Machine Learning

(CSE-464)

“MLthon”

Submitted by:

Manzoor Ul Haq
(2018-UET-NML-CS-22)
Mahmood Yousaf
(2018-UET-NML-CS-11)
Abdullah
(2018-UET-NML-CS-16)

Course Instructor:

Dr. Malik Jahan

Date: **July 19, 2021**

Table of Contents

1. Abstract

The aim of this project is to have an exhaustive comparison between AlexNet and a CNN model of any other architecture with respect to their accuracy, recall, precision, and data splits. For that, we have implemented AlexNet and a CNN model of our choice. We used Python language due to its better libraries, algorithms, data structures and optimized memory management. There are bunch of models out there, but requirement is of classification of required data. Classification is to be done between healthy bell pepper and bacterial spotted bell pepper. Our data set contains only two sets of data so to make our model efficient, we used binary classification.

2. Introduction

We have implemented simple CNN model, LeNet-5 and an AlexNet CNN.

3. Classifiers

- Alex Net

1. Alex Net is a classifier for image classification, which is actually using convolutional neural network (CNN). Alex Net became famous when Alex Net won the ImageNet Large Scale Visual Recognition Challenge in 2012.We used Alex Net for binary classification to check health of pepper.
2. Below is the architecture of Alex Net, which we used for binary classification to check health of pepper

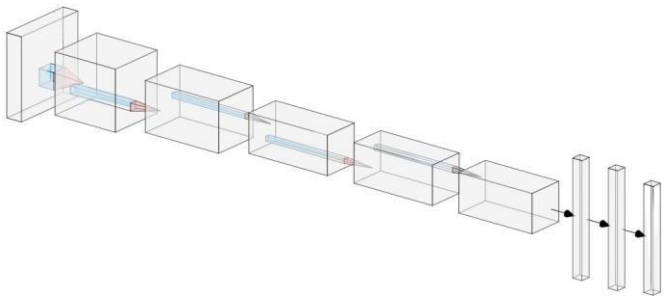


Figure 1: Alex Net Model style

- ## LeNet

1. It is one of the earliest models used for handwritten and printed characters recognition.
2. It is famous because of its simplicity and tends to perform image classification with multi-layer convolution neural network for image classification.

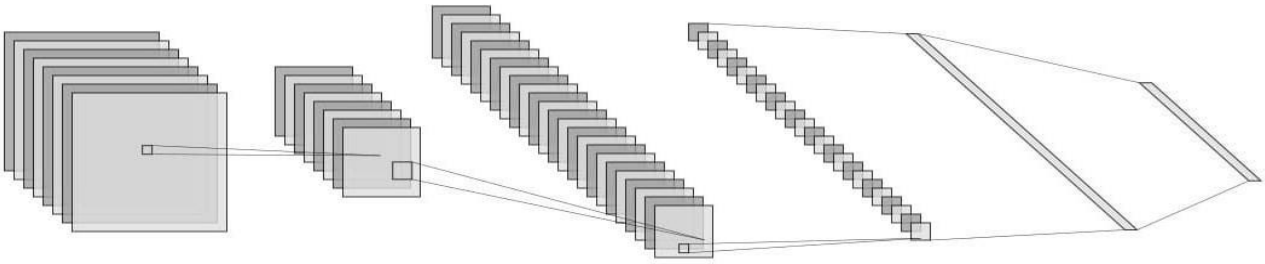


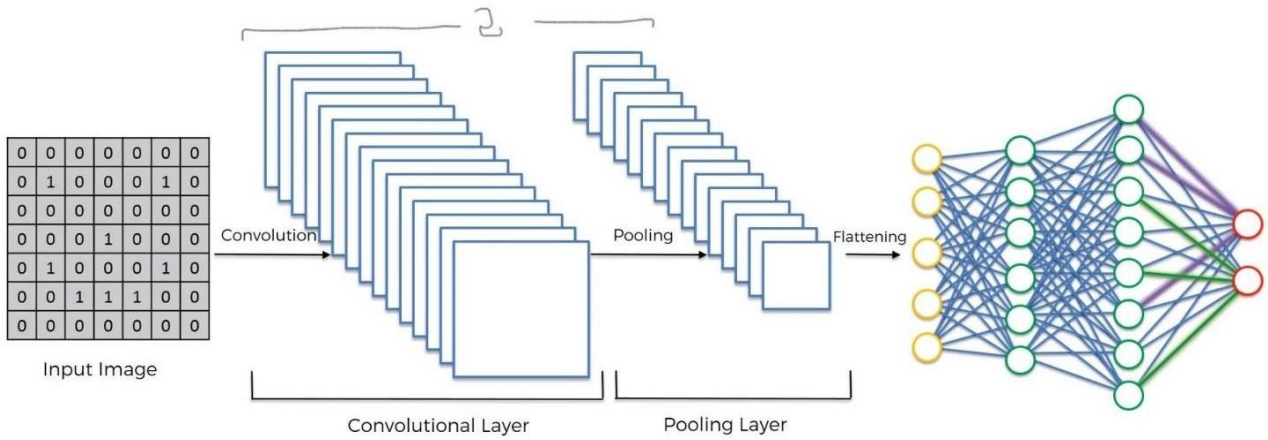
Figure 2: LeNet Model Style

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 40)      400
_________________________________________________________________
activation (Activation)      (None, 148, 148, 40)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 40)        0
_________________________________________________________________
dropout (Dropout)            (None, 74, 74, 40)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 50)        18050
_________________________________________________________________
activation_1 (Activation)    (None, 72, 72, 50)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 50)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 36, 36, 50)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 60)        27060
_________________________________________________________________
activation_2 (Activation)    (None, 34, 34, 60)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 60)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 17, 17, 60)        0
_________________________________________________________________
flatten (Flatten)            (None, 17340)             0
_________________________________________________________________
dense (Dense)                (None, 150)               2601150
_________________________________________________________________
activation_3 (Activation)    (None, 150)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 302
_________________________________________________________________
activation_4 (Activation)    (None, 2)                 0
=================================================================
Total params: 2,646,962
Trainable params: 2,646,962
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: LeNet Summary

- ## CNN Model

1. Two convolution layers used each with 32 filters, which exactly corresponds to the number of features; we want to find in an image. This provide us with the set of maps (feature maps).
2. In pooling, we are using pool size of 2.
3. The activation function in convolution layer is “Relu”. Increases the non-linearity and replaces all negative with zeros.
4. In the fully connected layer, there are 128 neurons with activation function “relu” and the last layer contains a single neuron with a “sigmoid function.



- *Figure 4: Custom CNN Model Style*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 45, 45, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1982592
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,992,865
Trainable params: 1,992,865
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Custom CNN Summary

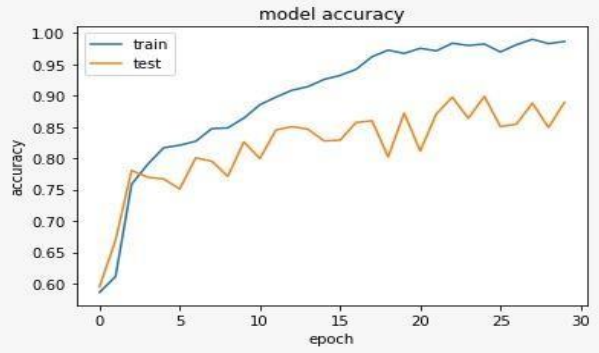
4. Data Cleaning & Pre-Processing

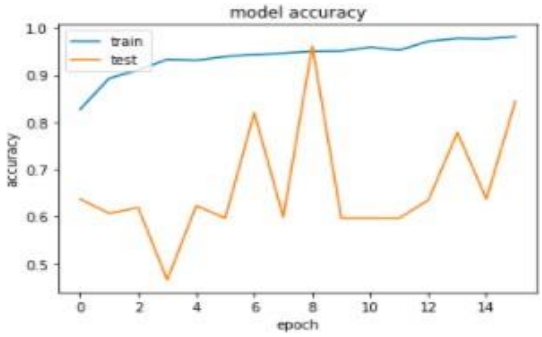
- CNN Model and Alex Net

1. For training dataset, we did some image augmentation, which includes recalling, resizing, flipping and zooming in some range. We add conventional values to it.
2. For training dataset, we did not performed augmentation else, we just rescaled it.

5. Comparison w.r.t Accuracy

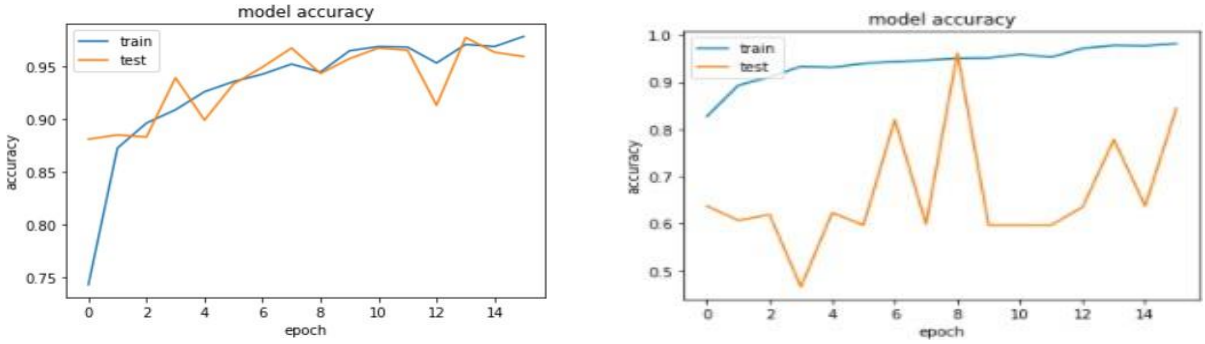
- **Alexnet vs LeNet**





The picture on the left is LeNet and the right-one is Alexnet, it is clearly seen able that AlexNet is much under fitted as compared to LeNet. LeNet graph has more linearity as well as more validation accuracy means our test data was more efficiently classified in LeNet.

- **Alexnet vs Custom CNN**



Picture on the left is Custom CNN and the right one is AlexNet, the same case is here, our Custom CNN performed much better than AlexNet, More accuracy plus linearity in the graph. Hence Alex was not able to perform well.

6. Comparison w.r.t Recall and Precision

Alexnet vs LeNet

```
Epoch 16/16
62/62 [==============================] - 80s 1s/step - loss: 0.0467 - accuracy: 0.9818 - precision: 0.9773 - recall: 0.9835
- val_loss: 2.8195 - val_accuracy: 0.8448 - val_precision: 0.8164 - val_recall: 1.0000
```

Figure 6: AlexNet Precision and Recall

```
Epoch 30/30
55/55 [==============================] - 53s 961ms/step - loss: 0.0351 - accuracy: 0.9867 - precision: 0.9879 - recall: 0.9758
- val_loss: 0.3567 - val_accuracy: 0.8896 - val_precision: 0.8756 - val_recall: 0.9100
```

Figure 7: LeNet Precision and Recall

Alexnet vs Custom CNN

```
Epoch 16/16
62/62 [==============================] - 80s 1s/step - loss: 0.0467 - accuracy: 0.9818 - precision: 0.9773 - recall: 0.9835
- val_loss: 2.8195 - val_accuracy: 0.8448 - val_precision: 0.8164 - val_recall: 1.0000
```

Figure 8: AlexNet Precision and Recall

```
62/62 [==============================] - 35s 560ms/step - loss: 0.0623 - accuracy: 0.9788 - precision: 0.9277 - recall: 0.9587 - val_loss: 0.0873 - val
_accuracy: 0.9597 - val_precision: 0.9292 - val_recall: 0.9595
```

Figure 9: Custom CNN Precision and Recall

7. Conclusion

We implemented three models AlexNet, LeNet and our own custom CNN and did many comparisons which are listed above.

Interesting thing to note that, the data preprocessing plus the data split were kept same for each architecture. Only difference was the change in architecture (Conv. Layers and their values).

Now, we concluded keeping in view the comparisons that AlexNet was unable to perform efficient classification which might be due to its much complex architecture trained on simple binary classification. On the other hand, Our Le-net was better with respect to validation on testing as well as on raw data but custom CNN outperformed from both models. Custom cnn got much more linearity in validation of test data plus on prediction of raw internet images. Hence, Custom CNN performed better than AlexNet and Le-Net.