An application of Deep Learning for Computer Vision

Saran Mishra

May 2020

1 Introduction

This analysis is an application of deep learning and an introduction to computer vision problems. Specifically, I target the utilization of these techniques to medical imaging and was inspired research previously conducted on this topic[2]. The goal of this analysis is to use X-ray scans of chest and lungs from the Guangzhou Women and Children's Medical Center in China, and be able to predict with the most accuracy, cases of Pneumonia. Diagnosing Pneumonia quickly and efficiently is an issue in the medical community especially during a global pandemic like Sars-Cov-19. [3]. Medical imaging is experiencing a growth of artificial intelligence (AI) driven applications which are scale-able and seek to eliminate error rates as well as perform faster than human practitioners.

1.1 Table of Contents

2 Image Data and problem set up

Image data is quite different than non-image data, as one would expect. This data set, which was found on the popular data science platform Kaggle, breaks down to 5863 jpeg files, each containing a photo of an X-Ray of the lungs, as shown below. [**?**] Furthermore, the organizers of the data set have pre-establishes training, testing and validation sets. These sets were not altered for this analysis – a future analysis should randomize the data and expand the data

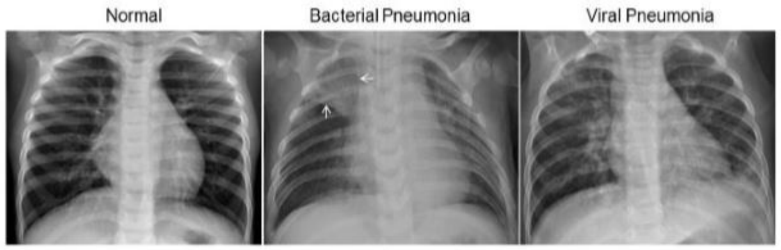


Figure 1: Example of Lungs Data

set.

I will now attempt to introduce the structure of image data in the simplest and most succinct of ways. Image data comes in a lot of forms but the most common version someone runs into is RGB images or a truecolor image [5]. We can think of each image having cells or pixels based on the size of the image. So a 12 by 12 photo would have 144 cells or pixels, for example. So, mathematically we interpret RGB photos as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel and the color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location [5]. Or in other words, every pixel holds some numerical value (the combination of Reds and Greens and Blues), within a range of 0 to 255.

In our case, we are given 5863 images. I seek to resize them into a standard 64 by 64 i.e. each image is composed of 4096 pixels. I then sought to pre-process the images further utilizing the Keras library [1]. In this portion, I was given the option to not just resize or normalize the images but also manipulate them to increase the amount of data the network would receive. So initially we seek to normalize by dividing each image by 255. But I then allowed the images to be flipped horizontally, I allowed for shearing i.e. the counter clockwise rotation of an image, and finally, I gave the program a chance to zoom into the image [10]. There was no "optimized" way of doing this portion of this analysis – the goal was simply to increase the amount of parameters that would be passed through the model while also maintaining a degree of simplicity. Another extension of this research, definitely, should try to optimize all parameters. More on that in our parameter tuning section.

3 The Image Classification problem

We know how image data is set up now, and we begin to focus on the problem. Image classification has been around for a long time. Specifically, computer vision in practice is divided into object recognition or object classification. For our purposes, we will be focusing on the latter.

If we look at one of the photos, our human eye sees the shoulder bones, the rib cage, the heart, the spine and a set of lungs, sometimes with some shading in certain areas. In our minds, we are able to identify these features with just the X-Ray even though we are not looking these same features directly in front of us. That identification is the motivation for image classification.

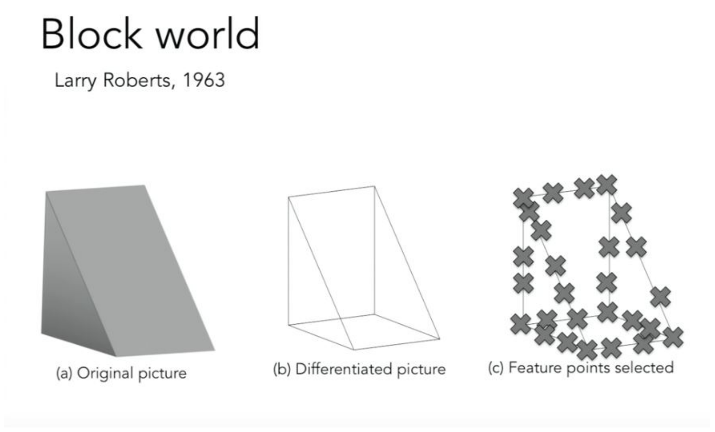


Figure 2: Larry Roberts' Blocks [4]

It was in 1964, that Larry Roberts, during his PhD dissertation tries to first "solve" this problem of identifying features in images. At first Roberts simplifies the visual world into geometric shapes. And furthermore, his goal was to be able to identify these shapes and reconstruct them based on some set of features extracted from these shapes (see graphic above). Researchers continued development in this area and asked more difficult questions. Most machine-learning models tended to over fit in the training set and did poorly when measured for accuracy. This is because visual data is particularly complex and models for image classification require large inputs and large parameter sizes. And thus researchers from Stanford University and Princeton University started image-net.org, as was discussed in class which includes "all the images one could find on objects" [4].

In this particular analysis, which is rather small, parameters for ConvNet models were in the hundreds of thousands and some times in millions (the highest being 3,937,409). In comparison to larger studies, this is a tiny.

We will move on to explore some methods of particular interest for this analysis such as the SIFT (Scale-invariant feature transform) and SURF (Speeded up robust features). The SIFT algorithm was patented for a long time but the patents have expired, so utilizing the functions in the OpenCV library - which is heavily utlized in this analysis - we are able to study some basic functionalities of the algorithms (SURF is also available). If we look at these from a historical standpoint, we notice that SIFT based algorithms were heavily utilized prior to the expanded usage of Convolutional Neural Networks or ConvNets (ConvNets) (see: figure 10).

The SIFT algorithm analyzes a new image by individually comparing each feature from the new image to a database created from past images and finds a "candidate matching features based on Euclidean distance of their feature vectors" [7] This really got to the crux of the feature detection problem and was as SIFT detectors, extract local detectors from the regions around the interest points or dense patches [11]. The SURF algorithm is a variant of the SIFT algorithm [8].

$$S(x,y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i,j) \tag{1}$$

SURF (Refer to Equation 1) is a faster algorithm and its methodology for feature detection is also different. SURF utilizes square-shaped filters as an approximation of Gaussian smoothing or blurring – a bit of this was touched upon in the very beginnings of this course. Specifically, this is the "I" in the equation above. SURF relies on the integral image or Summed Area Table Approach [9] as it generates the sum of values in a rectangular subset. Then the algorithm will take a square filter to the rectangular subset

SIFT on the other hand utilizes cascaded filters to detect scale-invariant characteristic points, where the difference of Gaussians is calculated on re-scaled images progressively. [8] The idea of the square shaped filters are vital and will return in the next section. These algorithms were then run on images which are labeled to have Pneumonia as an exploratory measure. We can see what exact features these algorithms are detecting. Both can be utilized further in photo comparisons – this was not done in this analysis as it is not the major focus.

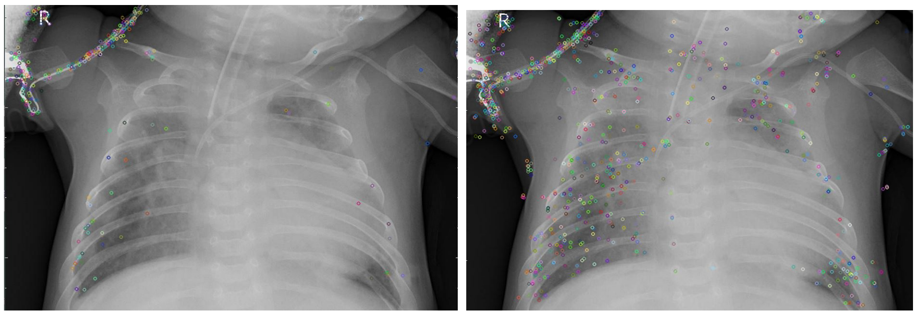


Figure 3: SIFT (Left) and SURF (Right) on positive Pneumonia case

4 Convolutional Neural Network

This section will introduce the theoretical framework of a Convolutional Neural Network (ConvNets) i.e. the focus of this paper before the next section will dive more into the application of the ConvNets utilized for this project.

Generally, the ConvNet follows the infrastructure of Artificial Neural Networks with slight but important variations meaning that there is some sort of an input layer then hidden layers and an output layer. A typical ConvNet has hidden convolutional layers, pooling layers and finally a layer to flatten prior to its classification. In practice we often see a many variations of the convolutional and pooling layers (See Figure 4 for details).

Convolutional Layers:
The convolutional Layers serve as the key to deciphering the incredibly large amount of data that images provide – and this is really the key, here. The ability to utilize all parameters and features of an image helps with not losing any information from the input. Specifically, convolutional layers utilize filters, similar to the SURF algorithm, which then tries to find a match between the filter and the feature in an image. (See figure 5 in appendix). The average of the results for a particular filter versus actual image feature are then recorded in a separate matrix.

Pooling Layers:
The stack of filtered images now goes into the pooling layer. Mathematically, this is quite straightforward (see Figure 7). We look at our stacked images and since we want the maximum value, using our same filter (3x3), we pick the largest value of the stacked image matrix and output the value into a third matrix. Why do we want the maximum value? Intuitively, let's think about what has been done so far. We first find the matches from our filtering thus identifying the significance of the feature at that location. Now, from that location, given the filter size, we pick the strongest feature i.e. the most matched.

Flattening Layer:
The flattening layer of the ConvNet serves the purpose of decision making i.e. voting, to see if, for example, a new image introduced to the network fits either in class A or B. We can see how (see figure 8 and 9) the weights which have been acquired from the previous layers ends up being the decision making tool for classification. In this layer, in practice, the activation function utilized is a sigmoid function, as this is a binary classification problem, at the end of the day.

5 Model Comparison and results

Given the fact that this is a classification problem, a host of "standard" classification tools were tested against the ConvNet algorithms. One of the most interesting and tricky portions of this analysis was the data manipulation of how to convert image data, which generally comes in 3-dimensions (RGB) as previously discussed, into a 2-dimensional vector which does not lose any of the information during the conversion process. The change in dimension does not lead to data loss as the third dimension has been translated between two dimensions.

Once the data was formatted into a 2-dimensional vector i.e. a vector of all the covariates of the images, any sort of classification mechanism could be utilized. So we will discuss the classification mechanisms in the order of complexity, ending with the ConvNet and its variations.

An interesting thing to note as we have been discussing dimensionality throughout this paper. The image data was originally in 3 dimensions i.e. the RGB but for the non-deep learning models run, it was converted to 2 dimensions i.e. grayscale so the non-ConvNet classifiers could handle the data.

5.1 Logistic Regression

The logistic regression classifier needs to be the first classifier discussed due to its simplicity. Also because elements of this classifier are utilized later on in our ConvNets. It also makes sense, intuitively, as our task is to build a binary classifier. As we know, mathematically, the motivation is as follows: instead of just classifying, the logistic regression classifier helps us understand the probability of successful classification. This quite important as the key measurement in this analysis is the accuracy percentage.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{2}$$

What the equation above can tell us, as it will later be utilized in the ConvNet, is how the sigmoid function classifies. We can note that the value of this function will always be positive but will be less than 1. Since we are using a rather baseline version (no tuning) in this portion of the analysis, the algorithm is quite simple. First, the data is converted to 2-dimensions, then we utilize the training set to predict the test set of the data. Here, we get a the accuracy of logistic regression classifier on test set to be 0.62

The implications are stark: a simple logistic regression classifier cannot seem to classify our image data with a good degree of accuracy. Logistic Regressions are related to Neural Networks and I like to think of them as a simple unilayered network.

But, as expected, an overly simplistic version fails to measure up to a hierarchical network structure such as a ConvNet. (Refer to classification matrices in appendix for more details on performance)

5.2 Support Vector Machines – Linear

The general framework of a linear Support Vector Machine (SVM) was utilized as another comparison for this analysis. SVMs by nature, seek to classify by finding a dividing line which divides the two sets of data or a line which maximizes the separation margin between the two sets.This analysis will not go into the deepest elements of the SVM classifier as it is not the focus.The SVM utilized in this analysis is not tuned in any capacity.

The SVM Model recorded an accuracy of 0.75. So we can start to see that the non-ConvNet classifiers are performing as expected so far. (Refer to classification matrices in appendix for more details on performance)

5.3 K-Nearest Neighbor

The final classification architecture utilized for comparison is the K-Nearest Neighbor (KNN) Classifier. KNNs utilize euclidean distances to classify. Prior to any sort of tuning, or in this case finding the ideal value for k, we started with a KNN where the k value was 2.

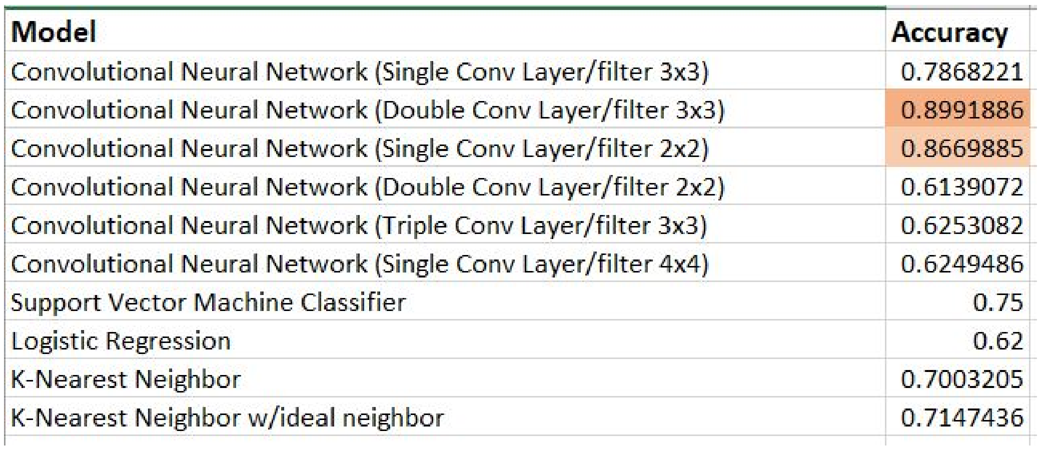
With k = 2, the accuracy found here was: 0.70032. So we have certainly improved our accuracy score versus the logistic regression classifier. This model was later tuned to find the ideal k value. (Refer to classification matrices in appendix for more details on performance)

5.4 Convolutional Neural Network

Finally, we come to the deep learning technique which is the central point of this analysis. In this section I will be discussing the parameters which I utilized for the ConvNet models.

We know the general structure of a ConvNet so this portion will follow that framework. We start with the first illustration of the ConvNet and select just one convolutional layer, followed by a single pooling layer and lastly the flattening layer.

Initially feature size, filter size were picked arbitrarily. The amount of features chosen were 32, the batch size was also 32, epochs were set to 10 and these manual sets stayed as a constant throughout. The Filter Size started with a 3 x3 filter and this was tuned further as we will see in the next section. The activation function for the convolutional layers was a ReLu (Rectified Linear Unit Activation function). Why was this the case? Well, if we take a look at figure 10 in the appendix, we can see that when we gather our stacked matrices from the convolutional layers, we often receive values less than 1. Those values convert to 0, and as is the nature of the ReLu function, only positive values are kept. This helps differentiate from vital features and non important features. Finally, the activation function on the flattening layer of the ConvNets were always sigmoid functions as this is a binary classification problem.



Model	Accuracy
Convolutional Neural Network (Single Conv Layer/filter 3x3)	0.7868221
Convolutional Neural Network (Double Conv Layer/filter 3x3)	0.8991886
Convolutional Neural Network (Single Conv Layer/filter 2x2)	0.8669885
Convolutional Neural Network (Double Conv Layer/filter 2x2)	0.6139072
Convolutional Neural Network (Triple Conv Layer/filter 3x3)	0.6253082
Convolutional Neural Network (Single Conv Layer/filter 4x4)	0.6249486
Support Vector Machine Classifier	0.75
Logistic Regression	0.62
K-Nearest Neighbor	0.7003205
K-Nearest Neighbor w/ideal neighbor	0.7147436

Figure 4: Results of this analysis

Furthermore, Backpropagation i.e. finding the negative gradient of the cost function, was utilized in every iteration of our Convnets. So if you can imagine the infrastructure of the Convnet, there was a simultaneous backpropagation algorithm running.

It is clear to see that even the simplest ConvNet model performed extremely well. The accuracy rate was 0.7868221 i.e. superior to all other models thus far. A complete table of results can be found below.

6 Parameter Tuning

Following the completion of one ConvNet, I sought to tune, to the best of my ability, the "user" based parameters. Specifically, as you can see in Figure 4, I focused on altering the size of the filter as well as the number of convolutional layers.

The KNN model was further tuned using the validation set to obtain the ideal K.

7 Conclusion

In this analysis, we took not just a historical look at computer vision, but also applied a thorough analysis of a supervised methodology to a deep neural network. We can see the power of the ConvNet algorithm as it performs considerably better than other classifiers of image data. Considering we are nearing 90 percent accuracy when the model is tuned, slightly, we should obviously declare that a "winner" if that was the goal. One thing I learned was the accuracy rates do depend on the number of parameters. However, so does the run time of the algorithm. Each ConvNet iteration took – even at the minimum – north of 25 minutes to run. While I would have loved to optimize all human inputs, given the run time and how taxing this is computationally, it would not have been ideal for the purposes of this analysis. The implications of this result are huge and fall in line with the original research done on this subject [2]. The authors of the Cell paper report that the ConvNet they trained with their human inputs had a 92.8 percent accuracy rate. The research in this area is growing and thus there are many ways to expand upon the materials and methodologies highlighted in this analysis. One obvious one would be to train ConvNet models on various different medical imaging data sets. Another obvious one would be to optimize the tuning of the model i.e. the human inputs. But what I remain exceptionally curious about is utilizing more Projective geometry concepts and expanding the robustness of ConvNets and their capabilities. The domain of computer vision is forever growing as is Deep Learning, so, it is safe to say that these methodologies can be applied to a variety of different fields.

8 Appendix

In this section I will speak a bit about the code attached to the analysis. The project was conducted primarily in python as I had prior experience doing Deep Learning utilizing the Keras wrapper for Google's TensorFlow library. A package heavily utilized is OpenCV i.e. the computer vision library. However, the code is not written in standard OOP format and written loosely like R-Code would be. My code has many instructions within, so a run should not be problematic. The files are broken up as such: there is a file for all of the ConvNets which are run and then a second file for the non Deep Learning Models. Below, you will find a complete glossary of all the figures and images relevant and utilized throughout the written work.
]

References

[1] Keras.io. Image preprocessing.

[2] Daniel Kermany. Identifying medical diagnoses and treatable diseases by image-based deep learning.

[3] R. Levitan. The infection that's silently killing coronavirus patients, 2020.

[4] Fei Fei Li. Convolutional neural networks for visual recognition.

[5] ECE dept. Northwestern University. Image processing toolbox.

[6] B. Rohrer. Convolutional neural networks, 2016.

[7] Wikipedia. Scale-invariant feature transform.

[8] Wikipedia. Speeded up robust features.

[9] Wikipedia. Summed-area table.

[10] Yumi. Learn about imagedatagenerator.

[11] Liang Zheng. Sift meets cnn:a decade survey of instance retrieval.

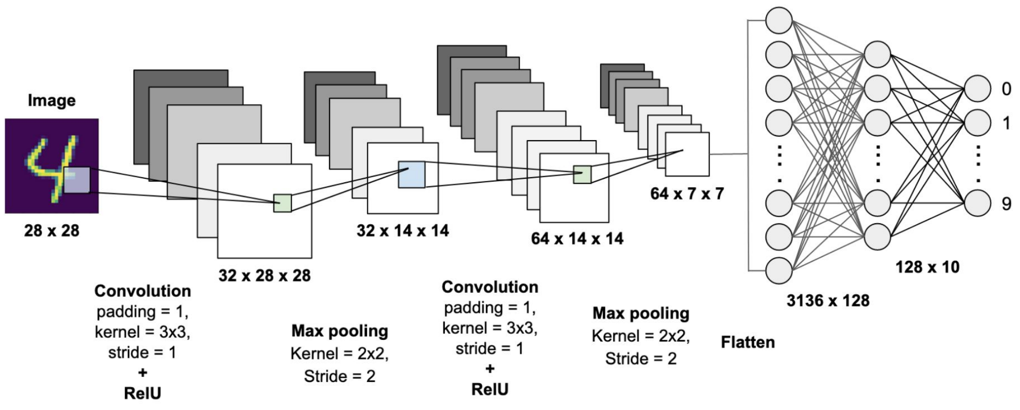
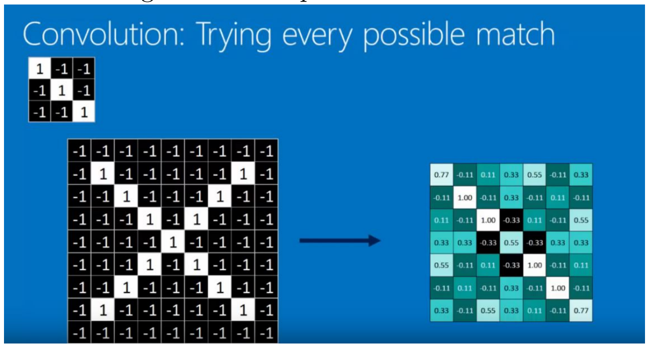


Figure 5: Example of CNN Architecture with MNIST Data



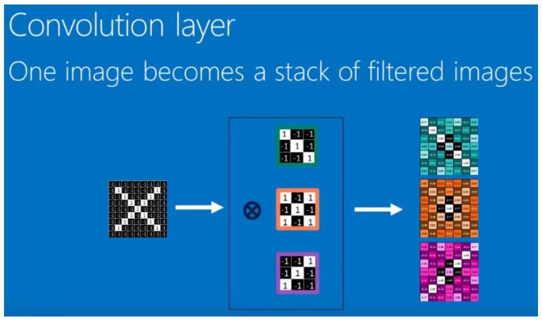
Convolution: Trying every possible match

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Figure 6: Example of Convolutional Layer,[6]



Convolution layer

One image becomes a stack of filtered images

Figure 7: Example of Convolutional Layer,[6]

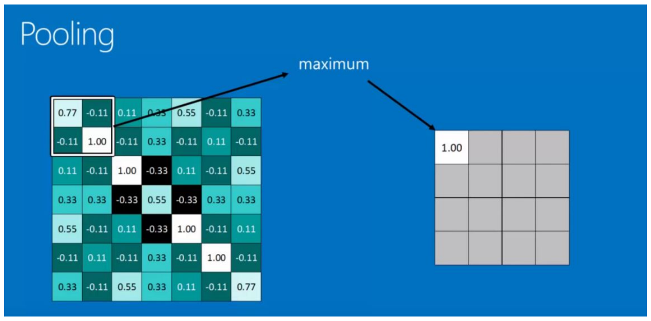


Figure 8: Example of Pooling Layer,[6]

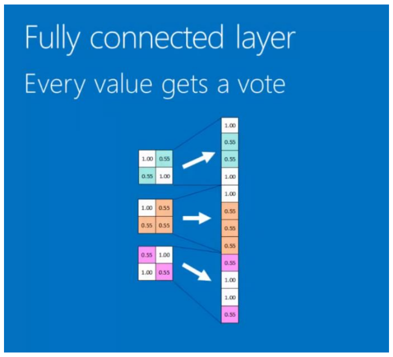


Figure 9: Example of Flattening Layer,[6]

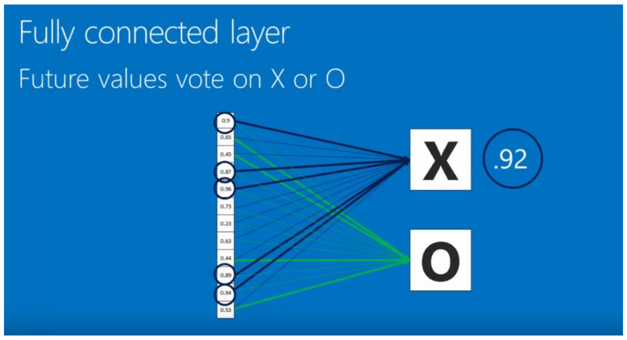
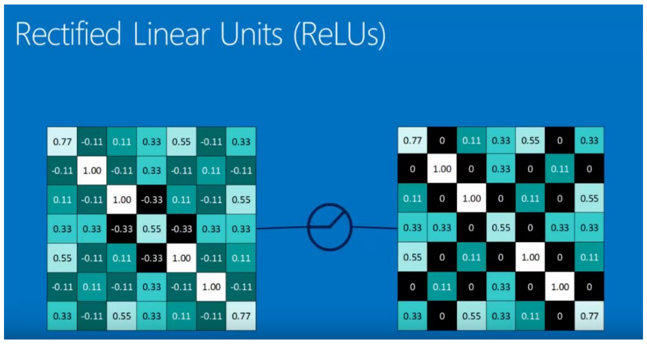


Figure 10: Example of Flattening Layer,[6]



Rectified Linear Units (ReLUs)

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Figure 11: Example of Relu Layer,[6]

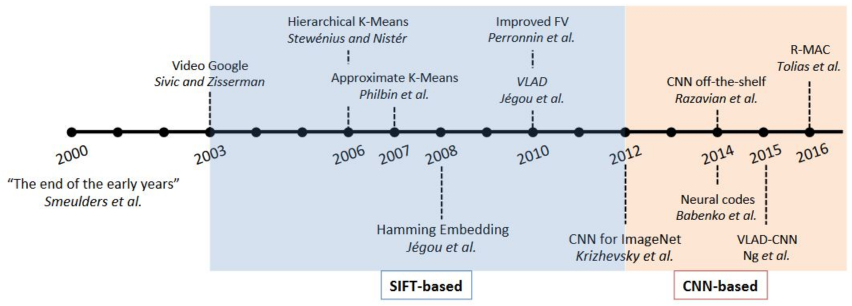


Figure 12: History of Computer Vision Algorithms,[]

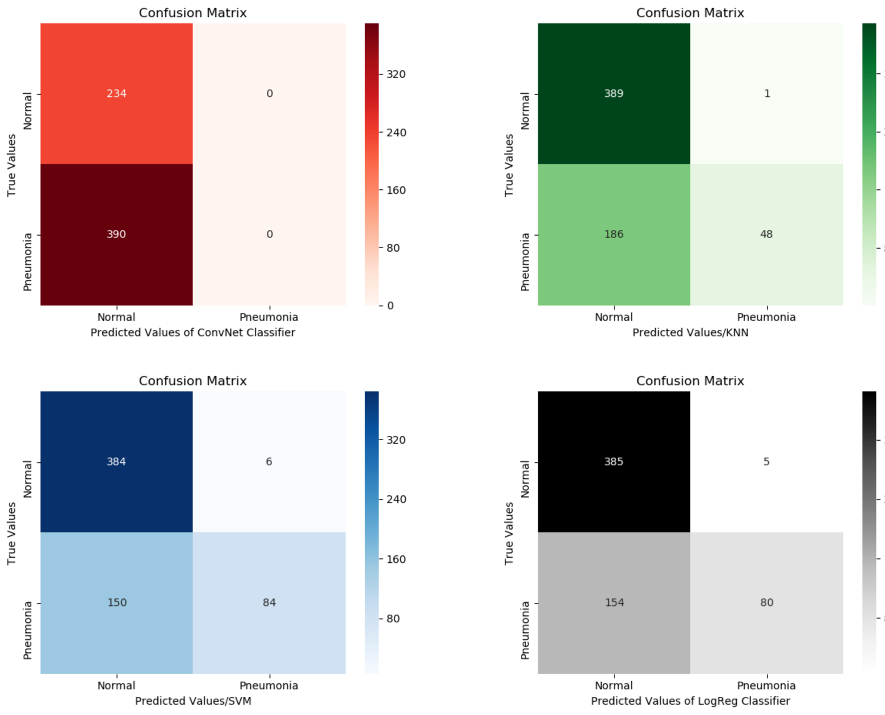


Figure 13: Confusion Matrices

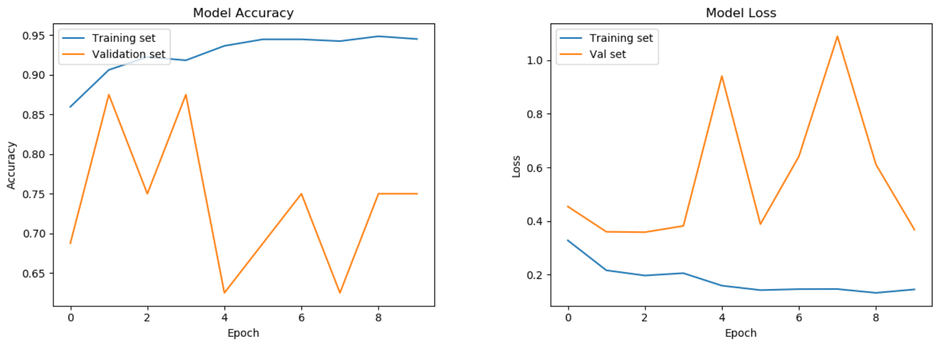


Figure 14: Accuracy and Loss of the best ConvNet Run,[]