Machine Learning Engineer Nanodegree

Capstone Project

Oliver Dixon
October 1st, 2020

I. Definition

Project Overview

This capstone project made use of a convolutional neural network (CNN) in order to detect whether a sample of chest X-ray images show the presence of pneumonia or are normal. The dataset is freely available on Kaggle

(https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia).

CNNs are the most popular choice when it comes to the field of computer vision. A CNN was constructed to extract the features necessary to determine if someone has pneumonia. Interpretation challenges are common when it comes to medical images and this model could help solve much of this problem. Usually, large datasets are needed to train CNNs and this is a common problem with medical images which is why data augmentation algorithms were used to improve validation accuracy. The validation accuracy of this model could be further improved but it achieved production level results as is.

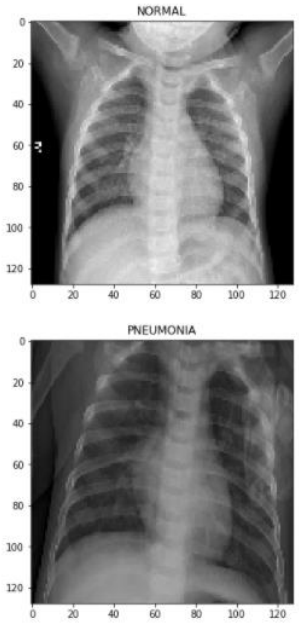
Problem Statement

Pneumonia is common worldwide and it is estimated that over 150million people get infected with it each year. Pneumonia can be deadly for some, especially in developing nations. Many developing nations have a severe shortage of doctors, nurses and medical facilities which means the poorest people have the least access to medical services. These vulnerable

communities need something that can provide a fast, cheap, reliable diagnosis along with the required medicine.

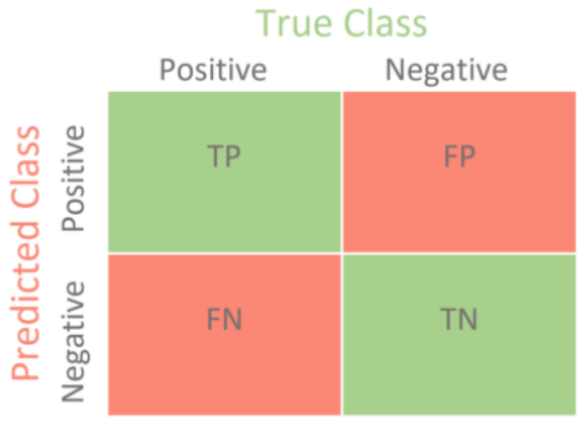
Artificial neural networks (ANN) are based on biological neural networks in the brain. A neural network is trained by adjusting neuron input weights based on the network's performance on example inputs. If it classifies an image correctly, the weights contributing to that classification are increased, while other weights are decrease. This seems simple but what makes it complex is that they are organized in a vast network of billions in the brain and are organized in consecutive layers. Likewise, an ANN can be constructed from a few simple neurons to a complicated network consisting of millions of neurons. This is why neural networks are preferred when it comes to images that are broken down into thousands of parameters.

CNNs are the most common of the deep learning algorithms for medical image classification. The algorithms and reinforcement learning used to determined the best hyperparameters during training are computationally very expensive which is why I designed a simple yet effective CNN to classify the pneumonia problem as shown in the images below.



Metrics

Four metrics will be used in order to see how well the model is performing, accuracy, precision, recall and F1 score. In the case of a binary classification the results can be looked at according to the confusion matrix in the figure below.



Confusion Matrix for Binary Classification

True Positives (TP) – These are the correctly predicted positive class. In this case, this will be the number of images that are correctly predicted to have pneumonia.

True Negatives (TN) – These are the correctly predicted negative class images. This will be the number of images predicted not to have pneumonia, i.e. normal chest x-rays.

False Positives (FP) – This is when the image was predicted to be positive (have pneumonia) but in actual fact, it was negative (normal).

False Negative (FN) – This is when the image was predicted to be negative (normal) but it was in fact positive (pneumonia).

Accuracy is calculated by taking the correct observations and dividing it by the total number of observations.

Accuracy = (TP + TN)/(TP + FP + FN + TN)

Precision is the correctly predicted positive cases over the total predicted positive observations. Those with actual pneumonia divided by those correctly predicted to have pneumonia and those falsely predicted to have pneumonia.

Precision = TP / (TP + FP)

Recall is the correctly predicted observations over all the observation in the true, actual class. Of all those who really have pneumonia, how many were labeled correctly.

Recall = TP / (TP + FN)

F1 Score is the weighted balance of Precision and Recall. This is useful in uneven class distribution.

F1 Score = 2*(Recall * Precision)/(Recall + Precision)

There will be a focus on the validation set accuracy to see whether overfitting is occurring.

II. Analysis

Data Exploration

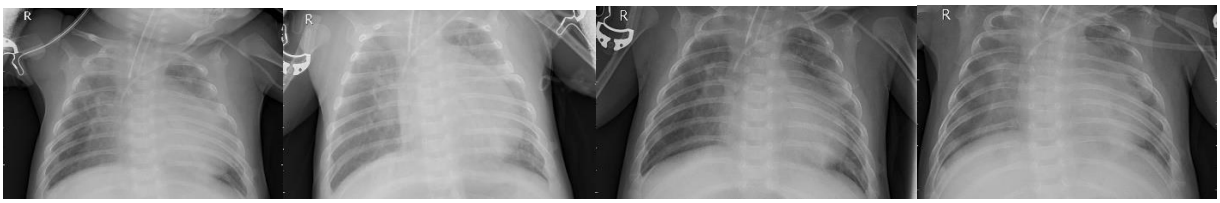
The data set consists of three folders: train, val and test. Each folder contains two sub-folders, one labeled NORMAL and the other labeled PNEUMONIA. Images were distributed as follows:

Label	Class	Train	Validation	Test	Total
0	NORMAL	1341	8	234	1583
1	PNEUMONIA	3875	8	390	4273

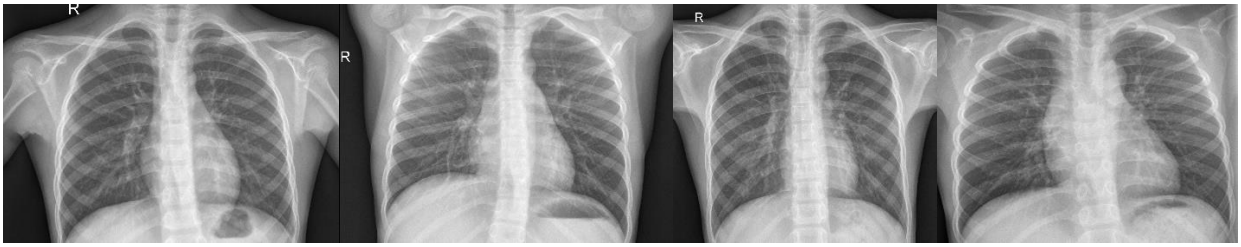
The data shows that there is a class imbalance between the two classes and this is especially important in the training set because that data will be used to train the model. As it stands the model will almost certainly be biased towards PNEUMONIA since there are three times the number of PNEUMONIA images than NORMAL images. This means that I would need to balance the classes before training the model.

The validation data set is very small compared to the training set and this might affect the validation accuracy I will be looking at to see if the model is performing without overfitting. I will need to create a new validation set will a larger sample of images. In order to visualize the data images had to be obtained and resized and labels had to be extracted.

X-ray images with pneumonia below:

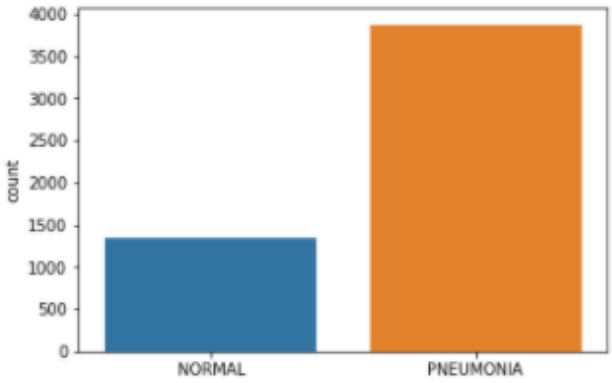


Normal chest x-ray images below:

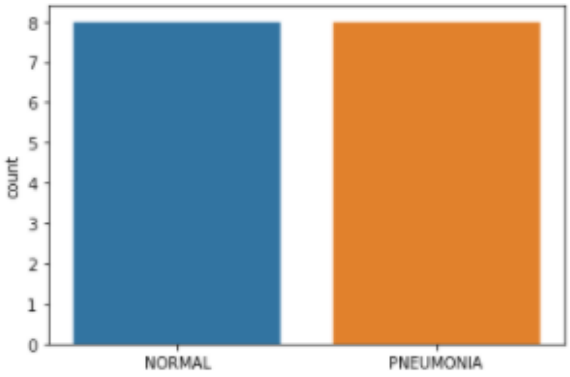


Exploratory Visualization

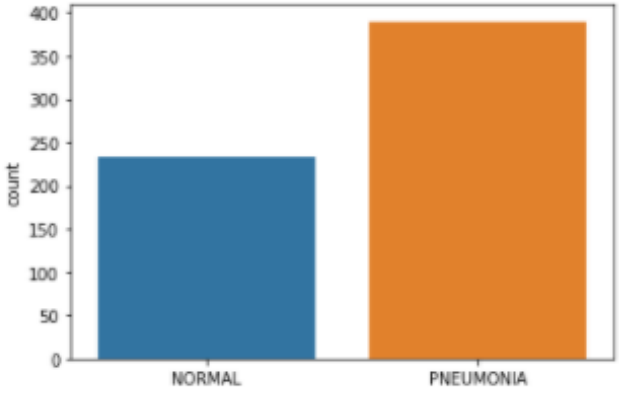
Training Images Class Distribution



Validation Image Class Distribution



Test Image Class Distribution



The visualization of the data clearly shows the class imbalance and the small sample size of the validation set that will need to be corrected.

Algorithms and Techniques

A CNN model, transfer learning and image augmentation will be used and implemented in this project.

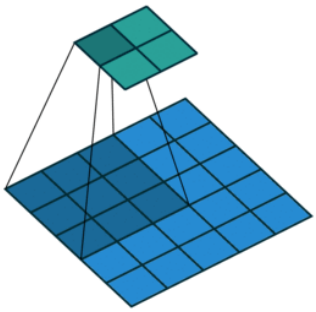
As mentioned, CNN have become the preferred choice for machine learning engineers in classifying images and is continuing to be developed in the case of self-driving cars, web search using Google Lens, medical imagery and even psychology by classifying facial expressions and body language, just to name a few.

Logistic Regression can be likened to a single perceptron (neuron). In Artificial Neural Networks (ANN) there is a group of multiple perceptrons/neurons at each layer. These layers usually consist of 3 layers, Input, Hidden and Output. Each layer is trying to learn weights that will have a certain outcome, typically a classification. CNNs works on the same premise however, what sets CNNs apart are filters, i.e. kernels. These kernel extract features from the input using the convolutional operation. Images are used as input data and the imaged is then convolved with filters resulting in a feature map.

Output of Convolution

https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

A single filter is applied across different parts of an input to produce a feature map. The GIF below demonstrates how the 2x2 feature map is produced by sliding the same 3x3 filter across different parts of an image.



https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Each layer in a CNN has two kinds of parameters, weights and biases. The total number of parameters is the sum of all the weights and biases.

The success of CNNs depend heavily on choosing the correct hyperparameters.

- The **learning rate** controls how much to update the weight in the optimization algorithm. How we use the learning rate, whether constant or varying depends on the optimizer in use, SGD, Adam, Adagrad etc.
- The number of **epochs** is the number of times the entire training set pass through the neural network.
- The **activation function** is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not.
- The **batch size** is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

Transfer learning is when a pre-trained model is saved so that it can be used on new data. These pre-trained models are typically trained on large datasets for large scale classification. This model is then used as it or customized to fit the desired task. In this case, our dataset is small with only

a few thousand images compared to hundreds of thousands of images or even millions of images used to train these pre-trained models used for transfer learning. Using a pre-trained model will allow me to take advantage of a lot of computing power it took to train the model while being able to add to the model if I need to in order to achieve my desired outcome. I will be making use of the pre-trained models available through Keras in Tensorflow.

Image augmentation transforms the image in different ways by changing the angle, flipping the image, rotating, stretching etc. so that the model can recognize different versions of the image. The X-ray images are not all zoomed equally or the angles are slightly off. If the model is trained without augmentation then it could overfit when the model is tested against the validation data. Image augmentation might not be as necessary if the dataset has millions of images where there are naturally occurring inconsistencies but in this case image augmentation is needed because of the small dataset and the relative uniformity of the images.

Benchmark

One of the top performing notebooks in Kaggle for this dataset was posted two years ago and remains one of the top notebooks. (https://www.kaggle.com/madz2000/pneumonia-detection-using-cnn-92-6-accuracy) He was able to achieve training accuracy of 0.9682 and validation accuracy of 0.8750.

I will look at his results to see how I compare with her as a benchmark.

III. Methodology

Data Preprocessing

Several data augmentation methods were employed in order to enlarge the size and quality of the dataset. This is helpful in order to avoid overfitting. The table below shows the settings used in image augmentation.

Method	Setting
Rescale	1/255
Rotation range	40
Width shift	0.2
Height shift	0.2
Shear Range	0.2
Zoom Range	0.2
Horizontal Flip	True

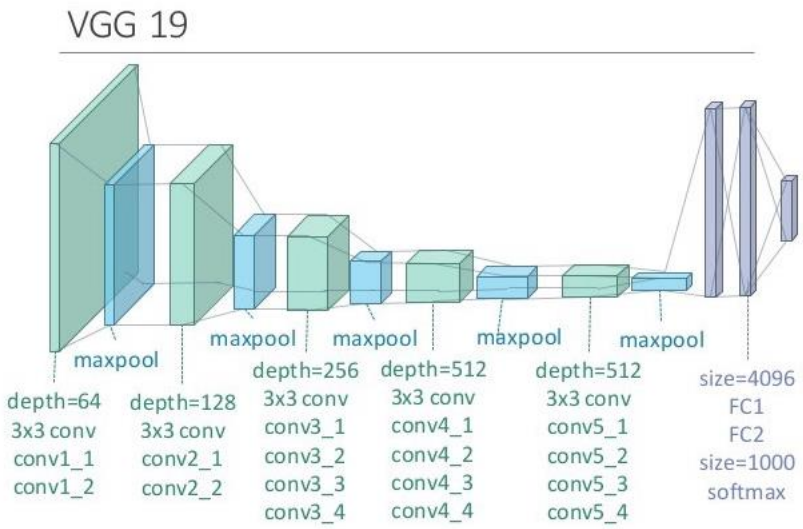
Rescale represents image reduction. The rotation is the range in which images were rotated during training. Width shift is the horizontal shift, and height shift is the vertical shift during training. Shear range clips the image angles in an anti-clockwise direction, the zoom range zooms the images and the lastly, the images were horizontally flipped. I also use the data augmentation function to split the training set into a bigger validation set (25%) in order to get better validation results.

Implementation

I had to make multiple attempts to get my CNN model to work correctly. Originally, I used a pre-trained model, MobileNet_v2 with an additional dropout layer of (0.5) and Dense layer with sigmoid activation for binary classification. The problem I encountered was that my validation accuracy remained the same after each epoch. All the other metrics would improve except validation accuracy. I looked at my code again to see if my classes were balanced and I applied sklearn's class_weight to balance the classes. Still, no improvement in my validation accuracy. Next I constructed my own CNN from scratch, without using a pre-trained model, and again, no success. I then played around with the hyperparameters and using different

Tensorflow, pre-trained models. What ended up working was the VGG19 pre-trained model, changing the optimizer from “adam” to “SGD” and lowering the learning rate quite significantly. It is very likely that the optimizer found a local minimum for the loss which is why I got stuck with the same validation accuracy. However, changing the optimizer and learning rate and using a different pre-trained model gave me good results.

Below is a visualization of the VGG19 architecture up the last maxpool layer. After the maxpool layer in this image I flattened the output and added three more hidden layers (512, 128, and 64) with dropouts with a final dense layers with a sigmoid activation.



Refinement

Initially, when using MobileNet_v2, my optimizer was set to “adam” and my learning rate was 0.001, batch size was 32 and attempted multiple trainings at different epochs all yielding the same validation score of 0.7350. I also found that the data is not shuffled which could also add a bias to the training but this also had no affect on my validation accuracy. After trying other pre-trained models, starting a CNN from scratch, changing the optimizer and lowering the learning rate I was finally able to get improvement in my validation score. Using VGG19 and adding 3 hidden layers with dropouts, using the “SGD” optimizer and setting the learning

rate to 0.0000001 the validation accuracy started with 0.7559 and by the third epoch it was up to 0.8780. The training accuracy was 0.8256 so the model was underfitting. Next I generated a training model again but this time I used 20 epochs and ended up with the following results.

	Training Data	Validation Data
Accuracy	0.9000	0.8926
Recall	0.9013	0.9018
Precision	0.9195	0.9199
F1 Score	0.9103	0.9108

I also test Xception and EfficientNetB7 but both of these could not match VGG19's performance. Validation accuracy in both Xception and Efficient Net were in the 85% range, well below the 90% performed in VGG19.

VGG's results are good because the metrics in the training and validation data are very close to each other which means these are reliable results. The accuracy, recall, precision and F1 score are all around the 90% mark which is good enough for being employed to production. Pneumonia is contagious which means that a false negative could be costly so ideally in the case of contagious illnesses we would like to get the recall score closer to 100% but in the case of employing this technology in developing country even with the risks involved the benefit would outweigh the costs.

The metric could possibly be improved with hyperparameter tuning and trying different pre-trained models and experimenting with the hidden layers. This is most likely where most of the improvement would come from other than upscaling and/or using a larger dataset.

IV. Results

Model Evaluation and Validation

The final model included the pre-trained VGG19 model in Tensorflow with three added dense layers (512, 128, 64) with the “relu” activation, as well as three batch normalization layers and three dropout layers to avoid overfitting (0.7, 0.5, 0.3). The model was most sensitive to the type of optimizer used and the learning rate. As mentioned, my validation accuracy remained the same when using the “adam” optimizer and larger learning rates (0.1, 0.01, 0.001, 0.0001). The model is sensitive to the input data before it will start training the model so if the input data was not correct the model wouldn’t even start to train. The fact that the model can start training is already a good sign in CNNs. The training data metrics and validation data metrics were remarkably close together which means this is a good indication of how the model would perform in production. A few images from the testing data were used with the predictor and it made the correct predictions each time.

n this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model’s solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

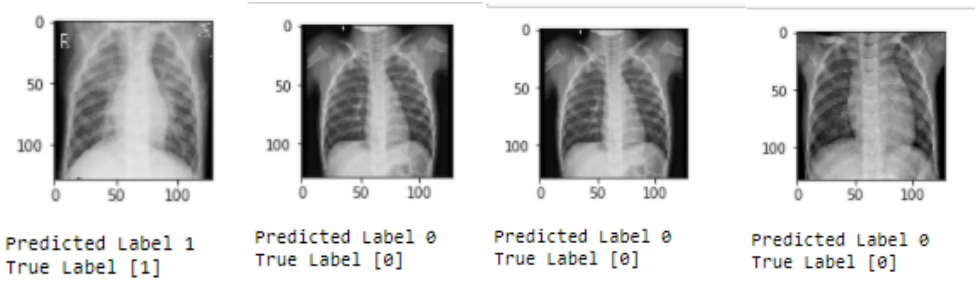
- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

I was able to able to achieve the benchmark's training accuracy but I did achieve a higher validation accuracy. My accuracies were within less than 1% of each other while the benchmark's accuracies were 9% apart from each other which indicates overfitting on benchmark's side. I would argue that my model would be more reliable in production because there is not as much overfitting and would therefore have solved the problem to good enough degree that it would product a lot more benefit than harm. Any xray image could be tested by this model and it would give the correct prediction 9 times out of ten.

V. Conclusion

Free-Form Visualization



The following images show the results of prediction on the test data and the model should correctly predict nine times out of ten. Of these four images the model correctly predicted all of them. One can see that it is very difficult for the untrained eye to see which of these x-rays have the presence of pneumonia and how well the model can classify.

Reflection

I have developed a model that takes X-ray images from the frontal view and then detects and classifies pneumonia. Data coming is transformed into smaller sizes and then also augmented because of the small size of the

dataset to the CNN extract features from different angles and sizes after which it is fed into a CNN. Next, the CNN extracts features from the images and then classifies them. CNNs are commonly used for image classification because they are able to be developed into a complex networks and handle a lot of data. Initially I faced the problem getting the same validation accuracy across many different epochs so I tried different pre-trained models, different optimizers and lower learning rate which then yielded a good improving validation accuracy. I tried a few different pre-trained models but got the best results with VGG19. Predictions we made using the testing data and all the predictions were correct. Models like these can mean that poor, developing countries can get diagnosed very quickly and then get the available treatment without having to pay large sums of money or wait for hours or days. Improvements are possible if larger datasets can be collected and used for training.

Improvement

As mentioned, larger datasets can be used but as far constructing a CNN it is possible that a custom build CNN might perform better but this will need a lot of computing power and time in order to test a lot. Hyperparameter tuning is almost certainly improve the performance but this again will need a lot of computational power. I used my training set in order to create a larger validation set but I think if I combined all the data first I would have a larger training set to train from which should yield better results.

References:

- https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
- https://www.kaggle.com/madz2000/pneumonia-detection-using-cnn-92-6-accuracy