IDML 730 E
Assignment 3: Study of AlexNet CNN

IIT2015113 - Faheem Hassan Zunjani
IIT2015505 - Souvik Sen
IIT2015511 - Sayantan Chatterjee

September 15, 2018

Abstract

In this experiment we studied the layers and network architecture of a Convolutional Neural Network (CNN) called AlexNet and have obtained a comparative representation of the performance with respect to the change in stride length, as well as their corresponding training times.

1 Introduction

AlexNet is a convolutional neural network, originally written with CUDA to run with GPU support, which famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% vs 26.2% (second place) top-5 error rates).

AlexNet has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. As of 2018 it has been cited over 25,000 times. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of GPUs during training.

2 Architecture

AlexNet was much larger than previous CNNs used for computer vision tasks (e.g. Yann LeCuns LeNet paper in 1998). It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs. Today there are much more complex CNNs that can run on faster GPUs very efficiently even on very large datasets.

The detailed archiecture of AlexNet CNN is illustrated in Figure 1.

AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers.

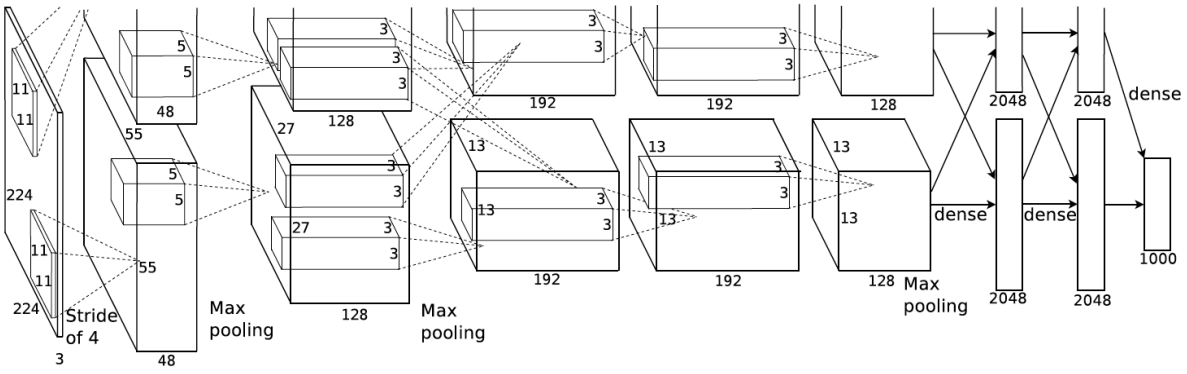


Figure 1: Architecture of AlexNet

Multiple Convolutional Kernels (a.k.a filters) extract interesting features in an image. In a single convolutional layer, there are usually many kernels of the same size. For example, the first Conv Layer of AlexNet contains 96 kernels of size 11x11x3. Note the width and height of the kernel are usually the same and the depth is the same as the number of channels.

The first two Convolutional layers are followed by the Overlapping Max Pooling layers that we describe next. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected layers. The second fully connected layer feeds into a softmax classifier with 1000 class labels.

ReLU nonlinearity is applied after all the convolution and fully connected layers. The ReLU nonlinearity of the first and second convolution layers are followed by a local normalization step before doing pooling. But researchers later didnt find normalization very useful. So we will not go in detail over that.

3 Experiment

Our experiment involves training the AlexNet CNN with variations in its strides and compare their performance, training time etc.

There are 3 stride parameters in the AlexNet model used during max-pooling. We have trained the network on 5 variations of stride parameters – the original network and 4 altered configurations.

The main model of the AlexNet architecture is described as follows:

```
network = input_data(shape=[None, 227, 227, 3])
network = conv_2d(network, 96, 11, strides=4, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 256, 5, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 384, 3, activation='relu')
network = conv_2d(network, 384, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 17, activation='softmax')
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)
```

The configurations of stride parameters we have used in our experiment are:

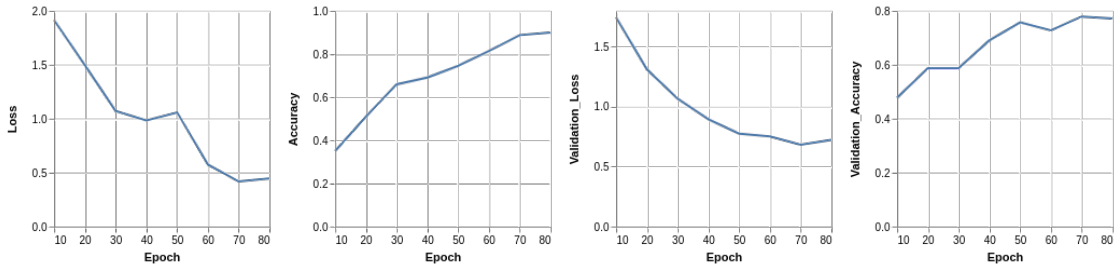
Model	Stride 1	Stride 2	Stride 3	Dropout 1	Dropout 2
Model 1	2	2	2	0.5	0.5
Model 2	3	3	3	0.5	0.5
Model 3	3	3	2	0.4	0.3
Model 4	3	2	2	0.4	0.3
Model 5	3	3	2	Dropconnect	

Note: The first row corresponds to the original parameters.

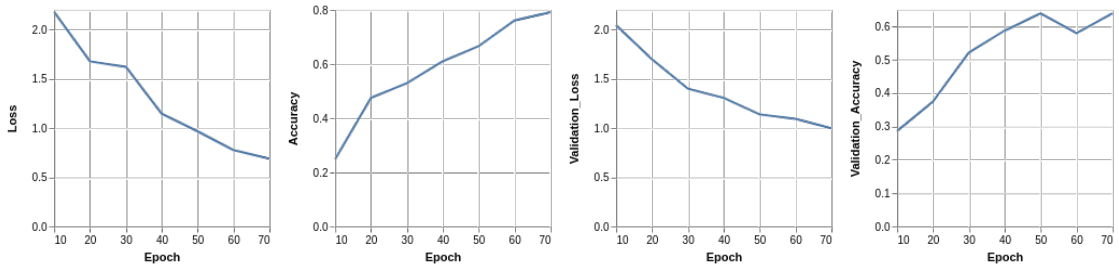
4 Analysis

In this section the results of the performances of all the trained models are illustrated.

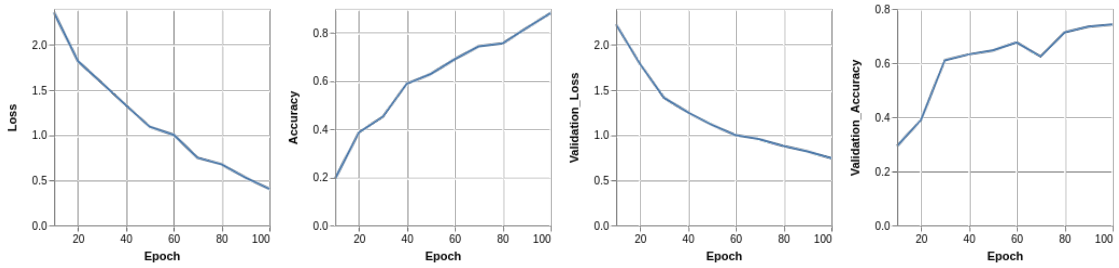
Figure 2 consists of the Epoch vs Loss, Accuracy, Validation Loss and Validation Accuracy plots for the said models.



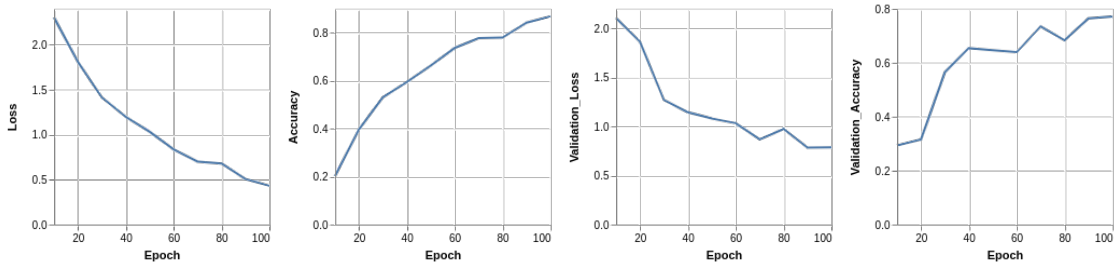
(a) Model 1: Strides (2, 2, 2), Dropout (0.5, 0.5)



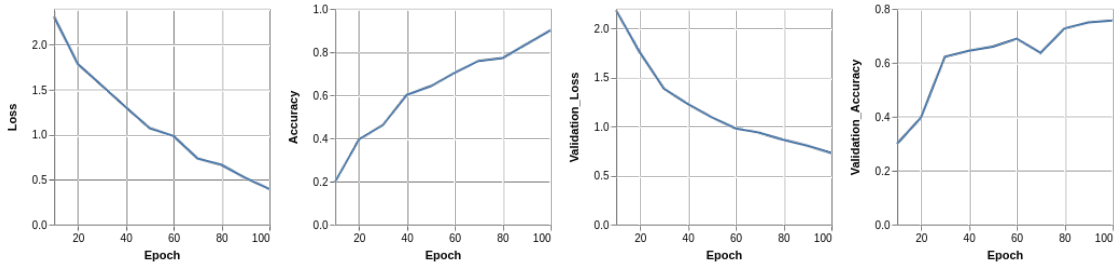
(b) Model 2: Strides (3, 3, 3), Dropout (0.5, 0.5)



(c) Model 3: Strides (3, 3, 2), Dropout (0.4, 0.3)



(d) Model 4: Strides (3, 2, 2), Dropout (0.4, 0.3)



(e) Model 5: Strides (3, 3, 2), Dropconnect

Figure 2: Performance analysis

5 Conclusions

Model 1, which has the original parameters, undergoes the steepest decrease in its loss function and also has no valleys in the validation accuracy plot with increasing epochs. It peaks at an accuracy 0.8 compared to other significantly lesser figures.

Model 2, which has the largest strides, has the least training time among the five models, but also sacrifices in accuracy at around 79% compared to the 91% accuracy of Model 1.

Model 3 and 4, with lesser dropouts are found to have comparatively more irregular validation accuracy curves with more peaks and troughs, not consistently increasing with increasing epochs.

Model 5 ditches the process of dropout and involves dropconnect with a similar set of strides as Model 3, but with a slight improvement in performance.