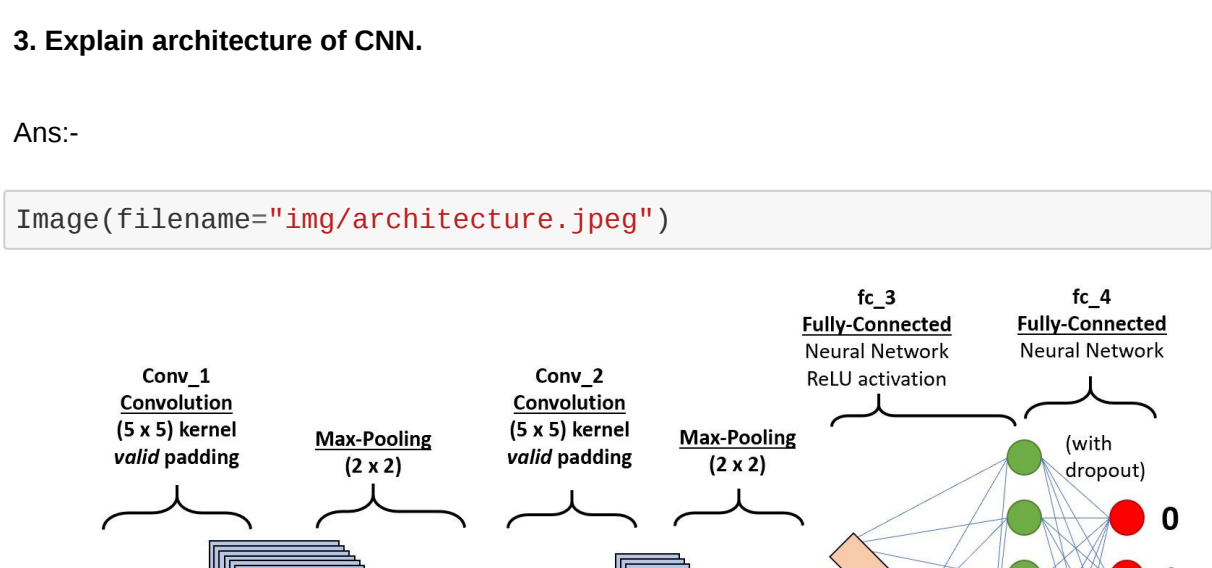
Fully connected layer - a traditional multilayer perceptron structure. Its input is a one-dimensional vector representing the output of the previous layers. Its output is a list of probabilities for different possible labels attached to the image (e.g. dog, cat, bird). The label that receives the highest probability is the classification decision.

3. Explain architecture of CNN.

Ans:-

In [8]:
```
Image(filename="img/architecture.jpeg")
```

Out[8]:



Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

4. State advantages and limitations of CNN.

Ans:-

Advantages

- CNN learns the filters automatically without mentioning it explicitly. These filters help in extracting the right and relevant features from the input data.
- CNN captures the spatial features from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. They help us in identifying the object accurately, the location of an object, as well as its relation with other objects in an image.
- CNN also follows the concept of parameter sharing. A single filter is applied across different parts of an input to produce a feature map.

Limitations

- CNN do not encode the position and orientation of object
- Lack of ability to be spatially invariant to the input data
- High computational cost
- Lot of training data

5. List applications of CNN.

Ans:-

- Agriculture
- Self-driving cars
- Surveillance
- Healthcare
- Face recognition
- Scene labelling
- Image classification
- Action recognition
- Human pose estimation
- Document analysis

References

[1] https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

[2] https://www.pluralsight.com/guides/data-visualization-deep-learning-model-using-matplotlib

Author Name:- Hemant Ghuge
LinkedIn:- https://www.linkedin.com/in/hemantghuge/
GitHub:- https://github.com/HemantGorakshGhuge