Applied AI COMP534

Third Assignment – Image Classification

Topic – Solving Image Classification Problems with Convolutional Neural Networks

Members – 1) Denis James 201603185

2) Aniruddha Guha 201590050

Introduction

This assignment consists of two goals:

(1) To compare the performance of two different CNN pretrained models studied.

(2) To propose a new model that can improve the performance of the best pre-trained model.

The two different CNNs that we chose for this assignment are GoogleNet and VGG net.

Data Description

This data set contains the x-rays of various people who either have covid or pneumonia and then there are some x-rays which are of people who are normal.

Hence, this is a great dataset to solve multiclass classification problems, or in our case perform image classification on. The data is separated into two files, Train data and Test data already. All of the images are stored in grayscale and they are all of the same size.

Libraries used

Some of the libraries that were used while working with these CNN models were –

1. Numpy
2. Matplotlib
3. Tensorflow
4. Keras
5. Sklearn

We used the Chest X-Rays dataset available at Kaggle at –

[**https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia?resource=download-directory**](https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia?resource=download-directory)

Dataset is organized into 2 folders (train, test) and both train and test contain 3 subfolders (COVID19, PNEUMONIA, NORMAL). Dataset contains a total 6432 x-ray images and test data have 20% of total images.

First we load the entire data from our GoogleDrive. Then we separate the data into train and test data, which is quite easy since they have already been provided in different folders. Once we go through and plot a few of the X-rays, we realised that not all images were of the same sizes. Hence, we had to size up all the X-rays to a common size of 224x224.

After we were done with that, we needed to make sure that to perform better we need to randomize the images before training the model. So, we had to do some data augmentations, such as flip the images randomly, rotate the image and even perform some zoom into the a random part of the image. There was no need to clean the data in our case since the data was already clean and sorted out as well. We analysed the images by plotting some of the X-Rays from the train folder to see what kind of images we are dealing with.

The next step was to choose which Convolutional Neural Network would we use for our project. For this purpose, we ran the scenario we had at hand and trained the model using 3 various types of CNNs to test which one would best suit our needs and data here. Initially we tried out the CNN architectures such as VGG16, VGG19, AlexNet and GoogleNet. After this trial, we finally selected the architectures AlexNet and GoogleNet as our choice of cnn models since the train time in Googlenet and Alexnet was far less than compared to the VGG networks and even some other architectures while providing us with a better performance and an higher accuracy.

While using the pretrained AlexNet and GoogleNet models, we didn’t just call the pre defined function for the models. Instead we represented all the layers of the model how they are supposed to be so that it would be easier for us and anyone else using the code to understand the layers and the architecture of the specific models that we are working with. When you call the pre-trained model straight away using the predefined function, that’s the layer’s structure that is actually being used in the background so we can even compare the architecture and all the layers of the models we used here for better understanding of the flow.

Diagram

Description automatically generated

The GoogLe Net CNN is a brilliant convolutional Architecture which is comparatively deeper than all the other networks that exist which enables it to learn non-linear features very efficiently. But unlike all the other deep convolutional neural networks, this network has lesser parameters to train and also takes lesser time to train in spite of it's multi-layer structure. This architecture is first of it's kinds to implement the Inception module which consists of the the input being fed forward into four different convolutional layers,

a 1\*1 conv layer

a 1\*1 conv and then a 3\*3 layer

a 1\*1 conv and then a 5\*5 layer

a 3\*3 max pooling and then a 1\*1 layer

The output from all these four blocks are stacked together under the process called filter concatenation. Implementing such a block onto our neural network reduces our computational expense massively. This makes our network wider instead of making it deeper and increasing complexity.

Say we were to train the deep network only on the basis of what the final output of the network is, then, due to the CNN being deep, the initial layers would go through a very negligible change in weights. When the error is backpropogated through so many multiple layers, the gradient value becomes negligible and the initial layers stop learning. To prevent this we branch out the network from the middle after every couple of inception modules and then forcefully train them on the basis of the outputs they give on the basis of these branches. These branches are called auxilary classifiers.

The implementation of these two improvements to our model instills significant change in the output the Network produces. Which is what resulted in the GoogLe Net winning the ILSVRC competion in 2014.

Diagram

Description automatically generated

AlexNet is another CNN architecture that won the ILSVRC12’ competition. This is a 16 layers deep convolutional neural network that implements multiple pooling layers and convolutions one after the other and ends with three final densely connected layers.

The input image is passed through the following layers in the order from top to bottom

an 11×11 conv layer with 4 strides

a max-pooling layer with 3×3 kernel 1 stride and 2 padding

an 5×5 conv layer with 2 strides

a max-pooling layer with 3×3 kernel 2 strides

a conv layer with 3×3 kernel 1 padding

a conv layer with 3×3 kernel 1 padding

a conv layer with 3×3 kernel 1 padding

a max-pooling layer with 3×3 kernel and 2 strides

a densely connected layer with 4096 neurons

a dropout layer with rate 40%

a densely connected layer with 4096 neurons

a dropout layer with rate 40%

a densely connected layer with 1000 neurons

AlexNet is an incredibly powerful model capable of achieving high accuracies on very challenging datasets. However, removing any of the convolutional layers will drastically degrade AlexNet’s performance. AlexNet is a leading architecture for any object-detection task and may have huge applications in the computer vision sector of artificial intelligence problems. In the future, AlexNet may be adopted more than CNNs for image tasks.

We already had the data divided into folders of train data and test data, so easily got the data from there. Then for each run we created batches of 32 from the train dataset and the test dataset as well. So, for each run the network got trained and tested on a batch of 32 samples at one time. And since the training and testing of the data took soo long, we only went 10 episodes for our models.

To train the networks, we first have to just compile it where we provided it with some values for the loss function and the optimization algorithm. After that, we can also provide a model summary which provides us more insight into the composition of the layers for our network that we are using at that moment. Once this is done, we can use the fit method to train the network along with it passing the required arguments such the training dataset, the validation dataset and the number of episodes. The training process then takes a long time. In our case, it took us about 3-4 hours each to evaluate the GoogleNet and the AlexNet networks.

Evaluation

After we ran the AlexNet model, we got the metrics after each episode.

The plots for the train/test accuracy and the train/test loss for the model were as follows –

A picture containing polygon

Description automatically generated

We can see that while training the model, the accuracy remains mostly constant starting with 81.3% at epoch 1 and ending up with 87.8% at epoch 10, and peaking with 92.2% at epoch 9.

When we start with testing of the model, the accuracy for it is extremely low initially with just an accuracy of 9% at epoch 1 but this suddenly spikes up to 81.8% at the very next episode, ending with an accuracy of 89.3% at epoch 10 and peaking at epoch 9 with an accuracy of 92.2%.

A similar trend can be noticed when we check the loss function for the model. While training the model, the loss for it is really low and stays that way throughout the training process. But while testing the model, the loss is initially very high and then has a sharp drop from the next epoch, and remains nearly the same similar to the training loss.

After that, we ran the GoogleNet model for which we got the metrics after each episode.

The plots for the train/test accuracy and the train/test loss for the model were as follows –

Chart, line chart

Description automatically generated A screenshot of a computer

Description automatically generated with low confidence

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

The plots above show us the accuracy and loss for all the three branches that branched out including the final output.

When we train the GoogleNet model, we can see that the first branch got the final accuracy of 93.7% while peaking with 95.05% at epoch 9. Similarly, the second and the third branches get the final accuracies of 94.27% and 94.14% respectively while both peaking at epoch 9 with 94.68% and 94.54% respectively.

Now if we look at the testing of the network, we see that the first branch gets a final accuracy of 92.46% while peaking at 93.94% at epoch 9. Similarly again, the second and the last branches get the final accuracies of 91.53% and 90.37% respectively while peaking at epoch 9 at 94.4% and 94.79% respectively.

Looking at the performance of both the networks, we notice that both of them had quite good results with only a difference of 3-4% in accuracies while testing the models at most times. Yet by a small fraction, the GoogleNet model performed better than the AlexNet. GoogLeNet training is also slightly faster than that of AlexNet, although AlexNet saves us a lot of memory space. The performance of GoogLeNet is the best outdoing than AlexNet on various parameter including time, accuracy, dropout, and the initial learning.

Since we got the best network as the GoogleNet, hence we started to create a new model using the GoogleNet model as the base network for it. In our new model, we took inspiration from the GoogleNet and kept the architecture quite similar to it. But we added another extra network and inception block to it, hoping that adding more layers would help us get finer results and increase the final accuracies of the model. Since we added another inception block to the our network, hence we got four branches branching out of the network including the final output.

After we ran our model, we got some metrics after each episode.

The plots for the train/test accuracy and the train/test loss for the model were as follows –

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

When we train our model, we can see that the first branch got the final accuracy of 95.07%. Similarly, the second, the third and the fourth branches get the final accuracies of 95.34%, 95.20% and 95.38%.

Now if we look at the testing of the network, we see that the first branch gets a final accuracy of 93.94% while peaking at 94.79% at epoch 9. Similarly again, the second and the last branches get the final accuracies of 94.95%, 94.25% and 94.48% respectively while peaking at epoch 9 at 95.1%, 94.64% and 94.72% respectively.

Also, in our model the value of loss is quite consistent when compared to the loss values of GoogleNet.

By looking at the accuracies of our model and that of the GoogleNet model, we can clearly see that there has been some improvements of about 2-3% in terms of performance of the network.

Also looking at the plots, we can see that the accuracies of our model are a lot more consistent and smoother when compared to the plots of the GoogleNet model. Hence, our model outperformed both, AlexNet as well as the GoogleNet networks.

According to our knowledge, the biggest factor behind this increase of performance is the addition more layers to the network. Increasing the number of layers can often greatly increase the capacity of the model. For example if we have a model with 2 layers and then also a model with 5 layers, then we can clearly understand that the latter model will have far greater capacity than the first model. Although, we should be careful to not add too many layers as doing so may cause our model more harm than good. Adding to this is that the model with more layers than required is likely to overfit while training, hence leaving the model unable to learn from the training data.

In the future, we can try to find out how many inception blocks can we add to increase the performance before reaching the point where we start overfitting the data. Also, we can try to change the architecture a little bit more by moving around the layers within the architecture or even maybe adding a few more convolutional layers to the inception blocks to see how it might affect the model and our data.

Challenges Faced

We had to first analyze the data given to us and understand how the data was distributed into various folders and how to use it to use for the pretrained models along with our own model as well. We had a huge number of images of the X-Rays which consisted to X-Rays of covid patients, pneumonia patients and also patients who are completely normal without either of those. Due to there being so many images, we were getting millions of parameters while training various models and hence as a result, it was taking us sometimes even over 15 hours just to run 5 epochs on some models. Due to this issue, we had to go for simpler architectures so that we could save as much time as possible on training and testing of the model.

Also, we had some understand all the models, their architectures and structures, and how do they exactly work and perform with various kinds of data juts so that we could choose any 2 of the models for the assignment. We had to understand how in detail the flow of the models and all it’s layers work, especially when working with GoogleNet since it’s such a complex network.

Also, while working on our network we had to understand and think about how to create it using the GoogleNet model as the base model, and then even increase the performance.

Task Allocation

In the python .py code file, we both worked together to get it all setup. Denis initially worked on creating the basic model architectures of the AlexNet and GoogleNet CNNs. Then, Aniruddha worked on creating a new CNN network which was based on the best network, i.e. GoogleNet in our case.

In the Report file, Aniruddha started off with the introduction section including the data description, libraries used, details of the pre trained models which were selected by us. Then Denis worked on the entire evaluation section where we gave a comparative analysis of the performances of the pre trained models, description of the new model that was proposed by us, discussion for the results we got from this new network and any more changes we could have made to it. Finally, Aniruddha worked on the explanation of the challenges that we faced while working on this project and then explaining the task allocation for both the members.

References

1. <https://stackoverflow.com/questions/47324571/keras-valueerror-negative-dimension-size-caused-by-subtracting-5-from-1>
2. <https://gist.github.com/joelouismarino/a2ede9ab3928f999575423b9887abd14>
3. <https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/#:~:text=The%20GoogleNet%20Architecture%20is%2022,of%20the%20full%20GoogleNet%20architecture>.
4. <https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/#:~:text=The%20GoogleNet%20Architecture%20is%2022,of%20the%20full%20GoogleNet%20architecture>.
5. <https://www.youtube.com/watch?v=AUH1hm30VG4>
6. <https://www.youtube.com/watch?v=7LQSdPjWjdA&t=336s>
7. <https://github.com/krishnaik06/Advanced-CNN-Architectures/blob/master/Transfer%20Learning%20Alexnet.ipynb>
8. <https://learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/>
9. <https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765>
10. <https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951>
11. <https://iq.opengenus.org/googlenet/>
12. <https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/>
13. <https://machinelearningmastery.com/how-to-control-neural-network-model-capacity-with-nodes-and-layers/>
14. <https://mohitjain.me/2018/06/06/alexnet/>
15. <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96>
16. <https://ojs.aaai.org/index.php/AAAI/article/view/10171#:~:text=We%20show%20that%20both%20GoogLeNet,discuss%20possible%20reasons%20for%20this>.
17. <https://datascience.stackexchange.com/questions/25267/keras-difference-beetween-val-loss-and-loss-during-training>
18. <https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia?resource=download-directory>
19. <https://github.com/priyavrat-misra/detect-covid19-from-xrays>