Computer Assisted Diagnosis in Chest X-Ray Images

Cole W. Zarifis
Aurora University

A capstone project submitted in partial fulfillment of the requirements for the degree of
Actuarial Science & Mathematics
Minor in Computer Science
2023

Contents

1 Abstract

Chest X-Ray images (CXR's) are widely used and important in diagnosing a wide spectrum of diseases yet, accurate diagnosis and documentation of findings from a CXR can be unreliable [32]. Compared to traditional diagnosis, Convolutional Neural Networks (CNN's) present such benefits as allowing for potentially more accurate second opinions in diagnosis as well as providing diagnostic support for developing countries that they would otherwise not be able to provide.

An accuracy of 92.27% was achieved in binary classification on the $VinDr-CXR$ data set for the presence of a disease. Using Bayesian analysis to determine the probability of a disease being present given a positive result of the network at 56.71% and the probability of no disease present given a negative result of the network at 73.04%. The methods used show the limitations of small network designs for disease classification without pre-trained Convolutional layers.

2 Introduction

The idea for neural networks comes from an understanding of how the brain works. The brain is comprised of neurons connected in a giant web. Our brains are able to learn and form memories, not depending on each individual neuron, but the ways in which they connect. These pathways can change and connections can strengthen, leading to a preference for a particular response. Because of this, information is found in the connection or association rather than the structure of the neurons. So, our brains retain information and learn by adjusting the strength of connections while creating new pathways. This fundamental understanding of the structure of the brain is the foundation on which Neural networks are built [21].

Effective training of CNN's is reliant on high-quality, annotated data sets. Many of the existing chest X-ray data sets have not been manually annotated. Rather, the labels for the diagnosis have been acquired by the data set by using Natural Language Processing. Computer generated labels potentially leads to inaccuracies throughout the data set as a whole.

3 Literature Review

3.1 Perceptron

The first neural network was proposed by Frank Rosenblatt in the late 1950's. His formulation of the "Perceptron" was the first step in creating the neural networks that we know today. The structure is very similar to that of the brain.

A single perceptron can be thought of as a neuron. It takes in inputs that are then multiplied by weights. The weighted sum of all the inputs connected to the perceptron is then subtracted by a bias. The bias and the weights are values that change, in turn, pushing the perceptron towards the desired output [26]. Through this process, a single perceptron can classify linearly separable data by comparing the perceptron's categorization with the known categorization. If the perceptron is incorrect in its classification, the weights are updated until it is able to linearly separate the data. [1] [24] [5]

3.2 Feed-Forward Neural Networks

A multi-layer perceptron essentially takes the image of a single-layer perceptron and adds more layers horizontally. For some number X_n of inputs and $a_n^{(k)}$ neurons where n represents the number of rows in that layer and k represents the number of columns in the network which will then connect to an outputs layer y_n. In this configuration, the inputs only propagate in one direction (feed-forward),left to right. Multi-layer perceptrons are used for hard-to-classify problem structures including: handwriting, the difference between the numbers 1 and 9, and how that is represented in computer code? To simplify, neural networks are solutions to problems that are difficult to represent using traditional computer coding techniques.

Feed-forwards neural networks are the most commonly used form of neural network due to their ease of training. For example, both the number of layers and number of neurons per layer can be adjusted depending on the complexity of the problem. This form of neural network has wide-reaching, real-world applications. While normal computer programs require different algorithms based on the problem being solved, the structure of multi-layer neural networks can be taught to solve numerous classes of problems[45] [39]. Multi-layer neural networks can be taught to solve problems that involve classification [44], modeling [4], association and mapping[20].

Due to the versatility of feed forward neural networks, in 2009 they were used in modeling the drying kinetics of pistachio nuts[25]. A multi-layer neural network was chosen for this application because the network is able to learn from experimental data, handle noisy or incomplete data and is proficient at generalization.

The neural network in this example had three input layers which analyzed temperature, airflow and time with a single output neural that predicted the moisture contents of the pistachio. In between the input and output layers, there were two hidden layers, the first containing 8 neurons which then connected to another layer containing 5 neurons, which subsequently connected to the output layer. The network was then taught by using 1500 data points for training, 500 for cross validation and another 500 for testing. When the system was tested, it produced more accurate data than all the other moisture modeling techniques before it. This example clarifies how versatile multi-layer neural networks are when given real world problems. This example only uses a small amount of input data to predict a single output, but network structure used can be scaled up to handle an enormous amount of input data, using hundreds of connection layers, to predict a much larger output range.

3.3 Recurrent Neural Networks

Feed-forward neural networks can only work with static data patterns. Because of this, inputs into the network must remain the same size and form for the network to be trained. This presents an issue when trying to train a network for speech recognition, as inputs in such a case can represent single words or entire sentences. Additionally, the speed at which the subject is talking can obscure the network's recognition of what is being said[41].

Recurrent Neural Networks (RNN's) overcome this issue by not having a well-defined input space. The network is able to adjust the size of its input space depending on the size of the inputted audio file. This also means that the output space must change depending on the input data.

Another reason for the success of RNN's in speech recognition is the way in which the weights connect between each layer. Neurons do not only connect to the next layer in the network, but also to adjacent and previous neurons. In speech recognition, previous words can assist in predicting future words, so a more dense web of connection is required. This detail further separates RNN's from Feed Forward Neural Networks in that they reconnect to past neurons [33]. Due to these factors, RNN's are best used for text reading[38], speech recognition [31] and seasonal data[17].

RNN's can be used to answer simple questions, such as first-order factoids. For example, the answer to "How many cups are in a liter?", is found in a knowledge base which houses the answers to this as well as other similar questions.

In 2017, a simple RNN was trained to answer these types of questions with higher accuracy than other networks of the time [40]. The knowledge base that was used contained a large set of facts, each having a [subject, relation, object] relationship. The network was trained to understand what pieces within the question could be used to query the knowledge base correctly to get an answer. This network was able to outperform state-of-the-art networks by almost 12 percentage points by using a simple RNN design.

3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN's) were introduced to replicate the human brains pattern recognition. Compared to traditional diagnosis, CNN's present such benefits as allowing for potentially more accurate second opinions in diagnosis as well as providing diagnostic support for developing countries that they would otherwise not be able to provide. This recognition is accomplished by using the same Feed-Forward premise as multi-layer neural networks, but reducing the number of neurons in each hidden layer as you move from input to output. This type of cascading allows neurons to become more specialized at pattern recognition the closer to the output they become. This also differs from Feed-Forward neural networks where, during image recognition, they will over fit the data making any slight variation in an image have a huge impact on the outputs [39]. Each layer of the neural network is thought to have a different "comprehension of the data". This allows for more generalized understanding of the image to appear as you continue along the network. [6] [15]

Networks that detect faces within images represent great examples of this type of comprehension [18]. In these networks, each layer of data serves to further clarify the face within an image. The first 3 layers in this example simply determine if there is a face present within the image. Once a face has been confirmed, the 3 subsequent layers serve to bound a box around it. This process continues until until the location of a face within the image is specified.

4 Method

4.1 Feedforward Neural Network

The inputs into the feed-forward section of the convolutional neural networks are denoted as $\boldsymbol{x} = (x_0, x_1, ..., x_n)^T$ where there are n number of pixel input values into the densely connected portion of the network. Each neuron is denoted in the form $a_n^{(k)}$ where n is the position within the k^{th} layer. Each of the weights connecting the neurons are $w_{ij}^{(k)}$ which is the connection between the neurons $a_i^{(k-1)}$ and $a_j^{(k)}$. These connection weights form a matrix $W^{(k)} = [w_{ij}^{(k)}]$ and the neurons within a layer form a vector $\boldsymbol{a}^{(k)} = [a_n^{(k)}]$. The first value in $\boldsymbol{a}^{(k)}$ being $a_0^{(k)} = 1$ which will be the bias value, simplifying the calculation. The bias for each neuron will be the weight connection of the first neuron in the previous layer. Each neuron has an activation function $\boldsymbol{a}^{(k)} = f(W^{(k)}\boldsymbol{a}^{(k-1)})$. A layer in the neural network is the previous layers output vector multiplied by the weighted connection to the next layer which is then fed into the activation function.

4.2 Convolutional Neural Network

CNN's are used to reduce the size and extract the features from an input image, Which can then be fed into a Feed-Forward Neural Network for classification. The reduction in image size is achieved by convolution performed by a kernel or filter B. The input image is read as an array of its pixel values A_{ij}. The filter is an array of weighted values of size $k \times k$ where k is much smaller than i and j. For convolution, the kernel starts as the top corner of the image and the kernel weights are multiplied by the corresponding pixel values, then the bias value w_0 is added. This weighed sum is then a single pixel output for a new image C_{ij}, created by convolving the original image with the filter.

$$C_{ij} = B \cdot A_{ij} = \sum_{dx=0}^{x} \sum_{dy=0}^{y} B_{dx,dy} \cdot A_{i+dx,j+dy}$$

The kernel then moves a distance of s pixel values. The weighted sum is taken again and the output is the next pixel value adjacent to the previous. Once the kernel reaches the end the image in the i direction, it then moves back to its starting position and down in the j direction a distance of s, similar to a type writer. The resulting size of C_{ij} is

$$\left\lfloor \frac{i-x}{s} + 1 \right\rfloor \times \left\lfloor \frac{j-y}{s} + 1 \right\rfloor$$

where $\frac{i-x}{s}$ and $\frac{j-y}{s}$ are both $\in \mathbb{N}^+$[30]

4.2.1 Feature Extraction

The weights that are within the kernel along with the bias perform feature extraction on the image which is then displayed on the resulting image. Broadly, features are attributes found within an image. Within a single layer, there can be multiple convolutional kernels, each having weights that will be trained, producing a different kind of feature extraction. Also, there are typically many layers of convolution in the network each potentially having several different kernels. [7] For example, during the process of feature extraction of handwriting digit classification, kernel can learn to extract the edges of the digits within the first layer by adjusting weights within the kernel. Then, in a second convolutional layer, the kernels can learn to extract types of shapes, like circles and straight lines. Then a final layer of convolution can be thought of as recognizing the combinations and orientation of circles and lines which can be categorized to predict what number is within the image. There is simple feature extraction at higher resolutions where as more layers of convolution are processed, more complex features within the image can be filtered. During training, the network is taught to adjust the weights within the network to produce optimal kernels for feature extraction within each convolutional layer. [35] So, instead of passing along the entire image, attributes found within the image are passed along to a trainable classifier, like a Feed-Forward Neural Network. Using kernels allows for feature extraction regardless of the position of features within the image. [43]

Setosa.io is an excellent tool to visual how changing kernel weights has different effects on the image produced. One thing this website does not have is the bias weights being added with the weighted sum. But this could be thought of as increasing or decreasing the brightness of individual pixels on the output image.

4.2.2 Overfitting

CNN's are primarily used to reduce the amount of parameters that need to be trained. In a CNN, the total weights and bias's used is drastically reduced which helps prevent overfitting. [39]. Consider a Feed-Forward network which inputs image of size 100×100, has a hidden layer of 20 nodes and outputs 16 different categorizations. There would be $10,000$ input nodes which would be densely connected to the 20 nodes within the hidden layer, increasing the number of weights to $200,000$. The 20 hidden nodes would be densely connected to the 16 output nodes adding another 320 weights with a resulting network size of $200,320$ trainable parameters. In a feed forward network, the number of trainable parameters is found by taking the sum of products of the nodes in adjacent layers. With a convolutional neural network, the amount of trainable parameters is determined by the filter size, number of filters in each layer, the step size of the filters and number of convolutional layers. Continuing the above example for a CNN, using the same 100×100 image and each convolutional has 7 kernel of size 5×5 with a step size of 1. The size of the first convolution would be $\left\lfloor \frac{100-5}{1} + 1 \right\rfloor \times \left\lfloor \frac{100-5}{1} + 1 \right\rfloor = 96 \times 96$. So, for each convolutional layer, the size of the image will be reduced by 4. To reduce the total number of pixels in the image to 16 or a 4×4 image $\frac{100}{4} = 25$. We would need to have 24 convolutional layers of size $5 \times 5 \times 7$ giving a total number of trainable parameters of $5^2 \times 24 \times 7 = 4,200$ which is a significant reduction of trainable parameters required for the feed forward network. Even with the reduction in total parameters, convolutional neural networks have been shown to achieve a statistically higher validation accuracy to similar feed forward neural networks [13]

Another way to reduce the amount of weights within the network is to introduce pooling layers. Max pooling is similar to kernel convolution, but instead of having weights, the output pixel value is just the largest pixel value within the kernel. This can be thought of as a summarizing of the information between two convolutional layers. The size of the pooling kernel is $z \times z$ where $s < z$ so there is overlapping within the pooling layer. The equation for the resulting image size is the same for kernel convolution. Overlapping is beneficial to network performance and has been shown to: reduce the percentage of time the classifier does not give the correct class the highest probability, reduce the percentage of time the classifier does not involve the correct class among the top 5 guesses, and generally make overfitting more difficult. [14]. Within the pooling layer, it is best to keep the size of z small because information within the image is lost when using max pooling. It is best to keep the pooling kernel size to be 3×3, 5×5 or 7×7 however, options must be weighted in such a way to avoid overfitting but as to not remove too much data, which would compromise the networks ability to categorize data. [36] Network architectures may vary, but placing max pooling layers throughout the hidden layers of the network reduces the number of trainable parameters. Rather than developing connections between each neuron in subsequent layers (such as in Feed-Forward Neural Networks) a group of neurons will connect to a single neuron in the subsequent layer. Instead, a group of neurons are going to connect to a single neuron in the next layer. This is where the convolutional neural network gets its name. [2]

4.2.3 Image Categorization

Convolutional Neural networks have been shown to perform well in image categorization. Image categorization is classifying an image into one of many categories [19] [14]. For example, the MNIST dataset [16] contains handwritten images of numbers with corresponding labels which represent the number found within the image. Each image only contains one category, meaning that an image cannot contain a single handwritten digit that is both a 4 and a 7. Models have been trained with 100 percent accuracy on the test set.

While these results are impressive, they are based off a well-curated data sets. Many real-world images have multiple labels within them. Images can contain multiple instances of objects, scenes, actions and attributes which require multiple labels for a single image. Multiple-label image classification is performed by allowing the output nodes of the network $\boldsymbol{y}$ to be independent from each other [42]. Having multiple labels in a single image is a much harder task because the input image and the output of the network are substantially more complex. Another increase in complexity is the validity of the training and test data sets. It can be much more difficult to accurately curate data sets containing all of the correct image classifications. [3]

4.3 Loss Function

In order to train the weights within a network, one must compare the network's categorization of the data set compared to its actual categorization. The final output layer of the network $\boldsymbol{a}^{(k)}$ is denoted by $\boldsymbol{y} = (a_0^{(k)}, a_1^{(k)}, ..., a_n^{(k)}) = (y_0, y_1, ..., y_n)$. For every weight there is a map from $\boldsymbol{x}$ to $\boldsymbol{y}$. To find how the network is performing, it must be evaluated on training data. First, networks output $\boldsymbol{y_n}$ is evaluated against the correct categorization of the data $\boldsymbol{z_n}$ for n training examples. Every training example has two components, the input into the network and the correct classification of the training example. Testing the input into the network against the correct classification is the forward pass of the neural network. For a single test of the networks performance using mean squared error loss function being

$$\epsilon(W^{(k)}; \boldsymbol{x_1}) = \frac{1}{2}\sum[\boldsymbol{y_1} - \boldsymbol{z_1}]^2$$

The output vector of the network, $\boldsymbol{y}$ is subtracted by the correct categorization vector $\boldsymbol{z}$. The sum of the differences for each row in the column vector is taken, which is then squared. The values are squared because we want to know the distance the networks guess is from the actual value categorization of the training data. It is multiplied by $\frac{1}{2}$ to make derivation simpler when calculating how to change the weights to match the training data. We are able to multiply by an arbitrary constant because $\epsilon(W^{(k)}; \boldsymbol{x_1})$ does not have any units associated with it, the goal is to simply reduce its value by changing the weights. $\epsilon(W^{(k)}; \boldsymbol{x_1})$ will give a value of zero if the network is correct in its classification and give a positive value if it is incorrect. It is important to note that $\epsilon(W^{(k)}; \boldsymbol{x_1})$ is the loss attributed to a single training example. Then, $\epsilon(W^{(k)}; \boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{...x_n})$ can be taken over all of the training data, producing

$$E = \sum_n \epsilon(W^{(k)}; \boldsymbol{x_n})$$

E is the loss function for all of the training examples or total loss. Minimizing the loss function by adjusting the trainable weights within the network will increase the accuracy of the networks categorization of the input data. The weights will be adjusted by using stochastic gradient descent.[34] [28]

The cost function is the key to adjusting the weights within the network. As shown above, the point of the loss function is to produce a penalty for misclassification of the networks prediction and for the loss function to increase the more incorrect the networks classification is. The penalty produced decides how much the weights within the network need to be adjusted along with the learning rate. Weighted binary cross entropy is a better loss function for calculating the networks misclassification. For a single training example

$$\epsilon(W^{(k)}, \boldsymbol{\alpha}; \boldsymbol{x_1}) = -\boldsymbol{\alpha} \sum \boldsymbol{z_1} \times \log(\boldsymbol{y_1}) + (1 - \boldsymbol{z_1}) \times \log(1 - \boldsymbol{y_1})$$

The first term $\boldsymbol{z_1} \times \log(\boldsymbol{y_1})$ helps mitigate false negatives during training. For example, consider a training example has target 1 and the machines output is .7 for the first category. This means that there is a 30% probability of false negative, or the network is 30% confident in the wrong result. So, $-\log(.7) = .15$ the network is penalized more for false positive. In the perfect categorization $-\log(1) = 0$. The second term applies the same increased penalty but for false positive values. The weights $\boldsymbol{\alpha}$ is the weight associated for each of the categorizations. These weights are used to set the importance of each of the categories. This is helpful for data sets that do not have an even distribution of the occurrence of each category. For example, If a data set has two categories and category A occurs 70% of the time while category B is the other 30%. The network may learn in training to predict category A with every training example, which would make the network 70% accurate, clearly this is not ideal. So it may be better to increase the weights for minority classes, penalizing the network greater for false negatives on classes that do not occur often during training[9]

4.4 Stochastic Gradient Descent

First, to discuss gradient descent and then why stochastic gradient descent is more practical for most applications. To minimize E, which minimizes the differences between predicted and actual output for all training examples, we must compute the partial derivative of E with respect to all of the weights in the network. This is the sum of all of the derivatives of the possible input-output cases. This is considered the backwards pass which looks at the partial derivatives with respect to the weights for all the layers, starting at the output layer and moving back to the input layer. This starts by computing

$$\partial E / \partial \boldsymbol{y} = \boldsymbol{y} - \boldsymbol{z}$$

The partial derivative of E with respect to the output space is the difference between the networks output and the correct categorization. This is just the derivative of $\epsilon(W^{(k)}; \boldsymbol{x}(p))$ which has been simplified by multiplying by $\frac{1}{2}$. Then the chain rule can be applied to calculate the input into $\boldsymbol{y}$

$$\frac{\partial E}{\partial \boldsymbol{a}^{(k-1)} W^{(k)}} = \frac{\partial E}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}^{(k-1)} W^{(k)}}$$

The chain rule allows us to split function such that $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$. Then, continuing applying the chain rule, the partial derivative of the output layer $\boldsymbol{y}$ with respect to the the weighted connected outputs of the previous layer, $\boldsymbol{a}^{(k-1)}W^{(k)}$ is

$$\frac{\partial E}{\partial \boldsymbol{a}^{(k-1)}W^{(k)}} = \frac{\partial E}{\partial \boldsymbol{y}} \cdot \frac{d}{dx}f(x)$$

So, this allows us to see how an input $\boldsymbol{a}^{(k-1)}W^{(k)}$ will effect the loss function, but we need one more layer of derivation to see how the weights $W^{(k)}$ will effect the loss function.

$$\frac{\partial E}{\partial W^{(k)}} = \frac{\partial E}{\partial \boldsymbol{a}^{(k-1)}W^{(k)}} \cdot \frac{\partial \boldsymbol{a}^{(k-1)}W^{(k)}}{\partial W^{(k)}}$$

Then the partial derivative of the weighted inputs of the previous layer $\boldsymbol{a}^{(k-1)}W^{(k)}$ with respects to the weights $W^{(k)}$ are the outputs.

$$\frac{\partial E}{\partial W^{(k)}} = \frac{\partial E}{\partial \boldsymbol{a}^{(k-1)}W^{(k)}} \cdot \boldsymbol{a}^{(k-1)}$$

This leads us to

$$\frac{\partial E}{\partial \boldsymbol{a}^{(k-1)}W^{(k)}} \cdot \frac{\partial \boldsymbol{a}^{(k-1)}W^{(k)}}{\partial \boldsymbol{a}^{(k-1)}} = \frac{\partial E}{\boldsymbol{a}^{(k-1)}W^{(k)}} \cdot W^{(k)}$$

Then, we can see how the previous layers output effects the output layer

$$\frac{\partial E}{\partial \boldsymbol{a}^{(k-1)}} = \frac{\partial E}{\boldsymbol{a}^{(k-1)}W^{(k)}} \cdot W^{(k)}$$

Now, it has been laid out how each part of the network effects the loss function. This allows you to calculate the effect of the weights has on output based on the previous layer. This process can be continued for each layer in the network where you can calculate the effect of the weights on the loss function for every weight in the network regardless of how many layers the network has. The amount each of the weights is changed depends on the $\max_{1 \leq j \leq n}$loss function η which will be discussed later. So,

$$\nabla E_W = \frac{\partial E}{\partial W^{(k)}}$$

Which is more generally thought of as

$$W^{(k)}(n+1) = W^{(k)}(n) - \eta \nabla E_W(n)$$

The learning rate η dictates how large of a step the new updated weights are going to make. This method of gradient descent requires us to run through every training example before updating the weights for the network. This means that we are taking the most direct approach to the local minimum of the loss function but is extremely time consuming and processor intensive.[27] [28]

Stochastic gradient instead updates the weight of the network after each individual training example or a small subset of training examples. Instead of calculating the loss function for all of the training examples before making an update.

$$W^{(k)}(n+1) = W^{(k)}(n) - \eta \nabla \epsilon_W$$

Stochastic gradient descent leads to a less optimal approach to the local minimum. This is because instead of calculating the optimal change in weight for all training examples, the weights are only being updated for a single training example or a small number of training examples. So updated the weights happens much faster because less calculation is required for each step. This increase in speed does have the trade off of being less accurate.

[11] Since we are no longer taking the optimal route to the local minimum of the loss function, it is possible to bounce around the local minimum without actually reaching. This problem can be compounded by not choosing an appropriate learning rate to train your data thus increasing the variability of each step when updating weights.[27] [10]. These issues can be resolved by having a dynamic learning rate or by updating learning as training progresses.

Adam uses the first and second moment to calculate dynamic learning rates, So the learning rate section could be taken away, because I am just going to use the Adam optimizer in actual model calculations.

4.5 Learning Rate

When using a learning rate, to accelerate the descent to a local minimum of a loss function, there are many forms the learning rate can take. As discussed, A learning rate that is small and constant would make its way to the local minimum, but would take a long time. If the learning rate is to large, the training algorithm would diverge or bounce around the local minimum. Finding whether a learning rate is to large or small may only be know as a result of trial and error or past experience. A standard practice is starting with a large learning rate, then reducing the size of the learning as the weights approach local minimum. In a hope to reduce the iterations required to reach a local minimum faster using a cyclical learning rate.

Cyclical learning rate allows the learning rate to cycle between a range of values. This is beneficial by increasing the learning rate will have a negative short term effect on the networks performance but have a positive effect overall. There is a maximum and a minimum value that is established which is cycled through. The learning rate will increase from the minimum rate, once the maximum rate is reached the learning rate will then decrease again to the minimum rate creating one cycle. This creates a triangular pattern between minimum and maximum values because it is increasing and decreasing at the same rate.

The length of a cycle can be determined by dividing the total number of training data by the batch size. For stochastic gradient descent, the batch size is 1, so the cycle length is equal to the number of training examples. Deciding the minimum learning rate is just simply $\frac{1}{3}$ or $\frac{1}{4}$ the maximum learning rate. It is best to do a short batch of training of the network where the learning rate will increase from a small value close to zero. The learning rate will increase fast over a small amount of training data. When the accuracy of the networks begins to decrease or stagnate as the learning rate increases, that would be the best maximum learning rate value.[37]

Another approach for selecting learning rates is by using the *Adam* Algorithm. The Adam algorithm updates the learning rate from calculating the first and second moment of the gradient. The maximum learning rate is also bound help mitigate the learning rate to become to large [12].

5 Model

5.1 The Data Set

The $VinDr-CXR$ [23] data set was built from more that 100,000 raw Dicom images collected from two of the largest hospitals in Vietnam. All of the personally identifiable information has been scrubbed from the Dicom files. The images have been annotated by a team of 17 radiologists with at least 8 years of experience. Each image in the training data set was independently labeled by 3 radiologists. The data set contains 18,000 postero-anterior (front view) chest X-rays images. Due to an ongoing competition, there were 15,000 images with labels and 3,000 images that had not had the labels released. The labels consisted of 14 critical radio-graphic findings: Aortic enlargement (0), Atelectasis (1), Calcification (2), Cardiomegaly (3), Consolidation (4), ILD (5), Infiltration (6), Lung Opacity (7), Nodule/Mass (8), Other lesion (9), Pleural effusion (10), Pleural thickening (11), Pneumothorax (12), Pulmonary fibrosis (13), and No Finding (14). "No finding" does not mean that there is no disease within the X-ray, but stating that the 14 radio-graphic findings above are not found within the image. Understanding how diseases present themselves within each of the X-ray's is not required for training the CNN, but having accurate labels for images where each disease is present is how the network learns.

Dicom images have been the standard for transmission of medical images for years. It is a abstract protocol which has five primary functions: Transmission and persistence of complete object (i.e. images, waveform, documents, etc), query and retrieval of objects, applying specific actions (i.e. printing on film), workflow management (supports work lists and status information), and Quality and consistency of image appearance (for display and printing). This format was created to keep medical images and associated information standard between hospitals around the world. The Dicom standard consists of 16 parts, PSE 3.1-3.18 with the absence of 3.9 and 3.13. These categories describe different information about the patient and the object being transmitted [22].

5.2 Formatting the Data

The 15,000 images have be split with 14,000 for train and 1,000 for test (93.33%/6.66%). A high percentage split was used because of the small size of the data set as a whole. The excel file accompanying the images contained the labels for the 15,000 images, with multiple annotations per image contains 67,914 data points with the 15,000 unique image names, classname (name of the finding), classid (id of the finding), radid (Radiologist responsible for the labeling), and then bounding box location for where the radiologist specified the located of the disease within the image. The location of the disease is not considered along with radiologist findings can be omitted as well along with the name of the disease present because it is captured in the classid. After trimming the data, the only remaining columns are "class_id" and "image_id" where the processes is shown in Appendix A.1.

Each of the 15,000 images potentially has multiple labels per image. So, one-hot-encoding was used to capture all 14 possible diseases that can be present in each image. So, the 15,000 unique image id's were left in the first column, while the other columns will be the 15 possible class id's present within an image A.2. Class id's had binary labels, containing a 1 if the category is within the image id and a 0 if not .

Another issue is going to be converting all of the Dicom images to PNG. This is done to utilize Keras built in methods for creating data generators using images and labels since Dicom files are not supported for data generators. So, each image is going to be converted to .PNG using pydicom and the processes can be found in Appendix A.3.

Now that the files are converted to .PNG, Keras data generators can be used. Keras ImageDataGenerator and flow_from_dataframe combines the one-hot-labels with the images into a format which can be fed into a CNN A.4. Since CNN's have a well defined input space, all images must be scaled to the same size. The images within the data set all have different sizes, so the average size of all of the images was calculated and used for the input size into the network A.5.

5.3 Weighted Binary Cross Entropy Analysis

5.3.1 Single Model Approach

The data set has 15 different labels for each image. There can be multiple labels present in a single image with the exception of the "No Finding" label. Within the training set, "No Finding" contributes to 70 percent of all images labels. While all other labels are observed 7 percent of the time on average A.6. So, we will adjust the weights of the 14 minority classes, testing 3 different weight vectors on the same model architecture to compare accuracy on the the training set B.1. The generic architecture has 10 convolutional layers, all with relu activation function. The first four layers are connected directly with the first layer having 10 filters with kernel size of 3×3 which takes in images of size 2200 $\times$2700, second, third and forth layers having 5 filters with kernel size of 5×5.

The input image sizes are smaller but will not matter because this modeling approach is not practical which will be discussed later. All subsequent layers has max pooling of size 2×2 between each convolutional layer which contains 3 filters of size 5×5. After all the convolutional layers, the images to a feature size of 2,652 which is flattened. The flattened feature map is connected to two dense layers, first having 64 nodes using relu activation. The second dense layer is the output which contains 15 nodes and sigmoid activation function. This generic network is trained with 10 epochs, optimizer Adam with binary cross entry and metric "accuracy". This model contains 174,900 trainable parameters. The three different weight vectors being tested are weights with minority class 10× that of the "No Findings" class, 4× and 2× which are compared to not using weights at all.

Weights	Accuracy	Loss
No weights	.8003	.1341
10×	.4584	1.4028
4×	.7450	.5141
2×	.8082	.1584

The best model performance is weighing the minority class 2 times higher than the "No Findings" class. This accuracy of 80.82% is deceptive because this does not mean that the model is accurately categorizing the diseases within an image 80% of the time.

Using one-hot-encoding produces sparse positive labels, meaning there are many more occurrences of 0's than 1's. The data set contains 15,000 unique images with 15 corresponding columns associated with them. $15{,}000 \times 15 = 225{,}000$ which is the total amount of labels. Within the 14 minority classes, there is a total of 15,365 observations that have a response of 1's with all other entries having a label of 0's. Within the "No Findings" class, there are 10,606 observations that are 1's with all other responses 0's. So, If a model was created that displays 14 zeros and a single one, meaning there is no disease present within any images for the entire data set, the model would only get 15,365 incorrect responses. Resulting in an accuracy of $\frac{225{,}000-15{,}365}{225{,}000} = .9317\%$.

To combat this issue, the data set will be split into two section, a binary classifier to determine whether there is or is not a disease present within an image and a network to determine the disease that is present within an image, given there is a disease present.

5.3.2 Binary Classifier Approach

Since a binary classifier is considered, the form of the data needed to be changed. So, all 14 diseases categorizes were removed leaving only image_id and "No Findings". Within the "No Findings" column, a zero represents if a disease is present and a 1 if there is no disease, so the labels were changed to "yes" and " no" for if a disease is present or not A.7.

As before, 3 weight vectors had been tested, all using the same model, which is similar to the model used in the previous weight analysis.B.2 contained several more convolutional and max pooling layers, with the addition of another fully connected layers towards the end of the network. The network was significantly smaller with a total number of trainable parameters of 18,133 and an output space of size 2. All of the weights vectors were trained using 5 epochs on the model architecture.

Weights	Accuracy	Loss
No weights	.8982	.2590
10×	.7187	.8280
5×	.8196	.6791
2×	.8082	.8664

The model that achieved the highest accuracy used no weights while training. So, for other model architectures proposed for the binary classifier, there will be no weights used within the loss function.

5.4 Disease Categorization Models

The approach of the disease categorization models is to categorize the diseases that are present within the X-ray, given there is a disease present. Similarly to the binary classifier, the data needs to be reformatted to make sure that there are only X-ray's which contain disease. This was achieved by first removing the "No Findings" column, checking each row and creating a list of the rows which contain all zeros. Then using the list of Boolean's to remove all rows which have all 0's as entries. This code can be found in Appendix A.8. This brings the total size of the data set down to 4,394 so there will be 4,101 reserved for training with the rest used for test which is a similar split as the binary classifier of 93.33%/6.66%.

Compiling the data in this form has the same sparse labeling issue as the single model approach. Within this data set there are rows 4,394 and 14 columns which results in 4,394 × 14 = 61,516 possible labels. There still remains the 15,365 observations which are 1's while all other labels are 0's. So, if a model is constructed that produces all 0's, there would only be 15,365 incorrect responses. This would produce and accuracy of $\frac{61{,}516-15{,}365}{61{,}516} = .7502$. So the model will be able to categorize diseases with some accuracy as long as it is performing better than 75%. There is no structure within the network to ensure that at least one positive label is produced, but is a behavior that should be learned by the network through training.

5.5 Modeling Approaches

The structure of the model architecture that have been trained were inspired by *Deep Residual Learning for Image Recognition* [8]. Some of the guidelines laid out are: increasing the size of the network does not always lead to better performance or a faster training error convergence, adding more layers to suitably deep models leads to a higher training error, large networks may experience degradation of the training accuracy as training increases, double the amount of filters when the feature map is halved, perform down sampling by increasing the step size of filters instead of using max pooling layers and residual networks outperform similar model architectures. In *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases*[42], a pre-trained residual convolutional neural network outperformed all other model architectures tested using a data set of chest X-ray images for computer assisted diagnosis.

The two model approach consists of a very small data set, where the binary classifier only has 14,000 images in the training set while 4,101 images are in the training set of the disease categorization model. Such small training sets can lead to over fitting within the training set, producing reduced accuracy in the test set. Two ways to combat the issue of overfitting is to augment the data and introduce a drop out layer [14]. Label-preserving augmentations to the data can artificially increase the size of the data. This is done by training the network on images reflected horizontally or vertically. Dropout sets the outputs of each hidden neuron to zero with some a specified probability, where .5 is recommended. This technique is performed in the classifier portion of the network, not within convolutional layers. Drop out reduces combats overfitting because it reduces neurons working in conjuncture with other neurons since neurons cannot rely on output of particular other neurons.

6 Results

For each of the models tested, the test set was evaluated on the two models which received the highest final training accuracy. Each model architecture proposed was trained for different lengths, due to the processing time per epoch and if it appeared the model had converged. The convergence of the network is highlighted by also determining the highest training accuracy achieved compared to the ending accuracy. All binary classifier models can be found in A.9 and disease categorization models can be found in B.5

6.1 Binary Classifier Models

The first model is a continuation of training the "No weights" model on more epochs of the data which we will call model B.2. The decision to continue training this model is because it is already performing at an accuracy of .8982 and may improve more with continued training. This model had continued training on the unaugmented data for an additional 4 epochs, 9 epochs on the data reflected horizontally and 6 epochs on the data reflected vertically. After training on the augmented data the final training accuracy was .8934 with a total of 24 training epochs in total. B.2 performed the best of all Binary Classifier models tested. After evaluation on the test set, the model achieved accuracy of .9227. This is significantly better .7, which would have been the accuracy of a model reporting "No Finding" within every image.

Model B.3 reflects model B.2 by increasing the amount of convolutional layers while reducing the number of max pooling layers by 1. Model B.3 demonstrates how increasing the number of weights and convolutional layers within a network may not always lead to better accuracy in categorization [8]. The network was trained on 6 epochs and only achieved an accuracy of .7018 which is a reduction of accuracy of .16 from B.2 which only had 5 epochs of training initially and had an accuracy of .8982.

Model B.4 implements some guidelines discussed in 5.5 on a small scale. The amount of filters is double every time the feature map is reduced by half besides the last 4 down-sampling, which helps reduce the size of the network. There are no max pooling layers, instead alternating convolutional layers with a stride of 2 followed by a layer with strides of 1. This model also implements a dropout of .5 in the first densely connected layer. Overall, this network was trained on 9 epochs which achieving a final accuracy of .7065 with a peak training accuracy of .7231 on the second epoch.

Model B.5 built upon B.4 by increasing the amount of convolutional layers while keeping the amount down sampling layers with stride 2 the same. The feature map in this network is now smaller, so the amount of neurons in the densely connected layers is reduced, but dropout is still applied. TALK ABOUT THE ACCURACY AND STUFF YA KNOW

6.2 Disease Categorization Model

Model C.1 disregards many of the guidelines discussed in 5.5 and uses 28 convolutional layers all of filter size 3×3 sparsely separated by 7 max pooling layers of size 2×2. The number of filters per layer increases as the size of the feature map is reduced. Drop out is not used within the densely connected layers. The network was trained on 27 epochs with a final training accuracy of .6968, with the highest accuracy achieved of .7118 on the 16 epoch.

Model C.2 implements a similar structure as model C.1 while increasing the total size of the network. There are a total of 85 convolutional layers with a total of 5 max pooling layers. The model does include drop out within the first densely connected layer. After 104 epochs a training accuracy of .7020 was achieved with the maximum accuracy of .7161 during the 54 training epoch. By more than doubling the amount of convolutional layers, there is only a slight increase in accuracy from model C.1. Yet, C.2 had the best final training accuracy of all Disease Categorization model tested, overall obtaining an accuracy of .7257 on the test set. This final accuracy performs worse than a strategic guesser, which would give responses of no disease present within every image.

Model C.3 was created to evaluate if it is better to increase the amount of filters per convolutional layers as the size of the feature map decreases. So, as the feature size decreases, the number of filters per convolutional layer decreases as well. The network is smaller than C.1 and implements a drop out layer as well. After training on 9 epochs this network architecture achieved a maximum accuracy of .3568 on 4 epochs and a final accuracy of .3509 on the 9 training epoch.

Model C.4 implements almost all of the advice discussed in 5.5. The amount of filters doubles every time that the feature map is reduced by half. There are 6 down sampling convolutional layers with kernel size of 3×3 and step size of 2. Each down sampling layer is followed by 5 convolutional layers with the same with step size of 1 with the same kernel besides the last down sampling layer which only is followed by one convolutional layer. Drop out is implemented in the first dense layer. Little improvement was compared to C.2 with a final accuracy of .6969 on the 26 training epoch. The highest accuracy achieved on the 23 epoch with an accuracy of .7190 which is the highest training accuracy achieved within the disease categorization models tested.

Model C.5 implemented similar structure as C.4 by reducing the number of convolutional layers and increases the amount of down sampling layers. When training this network training, it convergence rate was similar to C.4. The accuracy was increasing while the loss function was decreasing which reached a peak accuracy of .7164 on the 11 epoch. Then the average loss began to diverge and then the accuracy of the network decreased going as low as .0184 on epoch 16. After 146 epochs of training the final accuracy of the network was .2997, which is the worst of all models tested.

6.3 Data From All Models

Model Name	Max Training Accuracy/Epoch	End Training Accuracy/Epoch	Test Accuracy
B.2	.9033/13	.8934/24	.9227
B.3	.7126/5	.7018/6	???
B.4	.7231/2	.7065/9	???
B.5	.7093/5	.7043/7	??? center
C.1	.7118/16	.6968/27	???
C.2	.7161/54	.7020/104	.7257
C.3	.3568/4	.3509/9	???
C.4	.7190/23	.6969/26	???
C.5	.7164/11	.2997/146	???

7 Conclusion

The results are inconclusive as to the best model for Disease Categorization. Assuming that there is a disease present, this model underperforms even a strategic guesser when presented with an X-ray image. This is due to the model's inability to achieve an accuracy greater than .75, which would represent the model's accuracy if it were using all 0's as a response throughout the entire set. However, the results of the binary classifier were much more Significant, as it achieved a final test accuracy of .9227. However, this is somewhat uninformative, as the important values to calculate are the probability an individual has a disease, given the network reports a disease and the probability and individual does not have a disease, given the network does not report a disease. These probabilities can be found by applying Baye's Theorem.

The data needs to be split, where one list contains all images with diseases and another list containing all images without diseases for the test set, which can be found in A.9. These two data sets can be evaluated with the model, where the accuracy's are the probability positive result given a disease and the probability of negative result given no disease. $P(+|D) = .1521$, $P(-|ND) = .9519$, $P(D) = \frac{4{,}394}{15{,}000} = .29293$ and $P(ND) = 1 - P(D) = .70706$. Using total probability, $P(+) = .07856$ and $P(-) = .92143$.

$$P(D|+) = \frac{P(+|D) \cdot P(D)}{P(+)} \approx .5671$$

$$P(ND|-) = \frac{P(-|ND) \cdot P(ND)}{P(-)} \approx .7304$$

The probability a disease is present given a positive result is .6100 and the probability no disease is present given a negative result is .7355. So, overall the model is performing better than randomly guessing when determining if an individual has one of the 14 radiological findings present within the x-ray.

The models proposed and tested have shown to be less than optimal due to limitations in computational time, storage space, network complexity and processing speed. All of the models have were trained using Google Colab Pro, which has RAM limitations of 25.51 GB and randomly assigned GPU's for training. With more powerful GPU's and increased storage, much larger model architectures could have been explored and trained for longer periods of time.

In, *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases*[42], explored several different pre-trained models on a similar Chest X-ray data set. ResNet-50 [8] achieved the lowest categorization error of models tested having a total of 3.8×10^9 total trainable parameters. ResNet-50 had been trained on ImageNet 2012 classification data set [29] which contained 1.28 million training images. The pre-trained model continued training on the Chest-Xray data set for an additional 10,000 epochs. Using pre-trained networks would have likely increased overall accuracy and reduced overall training time. Also, exploring more complex network architectures instead of combinations of pooling and convolutional layers may have increased overall accuracy as well.

Another change that could be made to improve the training of the network is by implementing a to the loss function. Within binary cross entropy, weight adjustments to the false positive and false negative calculations would most likely improve convergence. Since the one-hot-encoding scheme produces much more 0's than 1's, increasing weight for false negatives would increase the loss function much more if a false negative is produces. Clearly the networks I used had difficulty in learning positive instances, so introducing these weights would better facilitate learning positive instances, increasing overall accuracy.

A Preprocessing

A.1 Trimming Labels

```
data_set = pd.read_csv("/content/drive/MyDrive/chestXray/train.csv")

data_set.drop(data_set.iloc[:, 3:8], inplace = True, axis = 1)  box
data_set.drop('class_name', inplace=True, axis=1) #removing class_name
training_imgs = ["{}.png".format(x) for x in list(data_set.image_id)]
training_labels_1 = list(data_set['class_id'])
data_set = pd.DataFrame( {'image_id': training_imgs,'class_id':
training_labels_1})
data_set.class_id = data_set.class_id.astype(str)
data_set = data_set.sample(frac = 1)
```

A.2 One-hot-encoding

```
data_set15k = data_set.drop_duplicates(subset='image_id',keep='first')

names15k = data_set15k['image_id'].tolist()
columnames = ['image_id','0','1','2','3','4','5','6','7','8','9','10',
'11','12','13','14']
data = pd.DataFrame(columns= columnames)
multilabel = pd.DataFrame(data)
multilabel = multilabel.append(pd.DataFrame({'image_id': names15k}),
ignore_index=True)

nam15k = multilabel['image_id']
namfull = data_set['image_id']
classfull = data_set['class_id']
for i in range(len(multilabel.index)):
  for k in range(len(data_set.index)):
    if nam15k[i] == namfull[k]:
      num = int(classfull[k])
      multilabel.iloc[i,num+1] = 1
```

A.3 Converting from .Dicom to .PNG

```
test_list = [ f for f in  os.listdir(inputdir)]

for f in test_list:   # remove "[:10]" to convert all images
    ds = dcmread(inputdir + f) # read dicom image
    ds = pydicom.pixel_data_handlers.apply_voi_lut(ds.pixel_array, ds)

    shape = ds.shape
    image_2d = ds.astype(float)

    image_2d_scaled = (np.maximum(image_2d,0) / image_2d.max()) * 255.0

    img = np.uint8(image_2d_scaled)


    cv2.imwrite(outdir + f.replace('.dicom','.png'),img)
```

A.4 Data Generators

```
columns = ["0","1","2","3","4","5","6","7","8","9","10","11","12",
"13","14"]

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_dataGen = ImageDataGenerator()

train_generator = train_dataGen.flow_from_dataframe(
    dataframe = train_set,
    directory="/content/drive/MyDrive/chestXray/trainaugmented15k",
    x_col="image_id",
    y_col=columns,
    class_mode="raw",
    target_size=(2447,2787),
    color_mode='grayscale',
    batch_size=8)

test_dataGen = ImageDataGenerator()


test_generator = test_dataGen.flow_from_dataframe(
    dataframe = test_set,
    directory="/content/drive/MyDrive/chestXray/trainaugmented15k",
    x_col="image_id",
    y_col=columns,
    class_mode="raw",
    target_size=(2447,2787),
    color_mode='grayscale',
    batch_size=8)
```

A.5 Image Size

```
import PIL
import os
width2 = 0
```

```
height2 = 0
i=0
inputdir = '/content/drive/MyDrive/chestXray/trainaugmented15k/'
test_list = [ f for f in  os.listdir(inputdir)]
for i,f in enumerate(test_list):
  image = PIL.Image.open(inputdir + f)
  width, height = image.size
  width2 += int(width)
  height2 += int(height)

print(width2/(i+1),height2/(i+1))

2446.2869333333333 2786.8666
```

A.6 Label Occurrence

```
x=0
for i in range(1,15):
  x += sum(train_set.iloc[:,i])/len(train_set)
  print(i,":",sum(train_set.iloc[:,i])/len(train_set))


print("average of less likely",x/14)
print(sum(train_set.iloc[:,15])/len(train_set))
1 : 0.2040145724694621
2 : 0.012429459247089077
3 : 0.030145010357882706
4 : 0.15322523037359811
5 : 0.023287377669833558
6 : 0.025716122580184297
7 : 0.04114579612829488
8 : 0.08822058718479892
9 : 0.05514679619972855
10 : 0.07621972998071291
11 : 0.06900492892349454
12 : 0.13272376598328453
13 : 0.00635759697121223
14 : 0.10757911279377098
average of less likely 0.0732297204902391
0.7081220087149082
```

A.7 Binary Classifier Data

```
presence = multilabel.drop(['0','1','2','3','4','5','6','7','8','9',
'10','11','12','13'],axis=1)
```

```
presence=presence.rename(columns={"image_id": "image_id", "14"
: "label"})

presence['label'] = presence['label'].map({1: 'no', 0: 'yes'})

presence = presence.random.sample(frac=1)}
```

A.8 Disease Classification Data

```
class_disease = multilabel.drop(['14'],axis=1)
print(class_disease)

a_series = (class_disease.iloc[:,1:16] != 0).any(axis=1)
print(a_series)

class_disease = class_disease.loc[a_series]
```

A.9 Baye's Theorem

```
import pandas as pd
import numpy as np
multilabel= pd.read_csv("/content/drive/MyDrive/chestXray/multilabel.csv")
multilabel = multilabel.drop(multilabel.columns[0], axis=1)
```

```
import random
presence = multilabel.drop(['0','1','2','3','4','5','6','7','8','9','10'
,'11','12','13'],axis=1)

presence=presence.rename(columns={"image_id": "image_id", "14"
: "label"})

presence['label'] = presence['label'].map({1: 'no', 0: 'yes'})

presence = presence.sample(frac=1)
```

```
presence_train = presence[0:14000]
presence_test = presence[14000:15000]
```

```
yes = presence_test[presence['label']=='yes']

no = presence_test[presence['label']=='no']

1055   2461ca359068b06237e93aae140a25f3.png   yes
4193   b6d81cd3e996c836eeeadcc896afdfe4.png   yes
1964   fae3d9c7c5d474bb9cac8ac1c8912688.png   yes
1045   7ee89ae8abd9f4ebe6447e05a1b4f744.png   yes
8550   f9e2672a5b9363b4df480c9b8b773379.png   yes
...                                     ...   ...
12752  a330d6606525415c5e462b9e13ca8452.png   yes
7572   9c6b55a1b4fe76943377f346658b5da5.png   yes
4863   d015e02257639b55b196e26820e10081.png   yes
5161   1b90f72310730c8e045e85e8b60c308d.png   yes
2273   72d2a32355662fd12a16420bd689bbf9.png   yes

[280 rows x 2 columns]
                                  image_id label
6977   fecd425d8cb75dd3f38e3e8e14302fd7.png    no
10858  4b1cc160e3288b5dcc65654cbcd06307.png    no
8229   6078e5a2cb7a66c31f39997098b32fc2.png    no
1520   7936ada7b9cb7669825340ef407980e4.png    no
4816   b7dab1e641e165ce08786d35a228b6c9.png    no
...                                     ...   ...
10614  62c14fe41339b118b7fa4a0bb569f3f9.png    no
5113   bbddec90a8d4ed86861978fb09267338.png    no
2914   8b4fe0184fb8a569f32465362350e3b3.png    no
268    4d1fd8165d04d255231c3c966184cc39.png    no
3805   81c71b5013b5e7c0b0f6740e2c62dd20.png    no
```

B Binary Classifier Models

B.1 Generic Weight Analysis Pt.1

```
def create_model():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 10,
                             kernel_size = (3,3),
                             activation = 'relu',
                             input_shape = (2200,2700,1)))

  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))


  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 64, activation = 'relu'))

  #Output Layer
  classifier.add(Dense(units = 15 , activation = 'sigmoid'))

  classifier.compile(optimizer = 'adam',
                  loss = 'binary_crossentropy',
                  metrics = ['accuracy'])

  return classifier
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 2198, 2698, 10)    100
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 2194, 2694, 5)     1255
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2190, 2690, 5)     630
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2186, 2686, 5)     630
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1093, 1343, 5)     0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 1089, 1339, 3)     378
_________________________________________________________________
```

```
max_pooling2d_7 (MaxPooling2 (None, 544, 669, 3)       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 540, 665, 3)       228
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 270, 332, 3)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 266, 328, 3)       228
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 133, 164, 3)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 129, 160, 3)       228
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 64, 80, 3)         0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 60, 76, 3)         228
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 30, 38, 3)         0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 26, 34, 3)         228
_________________________________________________________________
flatten_1 (Flatten)          (None, 2652)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                169792
_________________________________________________________________
dense_3 (Dense)              (None, 15)                975
=================================================================
Total params: 174,900
Trainable params: 174,900
Non-trainable params: 0
_________________________________________________________________
```

B.2 Generic Weight Analysis Pt.2

```
def create_model():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 10,
                             kernel_size = (3,3),
                             activation = 'relu',
                             input_shape = (2447,2787,1)))

  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(Conv2D(5,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu')) #7
  classifier.add(MaxPooling2D(pool_size = (2,2)))



  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 90, activation = 'relu'))

  #Output Layer
  classifier.add(Dense(units = 60 , activation = 'relu'))

  classifier.add(Dense(units=2 , activation = 'softmax'))
  classifier.compile(optimizer = 'adam',
                  loss = 'binary_crossentropy',
                  metrics = ['accuracy'])

  return classifier
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 2445, 2785, 10)    100
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 2441, 2781, 5)     1255
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2437, 2777, 5)     630
```

```
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2433, 2773, 5)     630       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1216, 1386, 5)     0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1212, 1382, 3)     378       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 606, 691, 3)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 602, 687, 3)       228       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 301, 343, 3)       0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 297, 339, 3)       228       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 148, 169, 3)       0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 144, 165, 3)       228       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 72, 82, 3)         0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 68, 78, 3)         228       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 34, 39, 3)         0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 30, 35, 3)         228       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 17, 3)         0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 11, 13, 3)         228       
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 6, 3)           0         
_________________________________________________________________
flatten (Flatten)            (None, 90)                0         
_________________________________________________________________
dense (Dense)                (None, 90)                8190      
_________________________________________________________________
dense_1 (Dense)              (None, 60)                5460      
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 122       
=================================================================
Total params: 18,133
Trainable params: 18,133
Non-trainable params: 0
_________________________________________________________________
```

B.3 Expansion of Weight Analysis

```
def large_classifier():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 10,
                             kernel_size = (3,3),
                             activation = 'relu',
                             input_shape = (2447,2787,1)))

  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(Conv2D(5,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))


  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))
  classifier.add(Conv2D(3,(5,5),activation = 'relu'))




  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 90, activation = 'relu'))

  #Output Layer
  classifier.add(Dense(units = 60 , activation = 'relu'))

  classifier.add(Dense(units=2 , activation = 'softmax'))
  classifier.compile(optimizer = 'adam',
                    loss = 'binary_crossentropy',
                    metrics = ['accuracy'])

  return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_617 (Conv2D)          (None, 2445, 2785, 10)    100
_________________________________________________________________
conv2d_618 (Conv2D)          (None, 2443, 2783, 5)     455
_________________________________________________________________
conv2d_619 (Conv2D)          (None, 2441, 2781, 5)     230
_________________________________________________________________
conv2d_620 (Conv2D)          (None, 2439, 2779, 5)     230
_________________________________________________________________
conv2d_621 (Conv2D)          (None, 2437, 2777, 5)     230
_________________________________________________________________
conv2d_622 (Conv2D)          (None, 2435, 2775, 5)     230
_________________________________________________________________
conv2d_623 (Conv2D)          (None, 2433, 2773, 5)     230
_________________________________________________________________
```

```
max_pooling2d_188 (MaxPoolin (None, 1216, 1386, 5)     0
_________________________________________________________________
conv2d_624 (Conv2D)          (None, 1212, 1382, 3)     378
_________________________________________________________________
conv2d_625 (Conv2D)          (None, 1208, 1378, 3)     228
_________________________________________________________________
conv2d_626 (Conv2D)          (None, 1204, 1374, 3)     228
_________________________________________________________________
conv2d_627 (Conv2D)          (None, 1200, 1370, 3)     228
_________________________________________________________________
max_pooling2d_189 (MaxPoolin (None, 600, 685, 3)       0
_________________________________________________________________
conv2d_628 (Conv2D)          (None, 596, 681, 3)       228
_________________________________________________________________
conv2d_629 (Conv2D)          (None, 592, 677, 3)       228
_________________________________________________________________
conv2d_630 (Conv2D)          (None, 588, 673, 3)       228
_________________________________________________________________
conv2d_631 (Conv2D)          (None, 584, 669, 3)       228
_________________________________________________________________
max_pooling2d_190 (MaxPoolin (None, 292, 334, 3)       0
_________________________________________________________________
conv2d_632 (Conv2D)          (None, 288, 330, 3)       228
_________________________________________________________________
conv2d_633 (Conv2D)          (None, 284, 326, 3)       228
_________________________________________________________________
conv2d_634 (Conv2D)          (None, 280, 322, 3)       228
_________________________________________________________________
conv2d_635 (Conv2D)          (None, 276, 318, 3)       228
_________________________________________________________________
max_pooling2d_191 (MaxPoolin (None, 138, 159, 3)       0
_________________________________________________________________
conv2d_636 (Conv2D)          (None, 134, 155, 3)       228
_________________________________________________________________
conv2d_637 (Conv2D)          (None, 130, 151, 3)       228
_________________________________________________________________
conv2d_638 (Conv2D)          (None, 126, 147, 3)       228
_________________________________________________________________
conv2d_639 (Conv2D)          (None, 122, 143, 3)       228
_________________________________________________________________
max_pooling2d_192 (MaxPoolin (None, 61, 71, 3)         0
_________________________________________________________________
conv2d_640 (Conv2D)          (None, 57, 67, 3)         228
_________________________________________________________________
conv2d_641 (Conv2D)          (None, 53, 63, 3)         228
_________________________________________________________________
```

```
conv2d_642 (Conv2D)          (None, 49, 59, 3)         228
_________________________________________________________________
conv2d_643 (Conv2D)          (None, 45, 55, 3)         228
_________________________________________________________________
max_pooling2d_193 (MaxPoolin (None, 22, 27, 3)         0
_________________________________________________________________
conv2d_644 (Conv2D)          (None, 18, 23, 3)         228
_________________________________________________________________
conv2d_645 (Conv2D)          (None, 14, 19, 3)         228
_________________________________________________________________
conv2d_646 (Conv2D)          (None, 10, 15, 3)         228
_________________________________________________________________
conv2d_647 (Conv2D)          (None, 6, 11, 3)          228
_________________________________________________________________
flatten_27 (Flatten)         (None, 198)               0
_________________________________________________________________
dense_81 (Dense)             (None, 90)                17910
_________________________________________________________________
dense_82 (Dense)             (None, 60)                5460
_________________________________________________________________
dense_83 (Dense)             (None, 2)                 122
=================================================================
Total params: 30,819
Trainable params: 30,819
Non-trainable params: 0
_________________________________________________________________
```

B.4 Following Guidance

```
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout,
BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers

def smaller2_large_dropout():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 4,
                             kernel_size = (3,3),
                             strides = (2,2),
                             activation = 'relu',
                             input_shape = (2447,2787,1)))


  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))




  classifier.add(Conv2D(16,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))


  classifier.add(Conv2D(32,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))



  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))


  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))


  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))


  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))







  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 1000, activation = 'relu'))
  classifier.add(Dropout(0.5))
  #Output Layer
  classifier.add(Dense(units = 300 , activation = 'relu'))
```

```
    classifier.add(Dense(units=2 , activation = 'softmax'))
    classifier.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])

    return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 1223, 1393, 4)     40
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 1221, 1391, 4)     148
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 610, 695, 8)       296
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 608, 693, 8)       584
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 303, 346, 16)      1168
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 301, 344, 16)      2320
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 150, 171, 32)      4640
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 148, 169, 32)      9248
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 73, 84, 64)        18496
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 71, 82, 64)        36928
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 35, 40, 64)        36928
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 33, 38, 64)        36928
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 16, 18, 128)       73856
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 14, 16, 128)       147584
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 6, 7, 128)         147584
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 4, 5, 128)         147584
_________________________________________________________________
flatten_1 (Flatten)          (None, 2560)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1000)              2561000
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_4 (Dense)              (None, 300)               300300
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 602
=================================================================
Total params: 3,526,234
Trainable params: 3,526,234
Non-trainable params: 0
_________________________________________________________________
```

B.5 Increase Guidelines

```
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers

def smaller3_large_dropout():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 4,
                            kernel_size = (3,3),
                            strides = (2,2),
                            activation = 'relu',
                            input_shape = (2447,2787,1)))


  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))


  classifier.add(Conv2D(8,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))



  classifier.add(Conv2D(16,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))

  classifier.add(Conv2D(32,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))

  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))

  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))

  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))
```

```
  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 768, activation = 'relu'))
  classifier.add(Dropout(0.5))
  #Output Layer
  classifier.add(Dense(units = 300 , activation = 'relu'))

  classifier.add(Dense(units=2 , activation = 'softmax'))
  classifier.compile(optimizer = 'adam',
                     loss = 'categorical_crossentropy',
                     metrics = ['accuracy'])

  return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_117 (Conv2D)          (None, 1223, 1393, 4)     40
_________________________________________________________________
conv2d_118 (Conv2D)          (None, 1221, 1391, 4)     148
_________________________________________________________________
```

```
conv2d_119 (Conv2D)          (None, 1219, 1389, 4)     148
_________________________________________________________________
conv2d_120 (Conv2D)          (None, 609, 694, 8)       296
_________________________________________________________________
conv2d_121 (Conv2D)          (None, 607, 692, 8)       584
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 605, 690, 8)       584
_________________________________________________________________
conv2d_123 (Conv2D)          (None, 302, 344, 16)      1168
_________________________________________________________________
conv2d_124 (Conv2D)          (None, 300, 342, 16)      2320
_________________________________________________________________
conv2d_125 (Conv2D)          (None, 298, 340, 16)      2320
_________________________________________________________________
conv2d_126 (Conv2D)          (None, 148, 169, 32)      4640
_________________________________________________________________
conv2d_127 (Conv2D)          (None, 146, 167, 32)      9248
_________________________________________________________________
conv2d_128 (Conv2D)          (None, 144, 165, 32)      9248
_________________________________________________________________
conv2d_129 (Conv2D)          (None, 71, 82, 64)        18496
_________________________________________________________________
conv2d_130 (Conv2D)          (None, 69, 80, 64)        36928
_________________________________________________________________
conv2d_131 (Conv2D)          (None, 67, 78, 64)        36928
_________________________________________________________________
conv2d_132 (Conv2D)          (None, 33, 38, 64)        36928
_________________________________________________________________
conv2d_133 (Conv2D)          (None, 31, 36, 64)        36928
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 29, 34, 64)        36928
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 14, 16, 128)       73856
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 12, 14, 128)       147584
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 10, 12, 128)       147584
_________________________________________________________________
conv2d_138 (Conv2D)          (None, 4, 5, 128)         147584
_________________________________________________________________
conv2d_139 (Conv2D)          (None, 2, 3, 128)         147584
_________________________________________________________________
flatten_3 (Flatten)          (None, 768)               0
_________________________________________________________________
dense_9 (Dense)              (None, 768)               590592
_________________________________________________________________
```

```
dropout_3 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_10 (Dense)             (None, 300)               230700
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 602
=================================================================
Total params: 1,719,966
Trainable params: 1,719,966
Non-trainable params: 0
_________________________________________________________________
```

C Disease Categorization Models

C.1 first

```
def create_model_nodrop_v2():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 2,
                              kernel_size = (3,3),
                              activation = 'relu',
                              input_shape = (2447,2787,1)))



  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))

  classifier.add(Conv2D(64,(3,3),activation = 'relu'))





  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 2240, activation = 'relu'))

  classifier.add(Dense(units = 100, activation = 'relu'))

  classifier.add(Dense(units=14 , activation = 'sigmoid'))
  classifier.compile(optimizer = 'adam',
                  loss = 'categorical_crossentropy',
                  metrics = ['accuracy'])

  return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_139 (Conv2D)          (None, 2445, 2785, 2)     20
_________________________________________________________________
conv2d_140 (Conv2D)          (None, 2443, 2783, 4)     76
_________________________________________________________________
conv2d_141 (Conv2D)          (None, 2441, 2781, 8)     296
_________________________________________________________________
conv2d_142 (Conv2D)          (None, 2439, 2779, 16)    1168
_________________________________________________________________
conv2d_143 (Conv2D)          (None, 2437, 2777, 16)    2320
```

```
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 1218, 1388, 16)    0
_________________________________________________________________
conv2d_144 (Conv2D)          (None, 1216, 1386, 16)    2320
_________________________________________________________________
conv2d_145 (Conv2D)          (None, 1214, 1384, 16)    2320
_________________________________________________________________
conv2d_146 (Conv2D)          (None, 1212, 1382, 16)    2320
_________________________________________________________________
conv2d_147 (Conv2D)          (None, 1210, 1380, 16)    2320
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 605, 690, 16)      0
_________________________________________________________________
conv2d_148 (Conv2D)          (None, 603, 688, 32)      4640
_________________________________________________________________
conv2d_149 (Conv2D)          (None, 601, 686, 32)      9248
_________________________________________________________________
conv2d_150 (Conv2D)          (None, 599, 684, 32)      9248
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 299, 342, 32)      0
_________________________________________________________________
conv2d_151 (Conv2D)          (None, 297, 340, 32)      9248
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 295, 338, 32)      9248
_________________________________________________________________
conv2d_153 (Conv2D)          (None, 293, 336, 32)      9248
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 146, 168, 32)      0
_________________________________________________________________
conv2d_154 (Conv2D)          (None, 144, 166, 64)      18496
_________________________________________________________________
conv2d_155 (Conv2D)          (None, 142, 164, 64)      36928
_________________________________________________________________
conv2d_156 (Conv2D)          (None, 140, 162, 64)      36928
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 70, 81, 64)        0
_________________________________________________________________
conv2d_157 (Conv2D)          (None, 68, 79, 64)        36928
_________________________________________________________________
conv2d_158 (Conv2D)          (None, 66, 77, 64)        36928
_________________________________________________________________
conv2d_159 (Conv2D)          (None, 64, 75, 64)        36928
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 32, 37, 64)        0
_________________________________________________________________
conv2d_160 (Conv2D)          (None, 30, 35, 64)        36928
```

```
_________________________________________________________________
conv2d_161 (Conv2D)          (None, 28, 33, 64)        36928
_________________________________________________________________
conv2d_162 (Conv2D)          (None, 26, 31, 64)        36928
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 13, 15, 64)        0
_________________________________________________________________
conv2d_163 (Conv2D)          (None, 11, 13, 64)        36928
_________________________________________________________________
conv2d_164 (Conv2D)          (None, 9, 11, 64)         36928
_________________________________________________________________
conv2d_165 (Conv2D)          (None, 7, 9, 64)          36928
_________________________________________________________________
conv2d_166 (Conv2D)          (None, 5, 7, 64)          36928
_________________________________________________________________
flatten_5 (Flatten)          (None, 2240)              0
_________________________________________________________________
dense_13 (Dense)             (None, 2240)              5019840
_________________________________________________________________
dense_14 (Dense)             (None, 100)               224100
_________________________________________________________________
dense_15 (Dense)             (None, 14)                1414
=================================================================
Total params: 5,771,026
Trainable params: 5,771,026
Non-trainable params: 0

_________________________________________________________________
```

C.2 second

```
def really_large_dropout():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 4,
                            kernel_size = (3,3),
                            activation = 'relu',
                            input_shape = (2447,2787,1)))

  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu')) #13
  #13 convolutional
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  #13 Convolutional
  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))

  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))

  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))

  classifier.add(MaxPooling2D(pool_size = (2,2)))

  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))

  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
```

```
    #Flattening
    classifier.add(Flatten())

    #Hidden Layer
    classifier.add(Dense(units = 3696, activation = 'relu'))
    classifier.add(Dropout(0.5))
    #Output Layer
    classifier.add(Dense(units = 300 , activation = 'relu'))

    classifier.add(Dense(units=14 , activation = 'sigmoid'))
    classifier.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])

    return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 2445, 2785, 4)     40
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 2443, 2783, 4)     148
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2441, 2781, 4)     148
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2439, 2779, 4)     148
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2437, 2777, 4)     148
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 2435, 2775, 4)     148
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2433, 2773, 4)     148
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 2431, 2771, 4)     148
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2429, 2769, 4)     148
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 2427, 2767, 4)     148
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 2425, 2765, 4)     148
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 2423, 2763, 4)     148
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2421, 2761, 4)     148
_________________________________________________________________
```

```
max_pooling2d (MaxPooling2D) (None, 1210, 1380, 4)     0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 1208, 1378, 8)     296
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 1206, 1376, 8)     584
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 1204, 1374, 8)     584
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 1202, 1372, 8)     584
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 1200, 1370, 8)     584
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 1198, 1368, 8)     584
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 1196, 1366, 8)     584
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 1194, 1364, 8)     584
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 1192, 1362, 8)     584
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 1190, 1360, 8)     584
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 1188, 1358, 8)     584
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 1186, 1356, 8)     584
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 1184, 1354, 8)     584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 592, 677, 8)       0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 590, 675, 16)      1168
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 588, 673, 16)      2320
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 586, 671, 16)      2320
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 584, 669, 16)      2320
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 582, 667, 16)      2320
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 580, 665, 16)      2320
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 578, 663, 16)      2320
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 576, 661, 16)      2320
_________________________________________________________________
```

```
conv2d_34 (Conv2D)           (None, 574, 659, 16)      2320      
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 572, 657, 16)      2320      
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 570, 655, 16)      2320      
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 568, 653, 16)      2320      
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 566, 651, 16)      2320      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 283, 325, 16)      0         
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 281, 323, 32)      4640      
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 279, 321, 32)      9248      
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 277, 319, 32)      9248      
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 275, 317, 32)      9248      
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 273, 315, 32)      9248      
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 271, 313, 32)      9248      
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 269, 311, 32)      9248      
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 267, 309, 32)      9248      
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 265, 307, 32)      9248      
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 263, 305, 32)      9248      
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 261, 303, 32)      9248      
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 259, 301, 32)      9248      
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 257, 299, 32)      9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 128, 149, 32)      0         
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 126, 147, 32)      9248      
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 124, 145, 32)      9248      
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 122, 143, 32)      9248      
_________________________________________________________________
```

```
conv2d_55 (Conv2D)           (None, 120, 141, 32)      9248
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 118, 139, 32)      9248
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 116, 137, 32)      9248
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 114, 135, 32)      9248
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 112, 133, 32)      9248
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 110, 131, 32)      9248
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 108, 129, 32)      9248
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 106, 127, 32)      9248
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 104, 125, 32)      9248
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 102, 123, 32)      9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 51, 61, 32)        0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 49, 59, 32)        9248
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 47, 57, 32)        9248
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 45, 55, 32)        9248
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 43, 53, 32)        9248
_________________________________________________________________
conv2d_69 (Conv2D)           (None, 41, 51, 32)        9248
_________________________________________________________________
conv2d_70 (Conv2D)           (None, 39, 49, 32)        9248
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 37, 47, 32)        9248
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 35, 45, 32)        9248
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 33, 43, 32)        9248
_________________________________________________________________
conv2d_74 (Conv2D)           (None, 31, 41, 32)        9248
_________________________________________________________________
conv2d_75 (Conv2D)           (None, 29, 39, 32)        9248
_________________________________________________________________
conv2d_76 (Conv2D)           (None, 27, 37, 32)        9248
_________________________________________________________________
```

```
conv2d_77 (Conv2D)           (None, 25, 35, 32)        9248
_________________________________________________________________
conv2d_78 (Conv2D)           (None, 23, 33, 16)        4624
_________________________________________________________________
conv2d_79 (Conv2D)           (None, 21, 31, 16)        2320
_________________________________________________________________
conv2d_80 (Conv2D)           (None, 19, 29, 16)        2320
_________________________________________________________________
conv2d_81 (Conv2D)           (None, 17, 27, 16)        2320
_________________________________________________________________
conv2d_82 (Conv2D)           (None, 15, 25, 16)        2320
_________________________________________________________________
conv2d_83 (Conv2D)           (None, 13, 23, 16)        2320
_________________________________________________________________
conv2d_84 (Conv2D)           (None, 11, 21, 16)        2320
_________________________________________________________________
flatten (Flatten)            (None, 3696)              0
_________________________________________________________________
dense (Dense)                (None, 3696)              13664112
_________________________________________________________________
dropout (Dropout)            (None, 3696)              0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               1109100
_________________________________________________________________
dense_2 (Dense)              (None, 14)                4214
=================================================================
Total params: 15,190,162
Trainable params: 15,190,162
Non-trainable params: 0
_________________________________________________________________
```

C.3 Third

```
def create_model_nodrop_v2():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 8,
                             kernel_size = (3,3),
                             activation = 'relu',
                             strides = (3,3),
                             input_shape = (2447,2787,1)))



  classifier.add(Conv2D(32,(3,3), strides=(3,3), activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2,2)))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))







  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 1008, activation = 'relu'))

  classifier.add(Dense(units = 500, activation = 'relu'))
  classifier.add(Dropout(0.5))
  classifier.add(Dense(units=14 , activation = 'sigmoid'))
  classifier.compile(optimizer = 'adam',
                   loss = 'categorical_crossentropy',
                   metrics = ['accuracy'])

  return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_656 (Conv2D)          (None, 815, 929, 8)       80
_________________________________________________________________
conv2d_657 (Conv2D)          (None, 271, 309, 32)      2336
```

```
_________________________________________________________________
conv2d_658 (Conv2D)          (None, 269, 307, 32)      9248
_________________________________________________________________
conv2d_659 (Conv2D)          (None, 267, 305, 32)      9248
_________________________________________________________________
conv2d_660 (Conv2D)          (None, 265, 303, 32)      9248
_________________________________________________________________
max_pooling2d_134 (MaxPoolin (None, 132, 151, 32)      0
_________________________________________________________________
conv2d_661 (Conv2D)          (None, 130, 149, 32)      9248
_________________________________________________________________
conv2d_662 (Conv2D)          (None, 128, 147, 32)      9248
_________________________________________________________________
conv2d_663 (Conv2D)          (None, 126, 145, 16)      4624
_________________________________________________________________
conv2d_664 (Conv2D)          (None, 124, 143, 16)      2320
_________________________________________________________________
max_pooling2d_135 (MaxPoolin (None, 62, 71, 16)        0
_________________________________________________________________
conv2d_665 (Conv2D)          (None, 60, 69, 8)         1160
_________________________________________________________________
conv2d_666 (Conv2D)          (None, 58, 67, 8)         584
_________________________________________________________________
conv2d_667 (Conv2D)          (None, 56, 65, 8)         584
_________________________________________________________________
max_pooling2d_136 (MaxPoolin (None, 28, 32, 8)         0
_________________________________________________________________
conv2d_668 (Conv2D)          (None, 26, 30, 8)         584
_________________________________________________________________
conv2d_669 (Conv2D)          (None, 24, 28, 8)         584
_________________________________________________________________
conv2d_670 (Conv2D)          (None, 22, 26, 8)         584
_________________________________________________________________
conv2d_671 (Conv2D)          (None, 20, 24, 8)         584
_________________________________________________________________
conv2d_672 (Conv2D)          (None, 18, 22, 8)         584
_________________________________________________________________
conv2d_673 (Conv2D)          (None, 16, 20, 8)         584
_________________________________________________________________
conv2d_674 (Conv2D)          (None, 14, 18, 4)         292
_________________________________________________________________
flatten_25 (Flatten)         (None, 1008)              0
_________________________________________________________________
dense_75 (Dense)             (None, 1008)              1017072
_________________________________________________________________
dense_76 (Dense)             (None, 500)               504500
```

```
_________________________________________________________________
dropout_19 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_77 (Dense)             (None, 14)                7014
=================================================================
Total params: 1,590,310
Trainable params: 1,590,310
Non-trainable params: 0
_________________________________________________________________
```

C.4 Fourth

```
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers

def smaller_large_dropout():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 4,
                             kernel_size = (3,3),
                             strides = (2,2),
                             activation = 'relu',
                             input_shape = (2447,2787,1)))


  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))
  classifier.add(Conv2D(4,(3,3),activation = 'relu'))


  classifier.add(Conv2D(8,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))



  classifier.add(Conv2D(16,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))


  classifier.add(Conv2D(32,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))


  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))

  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))

  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))






  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 1000, activation = 'relu'))
  classifier.add(Dropout(0.5))
  #Output Layer
  classifier.add(Dense(units = 300 , activation = 'relu'))

  classifier.add(Dense(units=14 , activation = 'sigmoid'))
  classifier.compile(optimizer = 'adam',
                    loss = 'categorical_crossentropy',
                    metrics = ['accuracy'])

    return classifier
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_389 (Conv2D)          (None, 1223, 1393, 4)     40
_________________________________________________________________
conv2d_390 (Conv2D)          (None, 1221, 1391, 4)     148
_________________________________________________________________
conv2d_391 (Conv2D)          (None, 1219, 1389, 4)     148
_________________________________________________________________
conv2d_392 (Conv2D)          (None, 1217, 1387, 4)     148
_________________________________________________________________
conv2d_393 (Conv2D)          (None, 1215, 1385, 4)     148
_________________________________________________________________
conv2d_394 (Conv2D)          (None, 1213, 1383, 4)     148
_________________________________________________________________
conv2d_395 (Conv2D)          (None, 606, 691, 8)       296
_________________________________________________________________
conv2d_396 (Conv2D)          (None, 604, 689, 8)       584
_________________________________________________________________
conv2d_397 (Conv2D)          (None, 602, 687, 8)       584
_________________________________________________________________
conv2d_398 (Conv2D)          (None, 600, 685, 8)       584
_________________________________________________________________
conv2d_399 (Conv2D)          (None, 598, 683, 8)       584
_________________________________________________________________
conv2d_400 (Conv2D)          (None, 596, 681, 8)       584
_________________________________________________________________
conv2d_401 (Conv2D)          (None, 297, 340, 16)      1168
_________________________________________________________________
conv2d_402 (Conv2D)          (None, 295, 338, 16)      2320
_________________________________________________________________
conv2d_403 (Conv2D)          (None, 293, 336, 16)      2320
_________________________________________________________________
conv2d_404 (Conv2D)          (None, 291, 334, 16)      2320
_________________________________________________________________
conv2d_405 (Conv2D)          (None, 289, 332, 16)      2320
_________________________________________________________________
conv2d_406 (Conv2D)          (None, 287, 330, 16)      2320
_________________________________________________________________
conv2d_407 (Conv2D)          (None, 143, 164, 32)      4640
_________________________________________________________________
conv2d_408 (Conv2D)          (None, 141, 162, 32)      9248
```

```
_________________________________________________________________
conv2d_409 (Conv2D)          (None, 139, 160, 32)      9248
_________________________________________________________________
conv2d_410 (Conv2D)          (None, 137, 158, 32)      9248
_________________________________________________________________
conv2d_411 (Conv2D)          (None, 135, 156, 32)      9248
_________________________________________________________________
conv2d_412 (Conv2D)          (None, 133, 154, 32)      9248
_________________________________________________________________
conv2d_413 (Conv2D)          (None, 66, 76, 64)        18496
_________________________________________________________________
conv2d_414 (Conv2D)          (None, 64, 74, 64)        36928
_________________________________________________________________
conv2d_415 (Conv2D)          (None, 62, 72, 64)        36928
_________________________________________________________________
conv2d_416 (Conv2D)          (None, 60, 70, 64)        36928
_________________________________________________________________
conv2d_417 (Conv2D)          (None, 58, 68, 64)        36928
_________________________________________________________________
conv2d_418 (Conv2D)          (None, 56, 66, 64)        36928
_________________________________________________________________
conv2d_419 (Conv2D)          (None, 27, 32, 64)        36928
_________________________________________________________________
conv2d_420 (Conv2D)          (None, 25, 30, 64)        36928
_________________________________________________________________
conv2d_421 (Conv2D)          (None, 23, 28, 64)        36928
_________________________________________________________________
conv2d_422 (Conv2D)          (None, 21, 26, 64)        36928
_________________________________________________________________
conv2d_423 (Conv2D)          (None, 19, 24, 64)        36928
_________________________________________________________________
conv2d_424 (Conv2D)          (None, 17, 22, 64)        36928
_________________________________________________________________
conv2d_425 (Conv2D)          (None, 8, 10, 64)         36928
_________________________________________________________________
conv2d_426 (Conv2D)          (None, 6, 8, 64)          36928
_________________________________________________________________
flatten_10 (Flatten)         (None, 3072)              0
_________________________________________________________________
dense_30 (Dense)             (None, 1000)              3073000
_________________________________________________________________
dropout_10 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_31 (Dense)             (None, 300)               300300
_________________________________________________________________
dense_32 (Dense)             (None, 14)                4214
```

```
=================================================================
Total params: 3,943,718
Trainable params: 3,943,718
Non-trainable params: 0
_________________________________________________________________
```

C.5 Fifth

```
def smaller2_large_dropout():
  classifier = Sequential()
  classifier.add(Conv2D(filters = 4,
                             kernel_size = (3,3),
                             strides = (2,2),
                             activation = 'relu',
                             input_shape = (2447,2787,1)))


  classifier.add(Conv2D(4,(3,3),activation = 'relu'))


  classifier.add(Conv2D(8,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(8,(3,3),activation = 'relu'))




  classifier.add(Conv2D(16,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(16,(3,3),activation = 'relu'))


  classifier.add(Conv2D(32,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(32,(3,3),activation = 'relu'))



  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))


  classifier.add(Conv2D(64,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(64,(3,3),activation = 'relu'))


  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),strides = (2,2),activation = 'relu'))
  classifier.add(Conv2D(128,(3,3),activation = 'relu'))


  #Flattening
  classifier.add(Flatten())

  #Hidden Layer
  classifier.add(Dense(units = 1000, activation = 'relu'))
  classifier.add(Dropout(0.5))
  #Output Layer
  classifier.add(Dense(units = 300 , activation = 'relu'))

  classifier.add(Dense(units=14 , activation = 'sigmoid'))
  classifier.compile(optimizer = 'adam',
                     loss = 'categorical_crossentropy',
                     metrics = ['accuracy'])

  return classifier
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_573 (Conv2D)          (None, 1223, 1393, 4)     40
_________________________________________________________________
conv2d_574 (Conv2D)          (None, 1221, 1391, 4)     148
_________________________________________________________________
conv2d_575 (Conv2D)          (None, 610, 695, 8)       296
_________________________________________________________________
conv2d_576 (Conv2D)          (None, 608, 693, 8)       584
_________________________________________________________________
conv2d_577 (Conv2D)          (None, 303, 346, 16)      1168
_________________________________________________________________
conv2d_578 (Conv2D)          (None, 301, 344, 16)      2320
_________________________________________________________________
conv2d_579 (Conv2D)          (None, 150, 171, 32)      4640
_________________________________________________________________
conv2d_580 (Conv2D)          (None, 148, 169, 32)      9248
_________________________________________________________________
conv2d_581 (Conv2D)          (None, 73, 84, 64)        18496
_________________________________________________________________
conv2d_582 (Conv2D)          (None, 71, 82, 64)        36928
_________________________________________________________________
conv2d_583 (Conv2D)          (None, 35, 40, 64)        36928
_________________________________________________________________
conv2d_584 (Conv2D)          (None, 33, 38, 64)        36928
```

```
_________________________________________________________________
conv2d_585 (Conv2D)          (None, 16, 18, 128)       73856
_________________________________________________________________
conv2d_586 (Conv2D)          (None, 14, 16, 128)       147584
_________________________________________________________________
conv2d_587 (Conv2D)          (None, 6, 7, 128)         147584
_________________________________________________________________
conv2d_588 (Conv2D)          (None, 4, 5, 128)         147584
_________________________________________________________________
flatten_16 (Flatten)         (None, 2560)              0
_________________________________________________________________
dense_48 (Dense)             (None, 1000)              2561000
_________________________________________________________________
dropout_16 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_49 (Dense)             (None, 300)               300300
_________________________________________________________________
dense_50 (Dense)             (None, 14)                4214
=================================================================
Total params: 3,529,846
Trainable params: 3,529,846
Non-trainable params: 0
_________________________________________________________________
```

References

[1] Hans-Dieter Block. "The perceptron: A model for brain functioning. i". In: *Reviews of Modern Physics* 34.1 (1962), p. 123.

[2] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[3] Thibaut Durand, Nazanin Mehrasa, and Greg Mori. "Learning a deep convnet for multi-label classification with partial labels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 647–657.

[4] Jamileh Farajzadeh, Ahmad Fakheri Fard, and Saeed Lotfi. "Modeling of monthly rainfall and runoff of Urmia lake basin using "feed-forward neural network" and "time series analysis" model". In: *Water Resources and Industry* 7 (2014), pp. 38–48.

[5] Yoav Freund and Robert E Schapire. "Large margin classification using the perceptron algorithm". In: *Machine learning* 37.3 (1999), pp. 277–296.

[6] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[7] Isabelle Guyon and André Elisseeff. "An introduction to feature extraction". In: *Feature extraction*. Springer, 2006, pp. 1–25.

[8] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Yaoshiang Ho and Samuel Wookey. "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling". In: *IEEE Access* 8 (2019), pp. 4806–4813.

[10] Robert A Jacobs. "Increased rates of convergence through learning rate adaptation". In: *Neural networks* 1.4 (1988), pp. 295–307.

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: `http://arxiv.org/abs/1412.6980`.

[13] Magnus Knutsson and Linus Lindahl. *A COMPARATIVE STUDY OF FFN AND CNN WITHIN IMAGE RECOGNITION: The effects of training and accuracy of different artificial neural network designs*. 2019.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[15] Steve Lawrence et al. "Face recognition: A convolutional neural-network approach". In: *IEEE transactions on neural networks* 8.1 (1997), pp. 98–113.

[16] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[17] Donghun Lee and Kwanho Kim. "Recurrent neural network-based hourly prediction of photovoltaic power output using meteorological information". In: *Energies* 12.2 (2019), p. 215.

[18] Haoxiang Li et al. "A convolutional neural network cascade for face detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 5325–5334.

[19] Weiyang Liu et al. "Large-margin softmax loss for convolutional neural networks." In: *ICML.* Vol. 2. 3. 2016, p. 7.

[20] David Lowe and Michael Tipping. "Feed-forward neural networks and topographic mappings for exploratory data analysis". In: *Neural Computing & Applications* 4.2 (1996), pp. 83–95.

[21] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[22] Mario Mustra, Kresimir Delac, and Mislav Grgic. "Overview of the DICOM standard". In: *2008 50th International Symposium ELMAR.* Vol. 1. IEEE. 2008, pp. 39–44.

[23] Ha Q Nguyen et al. "VinDr-CXR: An open dataset of chest X-rays with radiologist's annotations". In: *arXiv preprint arXiv:2012.15029* (2020).

[24] Albert B Novikoff. *On convergence proofs for perceptrons.* Tech. rep. STANFORD RESEARCH INST MENLO PARK CA, 1963.

[25] M Omid, A Baharlooei, and H Ahmadi. "Modeling drying kinetics of pistachio nuts with multilayer feed-forward neural network". In: *Drying Technology* 27.10 (2009), pp. 1069–1077.

[26] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[27] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[29] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[30] Ahmed Ali Mohammed Al-Saffar, Hai Tao, and Mohammed Ahmed Talab. “Review of deep convolution neural network in image classification”. In: *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*. IEEE. 2017, pp. 26–31.

[31] Haşim Sak, Andrew Senior, and Françoise Beaufays. “Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition”. In: *arXiv preprint arXiv:1402.1128* (2014).

[32] I Satia et al. “Assessing the accuracy and certainty in interpreting chest X-rays in the medical division”. In: *Clinical medicine* 13.4 (2013), p. 349.

[33] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[34] Hyunjune Sebastian Seung, Haim Sompolinsky, and Naftali Tishby. “Statistical mechanics of learning from examples”. In: *Physical review A* 45.8 (1992), p. 6056.

[35] Patrice Y Simard, David Steinkraus, John C Platt, et al. “Best practices for convolutional neural networks applied to visual document analysis.” In: *Icdar*. Vol. 3. 2003. Citeseer. 2003.

[36] Animesh Singh et al. “A Genetic Algorithm based Kernel-size Selection Approach for a Multi-column Convolutional Neural Network”. In: *arXiv preprint arXiv:1912.12405* (2019).

[37] Leslie N Smith. “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 464–472.

[38] Bolan Su and Shijian Lu. “Accurate scene text recognition based on recurrent neural network”. In: *Asian Conference on Computer Vision*. Springer. 2014, pp. 35–48.

[39] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. “Introduction to multi-layer feed-forward neural networks”. In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[40] Ferhan Ture and Oliver Jojic. “No Need to Pay Attention: Simple Recurrent Neural Networks Work!(for Answering” Simple” Questions)”. In: *arXiv preprint arXiv:1606.05029* (2016).

[41] Alex Waibel et al. “Phoneme recognition using time-delay neural networks”. In: *IEEE transactions on acoustics, speech, and signal processing* 37.3 (1989), pp. 328–339.

[42] Jiang Wang et al. “Cnn-rnn: A unified framework for multi-label image classification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2285–2294.

[43] Thomas Wiatowski and Helmut Bölcskei. “A mathematical theory of deep convolutional neural networks for feature extraction”. In: *IEEE Transactions on Information Theory* 64.3 (2017), pp. 1845–1866.

[44] Yudong Zhang et al. “Fruit classification using computer vision and feed-forward neural network”. In: *Journal of Food Engineering* 143 (2014), pp. 167–177.

[45] Jure Zupan and Johann Gasteiger. *Neural networks for chemists: an introduction.* John Wiley & Sons, Inc., 1993.