Exploratory Analysis of Convolutional Neural Networks for Chest X-Ray Image Classification

Derek Jiang, Sowmya Parthiban
Prof. Zhuowen Tu

Abstract:

Convolutional Neural Networks are being widely used in Computer Vision problems in recent times. With a vast majority of the world struggling with a global pandemic, CNNs are in need of the hour for faster medical diagnosis. This paper explores two closely related SOTA CNNs, VGG and ResNet models of different depths pre-trained on the ImageNet dataset in their ability to differentiate Chest X Ray images of healthy patients from those affected by COVID-19 over multiple training epochs. The images were found on multiple primary medical imaging databases as well as kaggle. The error metrics that were used for model evaluation were error-rate, sensitivity, and specificity. Using these error metrics, all models explored were found to perform exceptionally well.

Introduction:

The world in recent times has been struck by an inscrutable pandemic, COVID-19, due to a type of coronavirus called SARS-CoV-2. Several efforts have been made in different walks of science, and especially in Artificial Intelligence to study the disease for easier diagnosis and drug testing. Chest-X-Rays is one of the important tools that is used to determine lung-health in patients that have been affected by viral infections. These infections typically result in pneumonia in the lungs, the severity of which can be determined X-Ray images. In the past decade, Convolutional Neural Networks have evolved to show outstanding results in different areas of image analysis such as image classification and segmentation. The aim of this project is to deploy convolutional neural network models to help distinguish COVID-19-infected lungs from healthy lungs. We explored multiple architectures such as VGG and ResNet as well as different hyperparameters such as depth and training epochs to analyze how they affect classification error rate and other error metrics. All models we deployed have been proven to demonstrate really high classification accuracy results.

Method/Architecture Description:

VGG16 and 19:

VGG 16 as indicated by its name is a 16 layer deep CNN. One of the unique features of the VGG architecture compared to its predecessor models is that it utilizes smaller 3x3 receptive fields. Although previous models had used 3x3 receptive fields, they weren't as deep as VGG models - making them less discriminative in terms of image classification. It uses a convolution stride of 1 and 1 pixel of padding in order to preserve the size of the model after convolution. There are five layers of pooling between the convolutional layers with max-pooling being performed with a stride of 2 and a window size of 2x2. ReLU is used in the activation layers for non-linearity. A soft-max classification output is implemented in the final layer. The main difference between VGG 16 and VGG 19 is the depth - VGG 19 has 19 layers with 16 convolutional layers and 3 fully connected layers as compared to VGG 16 which has 13 convolutional layers and 3 fully connected layers in the end. While VGG 16 has 133 million parameters, VGG 19 has 144 million parameters.

ResNet:

Resnet models, although some of which are much deeper than VGG models, their architecture is designed in such a way that they are less complex than VGG. ResNet recognizes the need for extremely deep-layered networks for recognition of complex image features without overfitting and the problem of vanishing / exploding gradients. Although normalization of layers helps with convergence during back-propagation, the problem of degradation, accuracy saturation, requires another approach. ResNet solves this problem by constructing a deep residual learning framework. This involves short-cut connections from one layer to the next layer's next layer along with the already existing deep network connection. This enables the weight at each layer to be of a constant order, hence avoiding extreme gradients during back-propagation. These skip-connections are the difference between VGG and ResNet layers. These skip-connections do not add any extra parameters hence not increasing the complexity of the model. Despite their depth, ResNet models are much less complex than VGG models in terms of number of parameters. Similar to VGG models, ResNet models utilize 3x3 filters. Unlike VGG ResNet only has one pooling layer in the end implementing average pooling and only one fully-connected layer after that with softmax classification output. We chose to implement ResNet 34 and ResNet 101 as the other two models ResNet 50 and ResNet 152 are already widely implemented.

ResNet34: The architecture of ResNet34 is different from its deeper counterparts. It is divided into five main layers. The first layer is different from the rest in that it has 64 7x7 filters with a stride of 2 to help downsampling followed by a max-pooling layer. The rest of the blocks each are made of different number two convolutional layer blocks repeated 3, 4, 6, and 3 times as we go through each deeper layer. with ReLu activations in between. With each deeper block the number of convolution filters in the block doubles from 64 in Conv block 2 to 512 in Conv block 5. This is followed by average global pooling and a fully connected layer with softmax classification.

ResNet 101: The first two layers of ResNet 101 are the same as ResNet 34. The rest of the model consists of building blocks each having a bottleneck design. Each residual function has a stack of 3 layers - one layer with 3x3 filters in the middle, sandwiched by 1x1 convolutional layers. The 1x1 layers help in downsampling and upsampling images that go in and out of the 3x3 layer in the middle for reducing time complexity. There is a shortcut connection from the output of the last 1x1 layer of the previous stack to the first layer of the next stack. Multiple such stacks with an increasing number of filters at each layer starting from 64 to 512 in the first two layers of the stack and 256 to 2048 in the upsampling 1x1 convolutional layers make up the main architecture of the model. Similar to ResNet 34, this is followed by an average pooling layer, a fully-connected layer and a soft-max classification layer in the end.

Experiments:

Data and Problem Description:

Collecting data for this project was the most challenging part. Our dataset of pneumonia-affected lung images had multiple sources of infections. Despite millions of people across the world having been diagnosed with COVID-19, there is surprisingly a scarcity of imaging data related to the disease that is readily available on the internet. Our primary source of data was a kaggle - dataset https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset which has scrapped chest x-ray images from multiple sources across the world. This dataset contains images of healthy lungs as well as images of lungs infected by viral and bacterial infections. After querying images that were caused by COVID-19 we had 58 images. We scrapped more recent image data from radiopaedia.org and sirm.org - an Italian COVID-19 database. The images were a mixture of Posterior to Anterior view as well as lateral views. Each image was obtained from a

different patient hence there were no potential issues of data leakage from the training to test dataset. 38 images were found in radiopaedia and 34 in sirm.org. The total number of COVID-19 affected lung images was 130. A common phenomenon with medical dataset images due to the prevalence of healthy patients compared to unhealthy, is class imbalance - despite more web-scraping, the number of healthy images was still higher than COVID-19 affected images. We made sure that this imbalance held in the training as well as testing data set. The number of images of healthy lungs was 1576. 1342 images were allocated to the training dataset and 234 to the test dataset. The main aim of the paper was to compare different state of the art CNN algorithms to find out which performs best in classifying healthy lungs from COVID-19 affected lungs. We explored different architectures such as VGG and ResNet - each architecture with different depths and also analyzed how each architecture performed during different epochs in terms of error rate.

While the project focuses on the Chest-X-Ray dataset, since deep-learning algorithms require to be trained on a voluminous dataset, we utilized the fastAI API which trains its models on big datasets such as ImageNet. ImageNet is a dataset that has over 14 million annotated images belonging to over 21 thousand classes. The vastness of it enables us a model trained on this dataset to be successfully implemented in image processing without overfitting even for small datasets - which is the case in most medical image datasets.

Training Process

Using the fastai API made training very simple to do (code length wise, maybe not debugging wise), since all we have to do is have the images sorted in an imagenet-style format (folders denoting train/valid/test, with subfolders for each class).

The fastAI API allows using pre-trained models with reference to the ImageNet dataset. The VGG model uses mini-batch gradient descent and momentum for training. To prevent overfitting, weight decay and drop-out regularization are used for the fully-connected layers in the end. Weight initialization is an important potential issue with deep-learning networks. In VGG, weight initialization is done randomly in shallower networks and these weights are transferred to the first few and the final layers. The middle layers of the deeper networks are separately initialized. VGG utilizes Glorot initialization for weights which helps set small values of weight centered around a mean of 0. All biases in VGG are set to 0. This helps in curbing the problem of vanishing / exploding gradients during back-propagation. Stochastic Gradient Descent(SGD) has been used for back-propagation. Similar to VGG, ResNet is trained using the ImageNet dataset as well.

A problem we encountered was trying to get the file system working through Kaggle, since most fastai tutorials were done in an Ubuntu environment. However, we were able to reuse code from a different Kaggle notebook to get the images set up in the correct folder format, and we inserted some other images we found into the right folders. Then we have them loaded into the databunch format and we subject them to data augmentation by either scaling, flipping horizontally, or stretching the image so that the models don't become overtrained and just learn to recognize the COVID images (we have a relatively unbalanced dataset). After all the databunches were set up, we simply use the cnn_learner() function to create a pretrained model of our choosing, and we add error_rate (1-accuracy) as a metric (something we can observe during training). Then, we call fit_one_cycle(4), which uses the 1 cycle policy, changing the learning rate over time over 4 epochs. This function only trains the fully connected layers at the top, as the other layers are frozen to take advantage of transfer learning, since the model already comes pretrained with the imagenet dataset. We chose 4 epochs because any more and it would lead to overfitting. We then use the lr_find() method to try and find the optimal learning rate by looking at where the loss still trends downward with a learning rate that's still high. We then use that learning rate (by setting the max_lr parameter) and run another 4 epochs with all the layers unfrozen. We then save that model with the save() function. Afterwards, we can look and choose to remove images from the dataset with the highest loss rates, but judging by the pictures shown through multiple run throughs, we have not seen anything to remove based on the image quality alone (none of the images are blurry or have artifacts). We can then plot the confusion matrix and see how well our model does with classification. From observation of multiple run throughs, our model does a fairly good job of classifying the images, with only 4-6 images incorrectly classified out of ~260. What's nice about fastai is that it lets us see which images have the highest loss, what the prediction was, and what the actual value was.

Training and validation loss at each epoch:

ResNet34

epoch	train_loss	valid_loss	error_rate	time
0	0.148852	0.218164	0.078358	01:37
1	0.139776	0.135725	0.044776	01:39
2	0.115702	0.184929	0.048507	01:39
3	0.097334	0.102623	0.026119	01:38

ResNet101

epoch	train_loss	valid_loss	error_rate	time
0	0.125367	0.085285	0.033582	01:41
1	0.116810	0.058566	0.011194	01:42
2	0.093631	0.048649	0.007463	01:41
3	0.072298	0.049215	0.011194	01:42

VGG 16

epoch	train_loss	valid_loss	error_rate	time
0	0.120511	1.009778	0.246269	01:41
1	0.121524	0.650027	0.089552	01:42
2	0.094260	0.024206	0.003731	01:41
3	0.065304	0.032699	0.007463	01:43

VGG 19

epoch	train_loss	valid_loss	error_rate	time
0	0.157778	0.255269	0.093284	01:42
1	0.100808	0.120613	0.037313	01:42
2	0.086315	0.069996	0.018657	01:41
3	0.059999	0.101871	0.018657	01:42

Validation Error Metrics after the 4th epoch for each model:

Model	Error Rate	Accuracy	Sensitivity	Specificity
ResNet34	0.02612	0.97388	0.97059	0.97436
ResNet101	0.01119	0.98881	0.94118	0.99573
VGG 16	0.00746	0.99254	0.94118	1.00000
VGG 19	0.01866	0.98134	0.88235	0.99573

Conclusion:

After running all these models through multiple reruns (whether because we needed to update part of the code or because Kaggle crashed on us), we have noticed that the ending confusion matrix varied by quite a bit with its results. While every model scored well above 95% in terms of accuracy, it is hard to decide which model came out on top. Each model took around the same time to run with GPU acceleration enabled (ResNet34 took 2 seconds less per epoch), and achieved roughly the same metrics. This could be attributed to the high variability of these models, with hundreds of millions of parameters to train. This could be clearly seen through the learning rate finder graph, which varied greatly between runs, so it was hard to even find the right learning rate for each model. Also, the imbalanced dataset also contributed to the variability of metrics, since these models are used to training on thousands of images, while we only had 130 images of COVID positive x-rays after manually scouring medical websites for additional images. Perhaps with more time and resources, we would have been able to train more accurate models, but given the circumstances, we are very impressed with our results. Accuracy above 97% for every model is very impressive, especially given that this was done by two college undergraduate students with limited access to private medical information. With the proper tools, machine learning could become a very powerful tool for patient diagnostics in the very near future, allowing for online doctor's visits that can give accurate diagnoses through simple image uploads. One can hope that the powers of convolutional neural networks can be used to change the world for the better.

View code here:
https://www.kaggle.com/derekjiang/coronahack-eda-and-baseline-model

References:

1. https://towardsdatascience.com/fastai-image-classification-32d626da20
2. https://www.kaggle.com/praveengovi/coronahack-eda-and-baseline-model-final
3. https://www.kaggle.com/hortonhearsafoo/fast-ai-lesson-1
4. Very Deep Convolutional Networks for Large Scale Imaging
5. Deep Residual Learning for Image Recognition