CS6005 DEEP LEARNING TECHNIQUES

ASSIGNMENT - 2

IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORK IN PYTHON

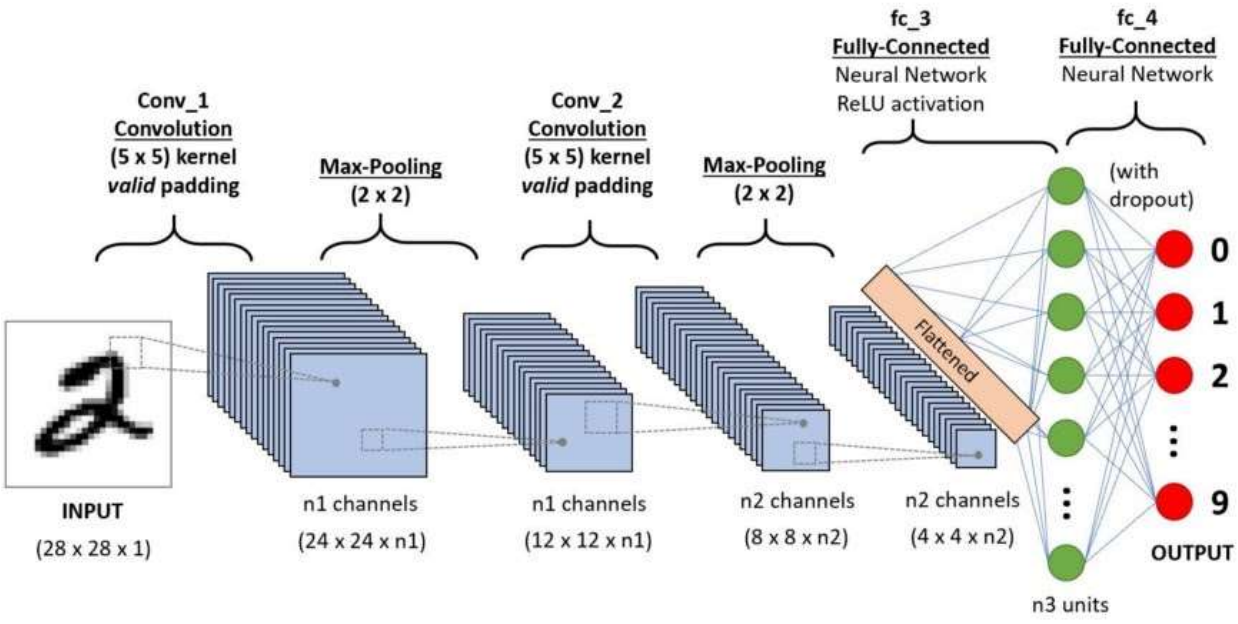
DONE BY

SABHARI P

2018103582

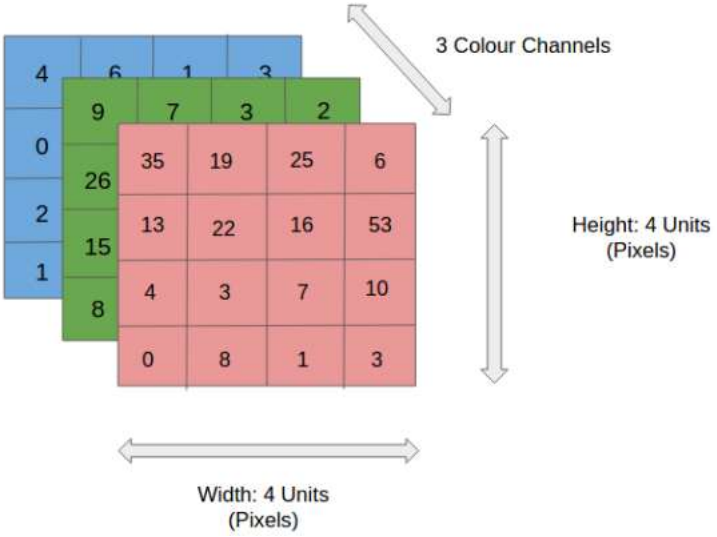
CONVOLUTIONAL NEURAL NETWORK:

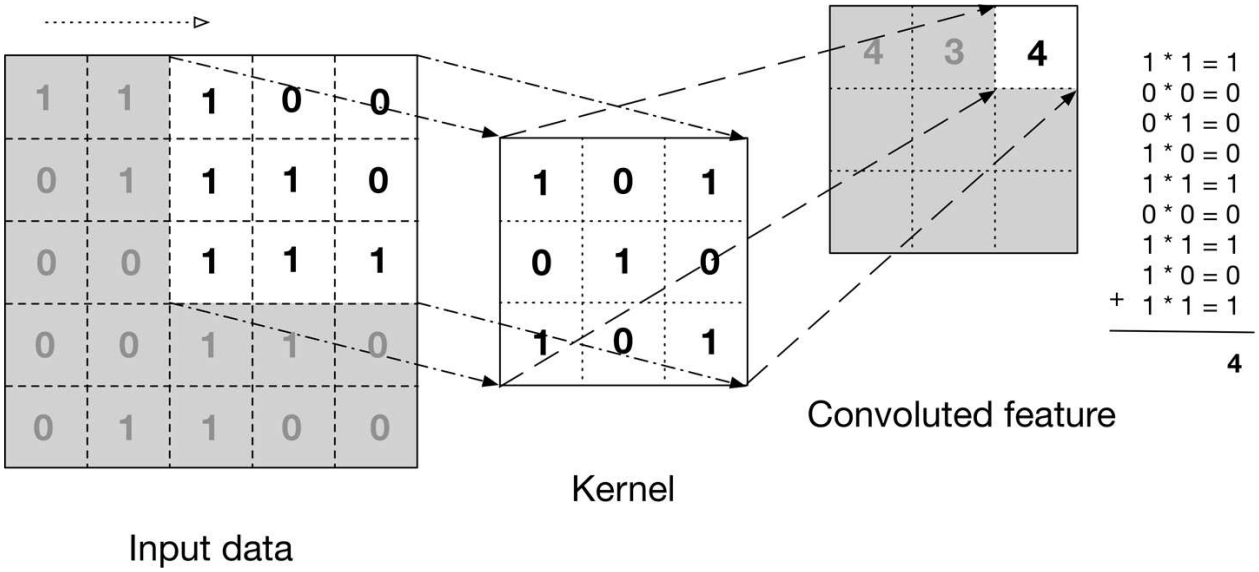
CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc. The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources. This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.

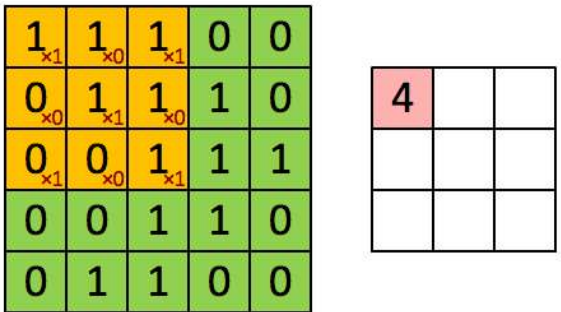


WORKING OF CNN:

An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.

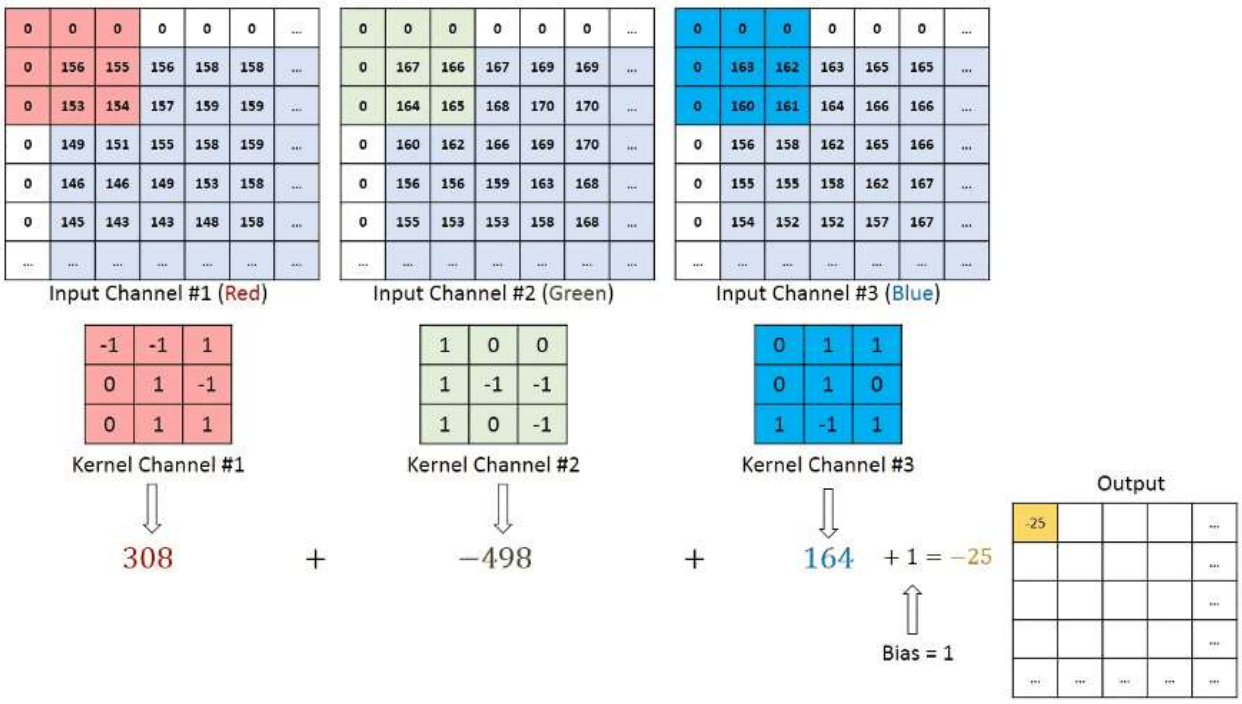






The above image shows what a convolution is. We take a filter/kernel(3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.

In the case of RGB color, channel take a look at this animation to understand its working



Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their

biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, etc.

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a “class.”

IMPLEMENTATION OF CNN:

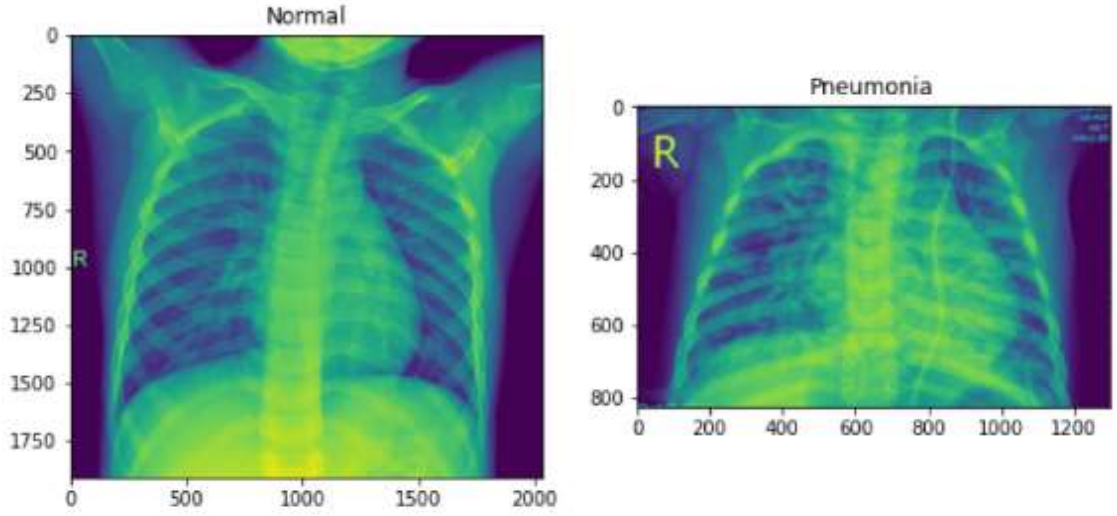
Dataset information:

The dataset has x-ray images of normal people and people affected by pneumonia. The purpose of this exercise is to see how accurate of a Neural Network we can create to classify X-Ray scans from patients with pneumonia.

IMPLEMENTATION:

1. Import the needed libraries.
2. Set path for training, test and validation set.
3. Let's plot a random xray of normal and affected people.

```
Out[5]: Text(0.5, 1.0, 'Pneumonia')
```



4. We declare our sequential model.
5. Add 2 convolution layers and 2 pooling layers. The pooling layers are placed at the end of the convolution layers.
6. Flaten the layers.

7. Then we add the fully connected layer with 1 input and output layer alone.
8. Compile the model with adam optimiser and binary crossentropy loss.Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.
9. Then we process the image for better fitting. Thus the summary of the model would look like below,

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

10. Now we fit the model with 10 epochs for training to observe training accuracy. Our training accuracy is **95.15%.**

```
Epoch 1/10
163/163 [==============================] - ETA: 0s - loss: 0.2478 - accuracy:
0.8984WARNING:tensorflow:Your input ran out of data; interrupting training. Mak
e sure that your dataset or generator can generate at least `steps_per_epoch *
epochs` batches (in this case, 624 batches). You may need to use the repeat() f
unction when building your dataset.
163/163 [==============================] - 75s 457ms/step - loss: 0.2478 - accu
racy: 0.8984 - val_loss: 0.7005 - val_accuracy: 0.6875
Epoch 2/10
163/163 [==============================] - 73s 449ms/step - loss: 0.2157 - accu
racy: 0.9137
Epoch 3/10
163/163 [==============================] - 72s 444ms/step - loss: 0.1923 - accu
racy: 0.9247
Epoch 4/10
163/163 [==============================] - 72s 443ms/step - loss: 0.1775 - accu
racy: 0.9354
Epoch 5/10
163/163 [==============================] - 72s 439ms/step - loss: 0.1789 - accu
racy: 0.9273
Epoch 6/10
163/163 [==============================] - 71s 435ms/step - loss: 0.1782 - accu
racy: 0.9314
Epoch 7/10
163/163 [==============================] - 72s 441ms/step - loss: 0.1517 - accu
racy: 0.9448
Epoch 8/10
163/163 [==============================] - 72s 441ms/step - loss: 0.1481 - accu
racy: 0.9436
Epoch 9/10
163/163 [==============================] - 71s 437ms/step - loss: 0.1403 - accu
racy: 0.9477
Epoch 10/10
163/163 [==============================] - 73s 449ms/step - loss: 0.1303 - accu
racy: 0.9515
```

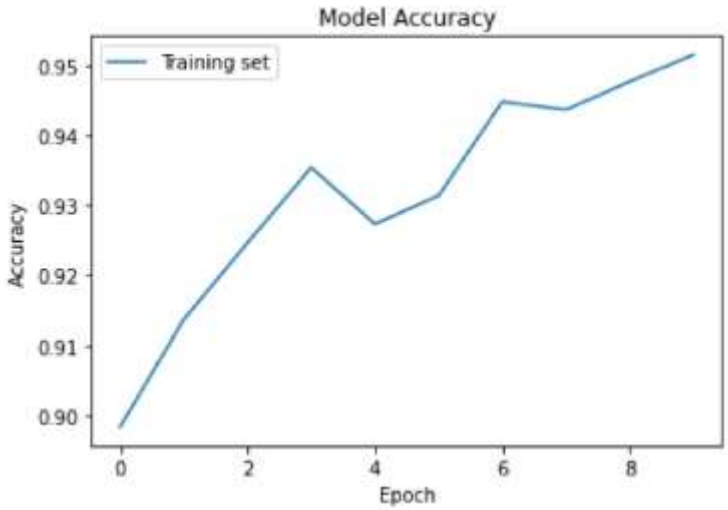
11. Then we evaluate for test accuracy and found it to be,

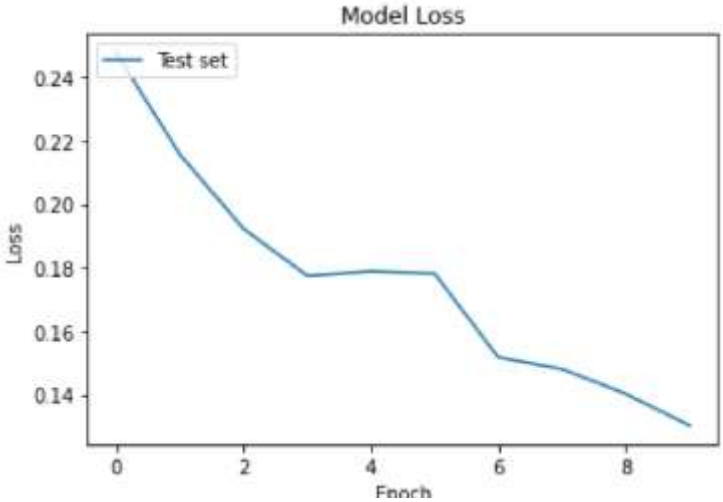
```
print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
The testing accuracy is : 87.5 %
```

Test accuracy is **87.5%**

12. We plot the accuracy and loss for inference,





We observe that the accuracy increases with increase in epochs.

TRAINING ACCURACY : 95.15%

TESTING ACCURACY : 87.5%

BELOW IS THE IMPLEMENTATION ON THE JUPYTER NOTEBOOK.

DATASET IS ALSO ATTACHED WITH THIS DOCUMENT FOR REFERENCE.