Momin Khan (A99112867)

COGS 181

Final Project Report

Abstract

This paper examines Convolutional Neural Networks and how different networks perform relative to each other. The models that will be used are VGG and ResNet.

Introduction

Convolutional Neural Networks, like Neural Networks, are made up of neurons that have weights and biases which can be learned from the data. They are deep, feed-forward artificial neural networks that have been utilized to analyzing visual imagery **(I.R.1).** One of the very first Deep Convolutional Neural Networks were introduced in 1994 by Yann LeCun. His insightful and pioneering work laid the basis of the fundamentals of Convolutional Neural Networks that we see today. The LeNet-5 architecture for deep CNNs Convolutional Neural Networks were first formally introduced to the world in 1994 Yann LeCun came up with LeNet-5 architecture for deep CNNs. It is remarkable that the architecture had insight that image features are distributed across an entire image, and convolutions with learnable parameters are an effective way to extract similar features at multiple locations with few parameters (I.R.2). LeNet was introduced in a time where there were no GPUs so computation times took relatively longer and

being able to cut down on the parameters and computation was an immediate advantage of CNNs. The Convolutional Neural Network uses a sequence of three layers: convolution, pooling and non-linearity. Convolution was first applied to the image to extract the spatial features of the image, then we would subsample using the spatial average of maps and then non-linearity was introduced using tanh or sigmoid activation functions. Finally, a multi-layer neural network (MLP) was used as a final classifier. Another key feature was sparse connection matrix between layers which helped avoid large computational costs. This network does serve as the building block of the many different architectures that we see present today. We will be working with the VGG16 model architecture and the ResNet model architecture and see how they behave with different tunings of their hyper-parameters. The results obtained can be used to draw inferences about these models and CNNs in general!

Method/Architecture

The different Convolutional Neural Network architectures that I used were the VGG model and the ResNet model. The VGG model was created at Oxford and was the first of its kind to use much smaller 3x3 filters in its convolutional layers. Furthermore, it was able to combine the convolutional layers as a sequence of convolutions. This went against the LeNet approach where similar features in an image were captured by using large convolutions. LeNet wanted to avoid using 1x1 convolutions on the first layers of the network and the great advantage that came with VGG was the introduction of using multiple 3x3 layers in a sequence that could emulate the effects of larger receptive fields like 5x5,7x7,9x9 and 11x11 for example. To represent the complex features, the VGG network used multiple 3x3 convolutional layers. We can see in blocks 3, 4, 5 that 256x256 and 512x512 3x3 filters are sued multiple times together to extract

more complex features. This amounts to having large 512x512 classifiers with 3 layers that are convolutional **(I.R.2).** This was not so computationally efficient as there would a large amount of parameters but it also meant increased learning power. The training of these networks became difficult and it had to be split into smaller networks with layers added one by one. The use of these 3x3 layers meant significantly more accurate ConvNet architectures. The convolution stride is fixed to 1 pixel and the spatial padding of convolutional layer input allows the spatial resolution to be retained after convolution. For spatial pooling, five max-pooling layers are used which follow some of the convolutional layers but not all of them. The Max-Pooling is performed over a 2x2 pixel window, with stride 2. After the convolutional layers, there are three Fully-Connected(FC) layers – the first two having 4096 channels each and the third layers performs 1000- way ULSVRC classification so it has 1000 channels for each class. The configuration of FC layers is the same for each class. The final layer is the softmax layer and the hidden layers all use ReLu activation functions.

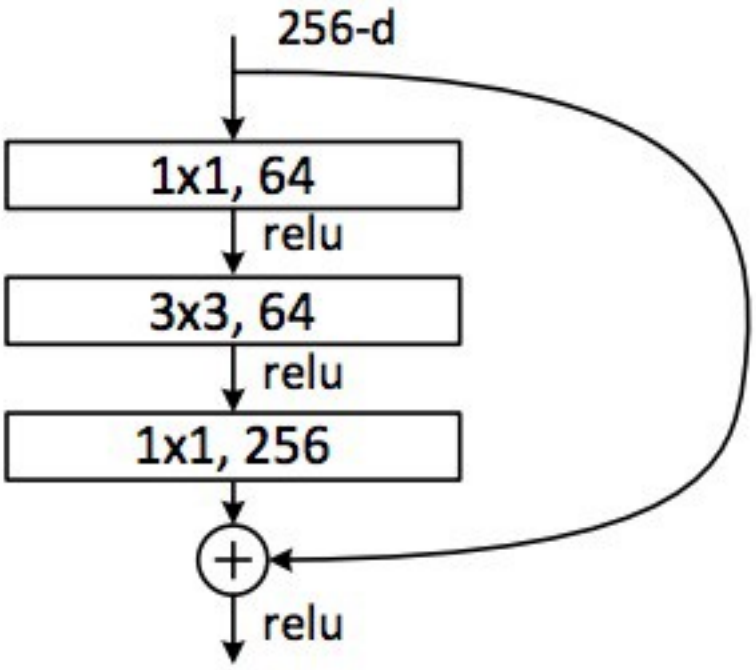
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

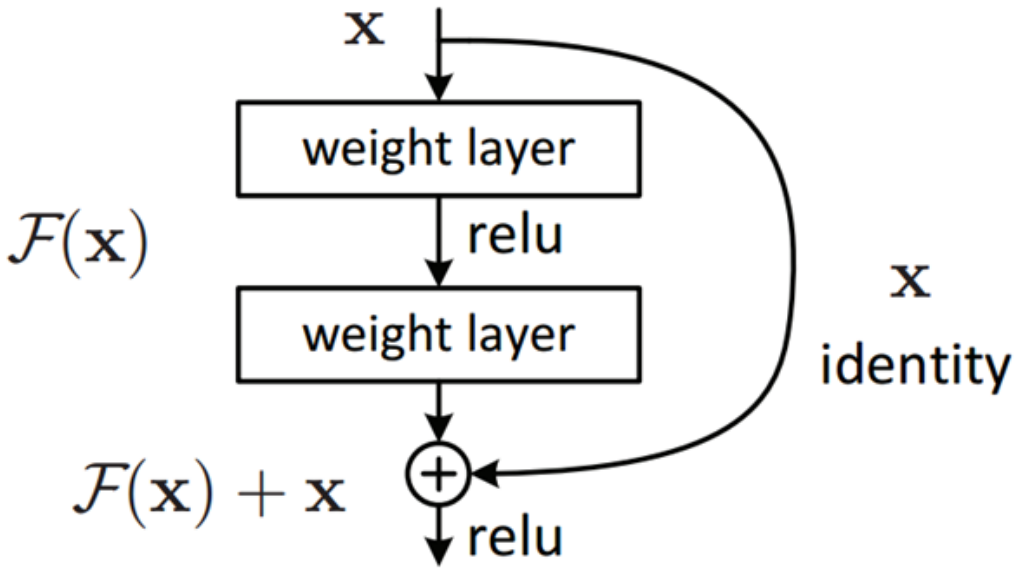
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

The ResidualNet model (ResNet) is a relatively new 152 deep convolutional neural network made by equal “residual blocks”. A constraint exists in deep networks (more than 30 layers) that they are not enough to do optimal end-to-end training. The idea that ResNet employs is to use blocks that reroute the input, and add to the concept learned from the previous layer **(I.R.7).** This was the very first time that a network with more than 1000 layers was trained. A ResNet with a large number of layers started to use a bottleneck layer. As seen in the diagram, the layer reduces the number of features at each input by first using a 1x1 convolutional with a smaller output, followed by a 3x3 layer and again a 1x1 convolution to a large number of features. This sort of

design allows the model to keep the computational costs low while also providing a lot of features. **(I.R.2)**





A residual block

The model, during learning the next layer, will also learn the concepts of the previous layer plus the input of that previous layer. In mathematical terms, the input x goes through a conv-relu-conv series and this gives us some F(x). This result is then added back to the original input x. That function can be defined as H(x) = F(x) + x. The difference between this and traditional CNNs is that in traditional CNNs H(x) = F(x) only. In ResNet, we are computing the term that you have to add, F(x), to your input. As in the words of the author, "it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping". This residual block might also be effective because during the backward pass of backpropogation, the gradient will be distributed because of the addition operations and will flow easily through the graph **(I.R.5).** The main noticeable featues of the ResNet were that it had 152 layers which was described as "Ultra-deep" by Yann LeCun and after the first 2 layers, the spatial size is compressed from an input volume of 224x224 to a 56x56 volume. It is said that ResNet model is the best CNN architecture that is out there today in the world and serves as a great innovation for the residual learning idea.

Experiments

The convolutional neural networks that were used in the experiment adopted the VGG16 architecture and the ResNet architecture. Both VGG16 and GoogleNet were trained and tested on the Cifar-10 dataset which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each containing 10000 images each. In the test batch, exactly 1000 randomly-selected images from each class exists. The classes are completely mutually exclusive.

The VGG16 model was used to run three experiments on the CIFAR-10 dataset. In the first experiment, the model was trained on the dataset using the original VGG16 paper specifications. Conventional hyper-parameters were defined as follows: Dropout = 0.5, Epochs = 250, Steps-per-epoch = len(xtrain)/batch_size, batch_size = 128, weight_decay = 0.0005, learning rate = 0.1 and learning_rate_decay = 1e-6. ReLu activation was used and the Stochastic Gradient Descent optimizer was used. For the pooling, MaxPooling was used in the experiment. After completion of the training and testing, the model gave a training accuracy of 97.67% and a training loss of 0.2527. On completion of testing, it gave a test loss of 0.4475 and a testing accuracy of 93.21%.

In the second experiment using VGG16, I kept the conventional hyper-parameters as defined above the same, and instead focused on using a different optimizer and activation functions. I

used the ADAM optimizer and sigmoid activation function, which greatly reduced the training and testing accuracy. After the training was done, a training accuracy of 9.94% was achieved with a training loss of 2.2563. After completing of testing, a testing accuracy of 10% was achieved with a testing loss of 2.5261.

In the third experiment, I kept all the conventional hyper-parameters the same except for the learning rate. I changed the learning rate to 0.2 from 0.1, and I used the SGD optimizer along with the ReLu Activation function. For pooling, instead of MaxPooling I used AvgPooling. I was able to get better results out of all the models with these specifications. The model produced a training accuracy of 98.66% with a training loss of 0.2032 and a testing accuracy of 93.44% with a testing loss of 0.4381. The testing accuracy came out better by just 0.23% than the first original model.

The ResNet 20 model was used to run three experiments on the CIFAR-10 dataset. For the first experiments, the conventional hyper-parameters were the following: batch_size = 256, epochs = 200, activation = relu, optimizer = ADAM, pooling = AvgPooling, num_filters = 16 and learning_rate = 0.001. The model after completing of the training process achieved a training accuracy of 99.55% with a training loss of 0.1292 and a test accuracy of 90.45% with a test loss of 0.5493.

In the second experiment, the ResNet model was run with the same hyper-parameters except for the following changes: the activation function that was used now was ‘sigmoid’ instead of ‘ReLu’, the optimizer that was used was Stochastic Gradient Descent instead of the ADAM

optimizer and the pooling was changed to MaxPooling2D from AvgPooling2D. The model after completion of the training process achieved a training accuracy of 30.31% with a training loss of 2.0229 and a test accuracy of 31.54% with a test loss of 2.006.

In the third experiment, the ResNet model was run with the same hyper-parameters as the first ResNet experiment with all the hyper-variables being kept the same. The only difference n this experiment was that the pooling technique used was MaxPooling instead of AvgPooling. The model after completion of the training process achieved a training accuracy of 99.62% with a training loss of 0.1359 and a test accuracy of 90.28% with a test loss of 0.5175.

Conclusion

In terms of the VGG model, the results that we got from the experiments showed that the first and third VGG experiment testing accuracy were really close to each other. With the original VGG architecture, the accuracy came out to be less than 0.23% than the third VGG experiment. This could be due to the different learning rate used in the third VGG model coupled with the use of the SGD optimizer and the sigmoid activation function. The second experiment of the VGG model showed that the ADAM optimizer along with the sigmoid function performed poorly on the data, giving us the lowest testing accuracy of 9.94% out of all the experiments performed in VGG or ResNet model. This leads us to conclude that Experiment 3 of the VGG model produced the best results on the CIFAR-10 Dataset.

In terms of the ResNet Model, the first experiment that we ran with the ResNet20 architecture produced the highest testing accuracy. In the second experiment, I modified the activation function, optimizer and pooling technique to be completely different than from experiment 1 and as expected, it performed relatively poorly giving us a test accuracy of just 31.54%. The third experiment was running with all the same parameters as the first experiment, except that the pooling technique that was used was MaxPooling instead of AvgPooling. We achievced a testing accuracy of 90.28% which came close to our original unmodified ResNet20 first experiment.

Between the two experiments, I think we can see that altering the different hyper-parameters on ResNet still allows us to achieve a better accuracy than VGG. As previously mentioned in the paper, ResNet is believed to be the best Convolutional Neural Network that exists today and I believe that the results that we got from the experiments also agree with this notion. I also think that the results might vary depending on the dataset that you are using, but the ResNet architecture is intriguing and something that I would definitely look more closely at going into the future.

All the individual experiments took between 2 to 2 and a half hour to run. I wanted to try more different parameter settings but due to time constraints I was not able to. I had to rerun the experiments a multiple number of times because the pod would get disconnected or I would lose my connection to the server leaving the experiment incomplete.

References

https://en.wikipedia.org/wiki/Convolutional_neural_network (I.R.1)

http://dataconomy.com/2017/04/history-neural-networks/ (I.R.2)

http://cs231n.github.io/convolutional-networks/ (I.R.3)

https://arxiv.org/pdf/1409.1556v6.pdf (I.R.4)

https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html (I.R.5)

https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3 (I.R.6)

https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/residual_net.html (I.R.7)

https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py