Handling Overfitting in Deep Learning Models and Their Applications

Isaac F. Bensaid BSc

February 15, 2022

Contents

0.1 Introduction

Given the uncertainty of the future, decision-making is a difficult task. If we can accurately predict the future, decision-making becomes easier. While making predictions might seem easy, making accurate ones is hard. When Data Scientists attempt to predict the future, many obstacles can arise. One obstacle is Overfitting, a common problem in Machine Learning and Deep Learning. Data Scientists often rely on historical data to predict the future. For instance, assume we would like to predict the outcome of the next presidential election. There is a tremendous amount of historical data about presidential elections. The question becomes, how do we select Hyperparameters – the data that could produce the most accurate prediction results – from a large pool of historical elections data. When Data Scientists do not choose their data wisely, Overfitting could undermine their predictions' accuracy. In some cases, a high number of Hyperparameters could increase the complexity of the Learning Algorithm and increase the likelihood of Overfitting[37]. Historical data helps Data Scientists train Learning Algorithms – also referred to as Machine/Deep Learning Models. Machine/Deep Learning Model can make accurate predictions without being explicitly programmed. Data Scientists train Machine/Deep Learning Models using a set of chosen training data: a collection of situations for which the desired output is known. For example, a collection of election data in which Data Scientists know the election outcome beforehand. The goal is that the Model will also perform well on predicting the output when fed validation data – data that the Model did not encounter during its training and its output was unknown. The ability to perform well on validation datasets is called Generalization [21]. Overfitting occurs when a model performs well on the training dataset and underperforms on the validation dataset [37]. We can explain Overfitting intuitively because all data – related to the mentioned election outcome prediction example – can be divided into two groups. The first group has data helpful for accurate election outcome prediction (also known as Hyperparameters). The second group has useless data for accurate election outcome prediction (also known as noise[37]). Machine/Deep Learning Models tend to overfit by memorizing properties of noise data that do not serve them well during the prediction phases. The higher the uncertainty and complexity for any prediction task, the more noise in its historical data. We may reduce Overfitting if we successfully determine which historical data to ignore [21]. One of the most prominent qualities in Deep Learning/ Machine Learning is how well a model performs on unseen data. Making accurate predictions on unseen data is a fundamental element of any DL/ML Model [8]. Many Data Scientists have been attempting to reduce Overfitting to improve prediction accuracy [36] [12]. However, up until today, there is no standard solution for Overfitting. No single technique will work perfectly for every problem. Each problem required a unique solution which may include more than one technique [40]. For example, Biologists have relied on Machine Learning to prevent or cure diseases [50]. In gene therapy — the practice of inserting corrective genetic material into defective cells — Biologists have used Machine Learning methods to develop or improve non-pathogenic viruses. Physicians use Non-pathogenic viruses to deliver corrective genetic materials to defective cells [9]. These viruses are called vectors. Biologists

believe they do not cause any human diseases because they have been disabled of any pathogenic effects [35]. During the delivery process, many hurdles can arise. For example, the immune system can destroy the delivery virus vector, preventing the delivery of the corrective genetic material. Overfitting could arise when Data Scientists predict which potential vectors are capable of evading the immune system — to deliver the corrective genes to the targeted cells. However, this thesis will focus on Overfitting in the medical diagnosis of neurodegenerative diseases. Suppose we would like to create a Model for neurodegenerative disease medical diagnosis. To create the Model, we need to train it with a medical records dataset. The complexity and uncertainty of this Model are likely to be high, increasing the likelihood of Overfitting. If we do not address Overfitting, it can arise and discredits the accuracy of our Model – which could result in unwanted prediction results. For example, Parkinson's disease (PD) is a common progressive neurodegenerative disease — characterized by motor and non-motor symptoms. While some diseases can be diagnosed with a lab test — such as cholesterol level and blood pressure measurements — Parkinson's disease cannot. Physicians rely on medical history and physical examination to diagnose Parkinson's disease. Physicians look for classic motor symptoms: resting tremor, stiffness, and slowness of movement. Additionally, vocal defections are a common symptom in the early stages of Parkinson's disease [13]. citegunduz2019deep trained Convolutional Neural Networks to recognize patients with such vocal defections using sets of vocal features. However, Overfitting prevented accurate classification between healthy individuals and PD patients. Luckily, they used two methods to reduce Overfitting — L2 regulations and Dropout (more on those methods later).

0.1.1 Problem Statement

Humans can overgeneralize sometimes. For example, assume we are conducting business with an American businessman for the first time. If we find the American businessman unethical, we might inevitably claim that all American businessmen are unethical. In machine learning, a similar issue called **Overfitting** can occur if the Model does not generalize well. **Overfitting** could occur because the Model learns the noise in the training set, which impacts its performance on unseen data. The noise picked up in the training set may not apply to unseen data, leading to inaccurate predictions. Linear Modules tend to be less likely to overfit. The more complex the Model, the more it will be prone to Overfitting. Neural network models contain multiple non-linear hidden layers, making them complex models to learn complicated relationships. Unfortunately, **Overfitting** is a common problem that prevents Neural Networks from making accurate predictions. Overfitting occurs when Convolutional Neural Networks (CNN) fails to make accurate predictions on unseen datasets. We will focus on mitigating Overfitting in CNN modules. This problem is still considered a standing problem in ML/DL[36]. Therefore, [36] imposed Regularization on the fully connected Layer(FCL) of the convolutional neural network. In a typical CNN, weights of FCLs make up most of the parameters of the network. So, first, [36] sparsified connections to the FCLs by applying L1 Regularization on the weights of the FCLs, then clipped small weights to zeros. Next, [36] also applied L2 Regular-

ization on all weights of the CNN to keep the weights around zero. [36] found that the numerous parameters of FCLs are a contributor to Overfitting. The number of parameters in CNN is vast. The high number of features makes the model complex, which tends to result in **Overfitting**. In biotechnology, therapeutics should meet high safety standards during their development. Meeting safety standards requires an accurate prediction of any adverse side effects. For example, in gene therapy, predicting the safety level of Viral Vectors could be a challenge due to Overfitting. Several individuals developed cancer during a gene therapy clinical trials using viral vectors to deliver corrective genes. In a study done by [53], it was demonstrated that Genotoxic Vectors have a unique gene expression signature. Based on this finding, [53] developed a "surrogate assay for genotoxicity assessment" (SAGA). SAGA uses this unique gene expression signature to distinguish genotoxic vectors from safe vectors, using machine learning to recognize the said gene expression signature. SAGA achieved an accuracy of 90.91% on a dataset of vectors with known genotoxic potential. This level of accuracy is arguably high, but given that gene therapy could lead to cancer, methods to identify safe gene therapy vectors require a higher degree of accuracy.

0.1.2 Convolutional Neural Networks

Convolutional Neural Networks or CNNs are a kind of neural network. CNNs focus on data that has a gird-like representation. For example, images that we represent as a 2-D grid of pixels. CNNs employ a linear mathematical operation known as convolution. Convolution is defined as the integral of the product of two real-valued functions after one is reversed and shifted. Let f and g be two real-valued functions, the convolution of the two functions is written as $f * g$, denoting the convolution operator with the symbol "$*$" [20][5].

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)\, d\tau$$

We could utilize several types of convolutions in ML and DL. Depending on the convolution operation, the input will be translated into some form, depending on the convolution operation used. Convolution can be detrimental to input data. For example, if we use a CNN to detect traffic lights, the bright color location is crucial. Changes to the location of the bright color can be an adverse impact caused by convolution. Pooling helps preserve the bright color in the image after the convolution layer's transformations. Pooling is a technique used to limit translation on the input data. CNN architectures can vary based on their specific application. Typically, CNNs consist of an input and output layer and several hidden layers in between. There is no one conventional CNN architecture. The type of architecture used in a CNN depends on the problem and its data. For example, It could be the case that a particular benchmark has a specific CNN architecture that is known to be the best. As scientists conduct further research, this architecture will eventually evolve[21]. CNN can be used in the medical field to help doctors diagnose patients. In [47], CNN was used to assess neurological damage in Parkinson's disease patients using

3D brain images. [47] used two CNN architectures differing on the output layer shape, the activation function, and the loss function. When using the same dataset of 508 3D brain images, it was evident that one of the different architectures outperformed the other by achieving higher classification accuracy of Parkinson's disease patients. The difference in prediction accuracy shows how modification in CNN architecture can improve its performance.

0.1.3 What is Overfitting in ML/DL

To demonstrate Overfitting, we used a dataset that contains medical records of Parkinson's disease patients. We programmed this experiment in Keras (Python interface for artificial neural networks [24]). The dataset we used contains two sub-directories. The first sub-directory has a collection of wave drawings of healthy patients, while the second sub-directory has a collection of wave drawings of patients with Parkinson's disease. The dataset is available [48] here.

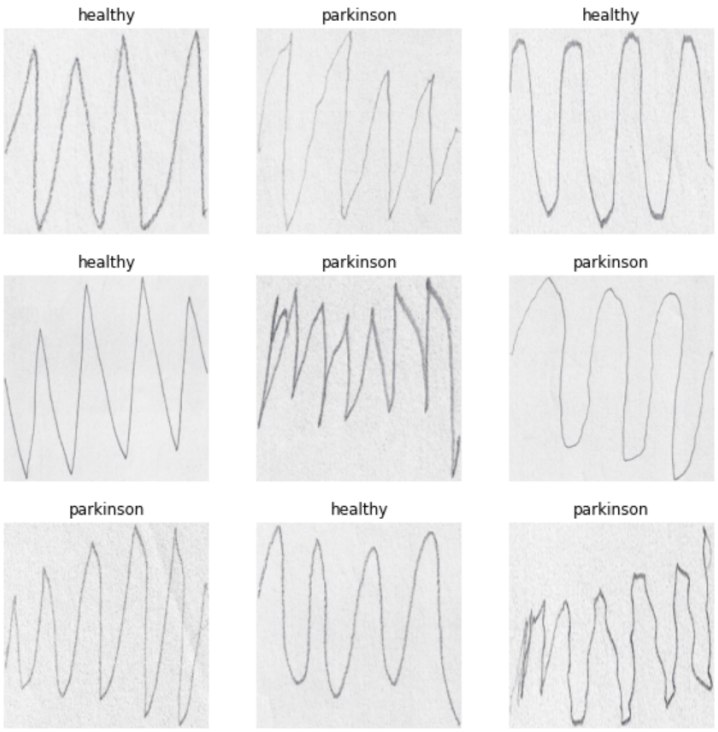


Figure 1: Waves drawings of healthy patients and patients with Parkinson's disease.

Figure 1 shows the waves drawings from the Parkinson's disease dataset. We randomly picked images from both sub-directories. Motor symptoms such as speed of drawing and pen pressure make detecting Parkinson's disease possible. The dataset consists of 102 images. We used 72 images to train a CNN model and 30 images to test it. Due to the small size of the dataset, the CNN model is highly prone to Overfitting. Therefore, data Augmentation — creating more data — is needed in this situation to reduce Overfitting.

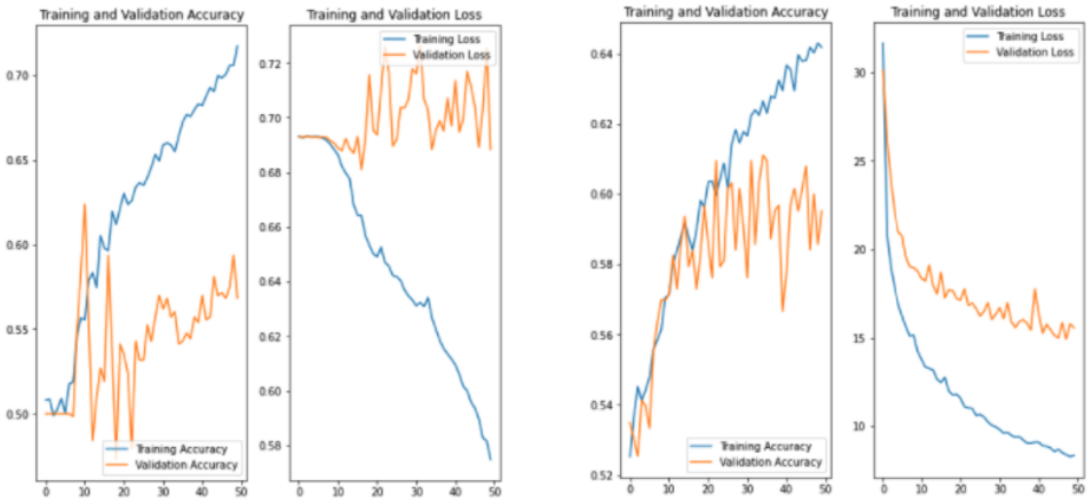
Experiment Results To Reduce Overfitting Using Parkinson's Disease Dataset

We built a CNN model with four layers to classify the Parkinson's Disease data into the sub-directories mentioned above: (Health patients, patients with Parkinson's disease). Figure 2 shows the specification of the CNN Model.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1 (Conv2D)              (None, 128, 128, 128)     3328

 max_pooling2d_12 (MaxPoolin  (None, 40, 40, 128)      0
 g2D)

 conv2 (Conv2D)              (None, 40, 40, 64)        204864

 max_pooling2d_13 (MaxPoolin  (None, 12, 12, 64)       0
 g2D)

 conv3 (Conv2D)              (None, 12, 12, 32)        18464

 max_pooling2d_14 (MaxPoolin  (None, 4, 4, 32)         0
 g2D)

 conv4 (Conv2D)              (None, 4, 4, 32)          9248

 max_pooling2d_15 (MaxPoolin  (None, 1, 1, 32)         0
 g2D)

 flatten_3 (Flatten)         (None, 32)                0

 fc1 (Dense)                 (None, 64)                2112

 fc3 (Dense)                 (None, 2)                 130

=================================================================
Total params: 238,146
Trainable params: 238,146
Non-trainable params: 0
_________________________________________________________________
```

Figure 2 CNN Configuration

As shown in figure 2, images were rescaled to 128 by 128 for better resolution, and we used four CNN layers. One of size 128, one of size 64, and two of size 32. To reduce Overfitting and improve the Model's accuracy, we used the following overfitting techniques: Dropout Regularization, Weight Constraints, Weight Initialization, L2 Regularization. Due to the dataset's limited size, we generated an additional 5110 images using data augmentation. We rotated the original image by 360 degrees, and we flipped it virtually and horizontally to produce other images from the original images in the dataset.

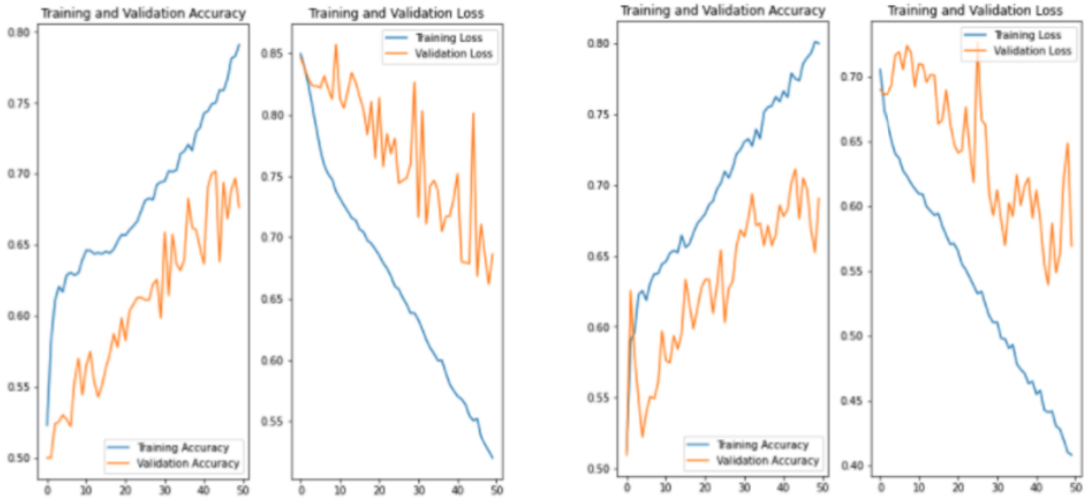


(a) Dropout: 57.41 Accuracy

(b) Weight Constraints: 60.71 Accuracy

Figure 3

Figure 3 shows the model performance using Weight Constraints and Dropout Regularization. The graph on the left in section (a) shows the training accuracy in blue and the validation accuracy in orange. The graph on the right in section (b) shows the training loss in blue and validation loss in orange. The same is true in section (b).



(a) L2 Regularization: 68.37 Accuracy

(b) Weight Initialization: 69.57 Accuracy

Figure 4

Figure 4 shows the model performance using L2 Regularization and Weight Initialization. The graph on the left in section (a) shows the training accuracy in blue and the validation accuracy in orange. The graph on the right in section (b) shows the training loss in blue and validation loss in orange. The same is true in section (b).

Method	Accuracy
Dropout Regularization	57.41 %
L2 regularization	68.37 %
Weight Constraints	60.71 %
Weight Initialization	69.57 %

Table 1

As we can see from table 1, figure 3, and figure 4 above. The module performed better when using Weight Initialization and L2 Regularization than Weight Constraints and Dropout Regularization.

Experiments Discussion

Overfitting takes place when the margin (gap) between the training error and test error is large [21]. The models have overfitted the data, as shown in the plots above. Training accuracy and validation accuracy are off by a large margin—the lower the loss, the higher the accuracy. As we can see in some of the graphs above, the training set has a low loss, which yields higher accuracy, but the validation set has a high loss which yields low accuracy. A high difference between training and validation loss is a sign of Overfitting. In other words, the models performed well in the training set but performed poorly in the validation set, resulting in Overfitting. While we tried to reduce Overfitting by using a few methods. Such as Dropout Regularization, Weight Constraints, Weight Initialization, L2 Regularization. Based on the figures above, there is still room for improvement. We saw a slight reduction in Overfitting after applying Weight Initialization and L2 Regularization on the Parkinson's Disease dataset because the training and validation accuracy are closely aligned. Due to Random Access Memory and GPU limitations, we could not produce a more significant number of augmented data and train the models for a longer time. If we train the Model with more data and longer time, there may be room for improvement, with a higher RAM and GPU.

0.2 Methods

Suppose we would like to create a model to detect objects. To create the Model, we need to train it with a dataset that contains images of the object of interest. Depending on the complexity of the Model, Overfitting can arise and deters the Model from correctly detecting the object of interest. There has been many research and development regarding Overfitting to improve the detection accuracy of such models. Several methods that help to reduce Overfitting have been developed [21][37], such as Regularization, early stoping, Dropout, Expanding the training set as well as the methods mentioned in this section. Additionally, we can be interested in understanding how learning, cognition, and creative behaviors emerge [45]. To conduct an artificial intelligence-driven study, we may use microscopic brain images, such as (Human brain MRI, Electrocorticography, Gene Atlas, and Tracer Injection). Any model

we create using the said data will be complex and depend on several hyperparameters. The proliferation of hyperparameters will most likely lead to Overfitting.

0.2.1 Regularization

Generally, any element added to the prediction model to compensate for data-scarce is referred to as regularization [40]. Regularization is one form of regression that discourages learning a more complex model to avoid the risk of Overfitting. We will start with a simple introduction to regression and more deep learning-related regression methods to motivate Regularization. The main objective of regression is to calculate the prediction error and minimize it. Predicting error and minimizing it can happen using several strategies. We can do so by altering the Model's capacity: A model's capacity can fit a wide variety of functions. Models with high capacity can learn from complex data and produce accurate results. It was concluded in [26] that increasing CNN models' capacity helped its units better adapt and specialize to its tasks [21]. For example, in a linear regression problem, our hypothesis space or capacity is the set of all linear functions. We can increase our capacity by adding the set of all polynomials to our hypothesis space. This increase will most certainly lead to Overfitting because our Model will have many functions to fit our data. The right size of capacity depends chiefly on the problem's complexity. Simple regression has a single explanatory variable [3]. It means that we will use a single variable to predict another variable. For example, If we merely used age to predict the likelihood of Parkinson's Disease. In reality, any effort to make predictions using a single variable without attention to the other factors could result in statistical difficulties (termed "omitted variables bias"). For example, we say that age increases the likelihood of Parkinson's Disease without looking into other factors.

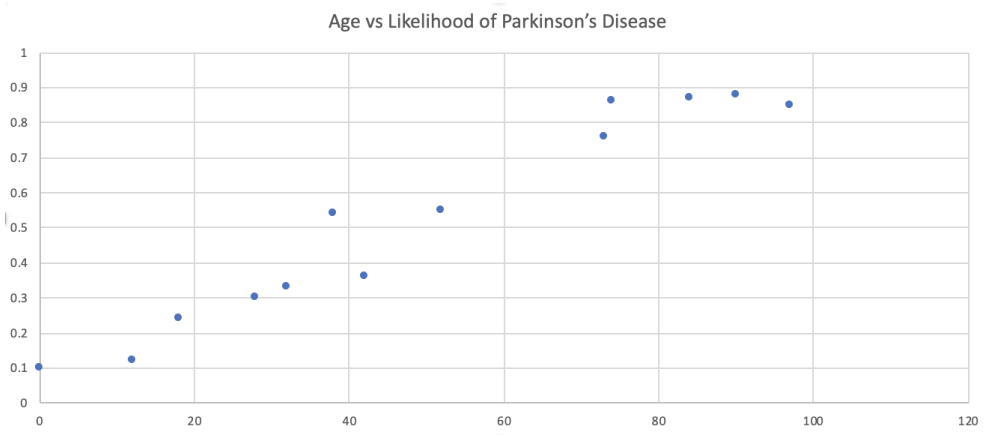


Figure 5: This data generated randomly for illustration.

Take figure 5, for example. We can see that sometimes older people tend to develop Parkinson's disease. However, the relationship is not perfect. It seems that age does not suffice for an accurate prediction of the likelihood of Parkinson's disease.

Therefore, we may deduce that other factors could influence the likelihood of Parkinson's disease; we may refer to them as "noise". Nevertheless, we can write the above relation mathematically as follows:

$$CI = A\beta + \epsilon,$$

Where β is the effect of an additional year of age on the likelihood of Parkinson's disease and ϵ is the "noise". CI is the dependent variable; A is the independent variable. The task of regression is to produce an estimate of β, based on the data provided and taken ϵ into account, meaning that we need to calculate the error ϵ and minimize it. One way to calculate the error is to calculate the difference between the actual and predicted values and average them. However, this might be problematic since there is a chance that the averaged variable will cancel each other out.

For this reason, we take the sum of the Square, formally known as Residual Sum of Squares (RSS)[4]:

$$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2,$$

Since this is a relatively simple example, we can shift our attention to more deep learning methods to minimize error. We will start with Dropout and move to Data augmentation, Ridge Regularization, Bayesian Regularization, and Early Stopping.

Dropout

Combining two different models help with reducing Overfitting via averaging their prediction. Unfortunately, such a process can be prolonged. Dropout helps to finesse this situation. Dropout is a simple method that prevents Neural Networks from Overfitting; when we say Dropout, we mean dropping units from a neural network, temporarily along with their associated connections, as shown in Figure 6.

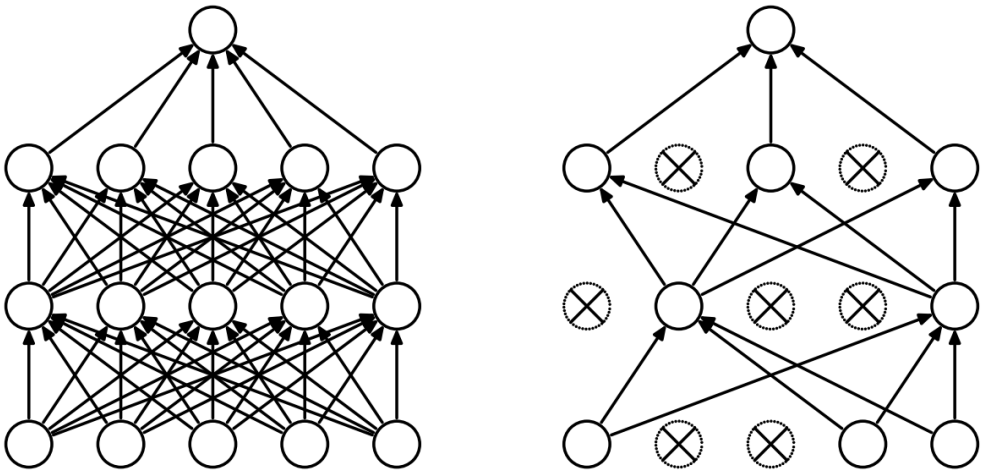


Figure 6: left diagram shows the neural network before applying Dropout [14].

The choice of which units to drop is random. Units that survived Dropout will constitute a "thinned" neural network. The left diagram in Figure 6 shows a thinned neural network. In [14][21] it was shown that classification using Dropout is better than 1000 sub-networks using Monte Carlo approximation. In [14] it was also shown that it is better than L1 and L2 Regularization. The computation and memory complexities of Dropout are ideal at O(n) – which is a function that increases linearly to reflect the time and space complexity of Dropout [40]. A linear complexity – as if the running time of a given algorithm increases linearly with the algorithm input – is highly preferable over higher-order functions/time complexities. For example, some algorithms have a quadratic time complexity typically denoted as $O(n^2)$. While Dropout ameliorates Overfitting, it also helps to combine enormous neural network architectures sufficiently [14].

Training with More Data - Data augmentation

The quantity and quality of the training dataset are a big factor in the Model's overall performance and accuracy. Expanding the training dataset will help improve the performance and accuracy of the Model. If we add noisy data, this technique will not work. However, training with more data can help improve the accuracy of the Model, especially in complicated models [37]. There is, however, a downside to this method. The high amount of data will increase the training time. Training data could be expensive to obtain. In the experiment done in [12] to combat overfitting in the ImageNet dataset – a dataset with 15 million labeled images mapped to 22,000 categories – [12] used data augmentation. First, [12] used "label-preserving transformation," which added more data to the training set. [12] extracted random 224×224 patches from the 256×256 images and trained the CNN merely on the extracted patches. The enlarging of the training set allowed for a more complex model without suffering from substantial Overfitting. Second, [12] performed Principal Component Analysis (PCA) – the method used to reduce the dimensionality of large datasets while preserving most of its information [16] – on the set of RGB (stands for Red Green Blue: combining these colors can produce images on screens) pixel values through the training set and alter the intensities of the RGB channels in training images. To each training image, [12] adds multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore to each RGB image pixel. $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ [12] add the following quantity:

$$[P_1, P_2, P_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where α_i and λ_i are ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the aforementioned random variable [12]. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is redrawn. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1 %.

Weight Decay known as (L2 or ridge regularization)

We only mentioned one way to control a model's capacity. Instead of excluding or including more functions in our hypothesis space, we can also alter our capacity by expressing a preference for a specific function in the hypothesis space. This can be done by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J [21] [27]:

$$\hat{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \ ,$$

Where α is a hyperparameter that weights the relative contribution of the norm penalty term Ω relative to the standard objective function $J(x, \theta)$. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization [21] [27]. As was shown in [27] picking different parameter norms Ω leads to different preferred solutions that suits the problem in question. In [27]:

$$\hat{J}(\omega; X, y) = \frac{\alpha_1}{2}\omega^T\omega + \alpha_2|\omega|_1 + J(\omega; X, y) \ ,$$

was used as objective function J. With $\frac{\alpha_1}{2}\omega^T\omega + \alpha_2|\omega|_1$ as a parameter norm penalty they were able to reduce overfitting and produce a state of the art results for denoising autoencoders: a special type of neural network which takes corrupted data as input and predict the uncorrupted data. But in ridge regularization to reduce variance and generalization error – the proportion of incorrect output – in a DL model, researchers added:

$$\Omega(\theta) = \frac{1}{2}\|w\|_2^2 \ ,$$

as parameter norm penalty to the objective function. This term is known as weight decay – also referred to as Tikhonov's regularization – that could also be added to the cost function in order to restrict the model's parameters from increasing and/or penalizes the parameters of the DL model which results in a smaller squared L2 norm weight [51] [1].

Bayesian Regularization

Bayesian Regularization imposes prior distributions – distributions that express a prior opinion before taking any facts into account – on a model's parameters and penalizes its weights [18]. However, typical neural network models use the mean square errors (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - a_i)^2,$$

as an objective function and don't optimize the weights of the model [43]. In typical neural networks models optimal configuration of its weights is derived by minimizing MSE. Bayesian regularization neural networks apply a probability distribution to the weights of the Model. The objective function in Bayesian regularization includes network weights. It is written as [43]:

$$F(\omega) = \alpha E_\omega + \beta E_D,$$

Where E_D is MSE. β and α are objective parameters. E_ω is the sum of squared network weights also known as weight decay:

$$E_\omega = \frac{1}{n}\sum_{i=1}^{n}(w_j)^2.$$

A gradient-based based optimization method is used to minimize the function $F(\omega)$ and optimize the hyperparameters β and α. Here is an overview of this method. For more details see [11][2]. Given a M as a neural network model, ω as vector weights and N as total number of network weights. $P(\omega|, \beta, M)$ is the prior density. $P(D|\omega, \beta, M)$ is the likelihood function. $Z_D(\beta)$, $Z_\omega(\beta)$ and $P(D|\alpha, \beta, M)$ are the normalization factors. Based on Bayesian rule we have:

$$P(\omega|D, \alpha, \beta, M) = \frac{P(D|\omega,\beta,M)P(\omega|\alpha,M)}{P(D|\alpha,\beta,M)}$$

Since the prior distribution of network weight is viewed as a distribution of Gaussian, we have [17]:

$$P(D|\omega, \beta, M) = \frac{e^{-\beta E_D}}{Z_D(\beta)}$$

$$P(\omega|\alpha, M) = \frac{e^{-\alpha E_\omega}}{Z_\omega(\alpha)}$$

$$Z_D(\beta) = (\frac{\pi}{\beta})^{\frac{n}{2}}$$

$$Z_\omega(\alpha) = (\frac{\pi}{\alpha})^{\frac{N}{2}}$$

Now the optimal weight should maximize the posterior probability $P(\omega|D, \alpha, \beta, M)$ [11]. [41] showed that Bayesian Regularization reduced Overfitting in inverse modeling for filters using Deep Neural Network. Moreover, [41] increased the prediction accuracy of the Neural Network by using Bayesian Regularization when trying to predict coupling Matrix from Simulated S-Parameters.

L1 Regularization

As [37] summarized it, for neural networks, the objective is to find a perfect set of weights and biases. Finding an ideal set of weights and biases becomes complicated when the number of hyperparameters increases. To find an ideal set of weights and biases, the Model needs sufficient data for learning. The amount of data needs to be proportional to the number of hyperparameters. The Model's complexity increases as the number of hyperparameters increases. Even though some do not affect its accuracy, overfitting models consider all hyperparameters. So, we need to minimize the weights of hyperparameters that do not affect the Model's accuracy. Because we do not know which hyperparameters affect the Model's accuracy, L1 Regularization tries to limit them all by minimizing the cost function of the Model. L1 Regularization adds a "penalty term" to the cost function as shown in the following formula:

$$\Omega(\omega) = \|w\|_1 = \sum_i \|w_i\| \ ,$$

Using Multiple Loss Functions

[15] used a DNN model with multiple loss functions to reduce Overfitting. [15] noted that researchers have not well explored the optimization of loss functions to prevent the overfitting problem. [15] used four kinds of the loss function in one Model, softmax loss, pairwise loss, LambdaRank top-1 loss, and LambdaRank top-2 loss. [15] explained that other loss functions might have the potential to prevent the algorithm overfitting to one softmax loss function. [15] was able to achieve a state of the art results on the CIFAR-10, MNIST, and SVHN datasets.

Early Stopping

Stopping the training early before the Model overfits the training dataset can reduce Overfitting. By monitoring the generalization error of the Model – a difference between the loss/error of a training dataset and its test dataset [30] – and training the Model multiple times with different values to select the number of epochs that produce the lowest error rate [7]. This method can be used to detect when Overfitting starts during supervised training; training is then stopped before convergence to avoid Overfitting. This method updates the Model to make it better fit the training data with each iteration. However, improving the Model's fit to the training data may increase the generalization error.

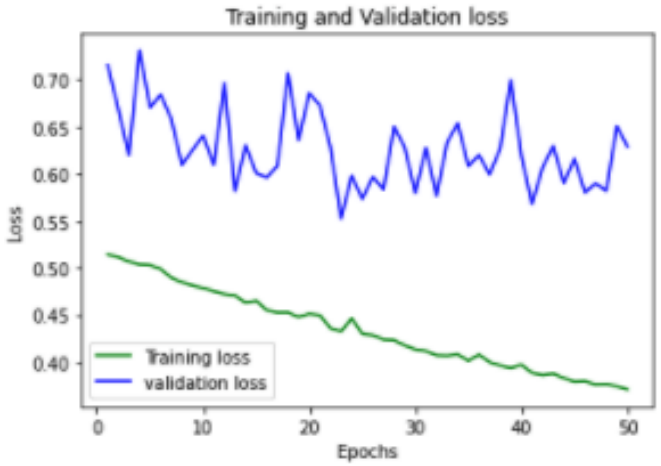


Figure 7 shows how training and validation errors decreased in Parkinson's Disease Dataset experiment from section 1.3.

When we run the Model, the training and validation errors decrease in parallel until epoch 7, then the validation error starts to increase (as we can see from figure 7). Early Stopping uses this fact to check at every iteration. If validation error is increasing, we stop the Model. In general terms, we execute the Model until the error on the validation set has plateaued; while doing so, we keep a record of the model configuration and the time the Model improved. Upon resuming training we, return to the stored configuration [21]. In [49] the model configuration was preserved using the so-called Checkpoint function in Keras – a function that saves the configuration of the Model once the early stopping algorithm halts the training. Using this approach [49] was able to achieve high classification results for diagnosing Parkinson's disease patients using MRI images. Notably, Early Stopping is equivalent to L2 Regularization, when we have a simple linear model with a quadratic error function and simple gradient descent [21]. This was also shown in [39] where early Stopping was resembled by using the quadratic function:

$$L(\Omega) = \sum_{i=1}^{n} (y_j - f(W, x_i))^2$$

as a loss function and applying gradient descent. Gradient descent is an optimization algorithm used in deep learning models to minimize their objective function $J(\theta)$ by

finding its local minimum. The algorithm starts by setting θ equal to a random value. A constant *alpha* known as learning rate is used as a small value that determines the length of the algorithm's step towards the local minimum. Then it computes the derivative of the function $J'(\theta)$ at the given values of θ. The algorithm repeats this process until it reaches a value of θ where the derivative is equal to zero. At this value of θ, the function will be at its local minimum, and the function will converge [21][19].

0.2.2 Modifying The Activation Function

This method shows the correlation between overfitting and activation functions. A modified activation function called: modified-sigmoid is based on the well-known sigmoid function. This activation function can effectively improve the accuracy of the Model and inhibit the overfitting problem [32]. Before we dig deep, let us describe activation functions and their role in neural networks.

General Neural Network Architecture

Every Neural Network contains many perceptrons. To understand how Neural Network works, we need to understand how perceptions work. Perceptrons are a function that can take any form of data as input and return a boolean value as output. A perceptron has weights, a bias, a weighted sum, input values, and an activation function.

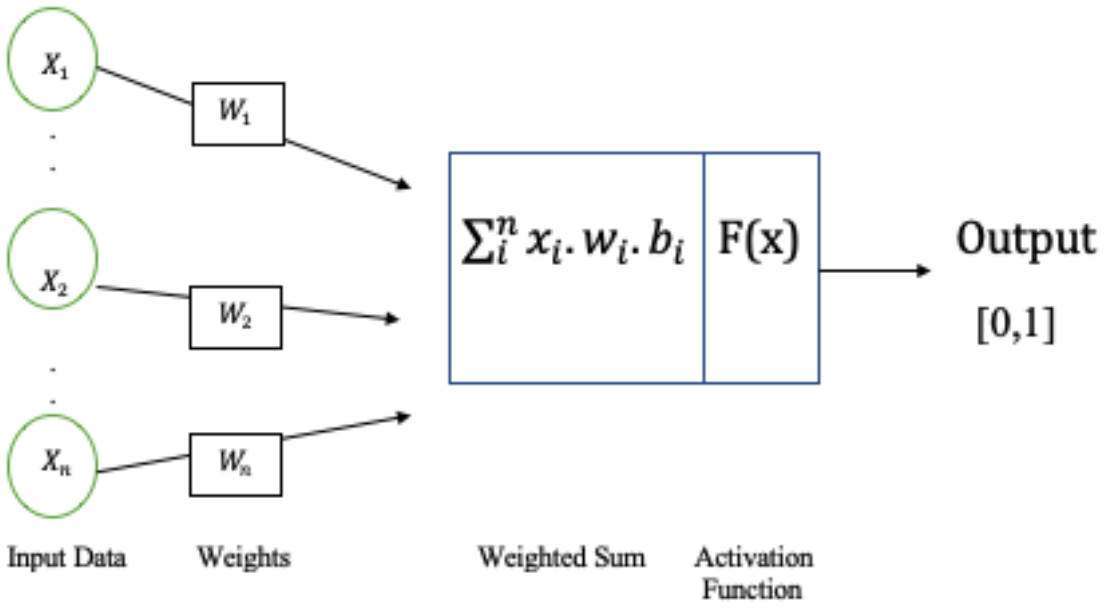


Figure 8: An image of a single perceptron. Adapted from [23]

As shown in Figure 8, all the inputs x are multiplied with their weights w, then we add all of the multiplied values and call the result the Weighted Sum. The weighted sum gets passed to the activation function.

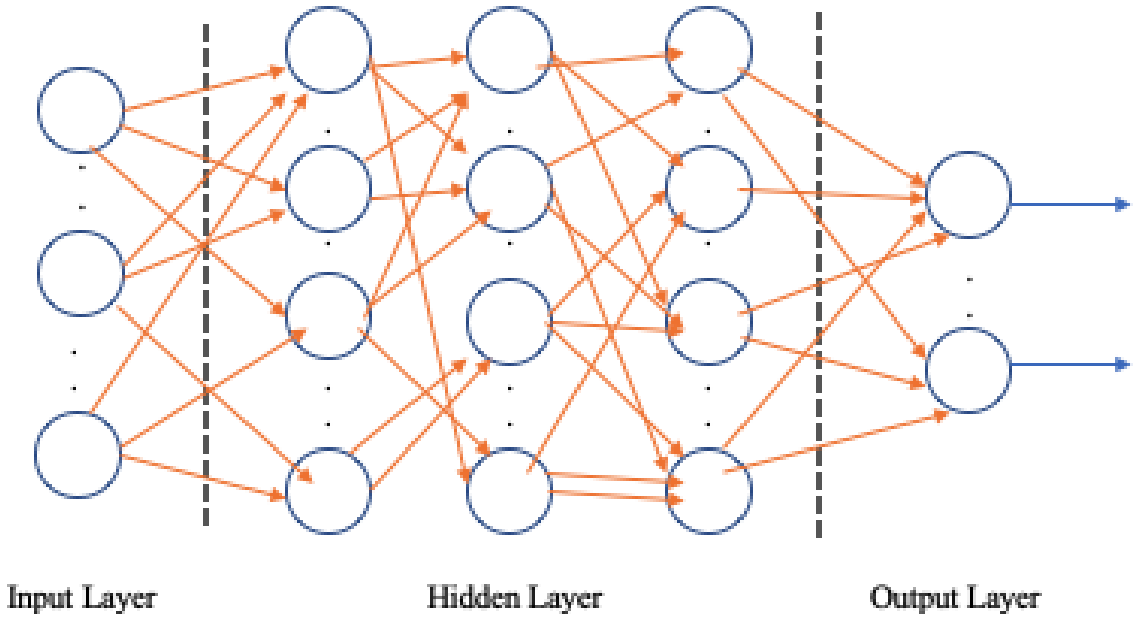


Figure 9 shows the entire Neural Network Architecture. Adapted from [23]

Every node in the Neural Network architecture shown in Figure 9 contains a perceptron. If we were to zoom closer at each node in the Neural Network Architecture, we would see the perceptron shown in Figure 11. As we can see, there are many layers to the Neural Network; the first layer is called the input layer, the last layer is called the output layer, and the layers in between are called the hidden layer. Each perceptron layer can receive input from the previous layer and produce output to the next layer. Therefore, the Neural Network Architecture can vary based on the type of Neural Network used and the problem we solve. For instance, CNN will have a slightly different architecture than what we just saw in Figure 9.

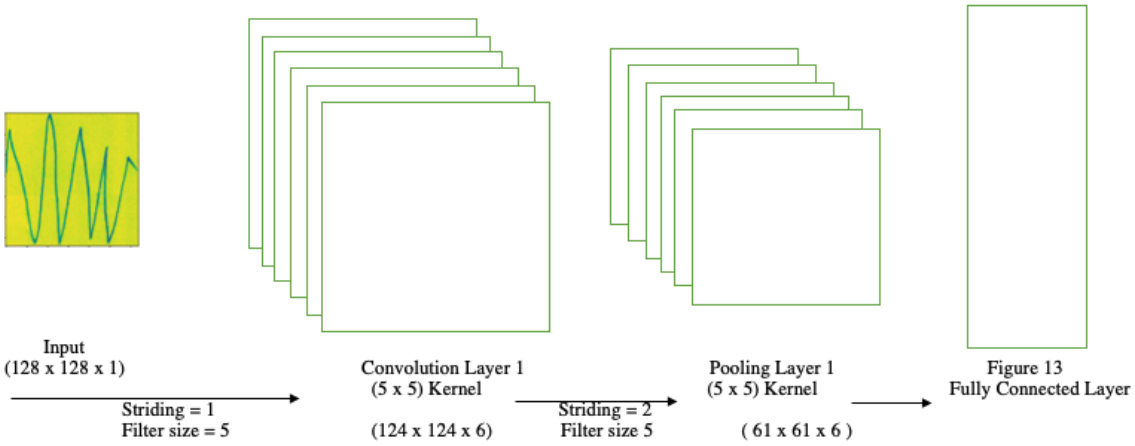


Figure 10 shows a sample of CNN Architectures. Adapted from our Parkinson's Disease example in section 1.3

In figure 10, we predicted whether a patient has Parkinson's disease from their handwritten wave. The image has dimensions height = 128 pixels, width = 128 pixels, and a depth of 1. Next, the image moves to the convolution layer shown in figure 15. Applying six filters of size 5×5 yields an image of height = 124 pixels, width = 124 pixels, and a depth of 6. Then it gets passed to the pooling layer, applying the same amount of filters and resulting in the same depth with an image of height = 61 pixels width = 61 pixels. We repeat this process until we flatten the image to a single layer, and then we feed it to the network shown in Figure 9 to classify it into the appropriate category [20].

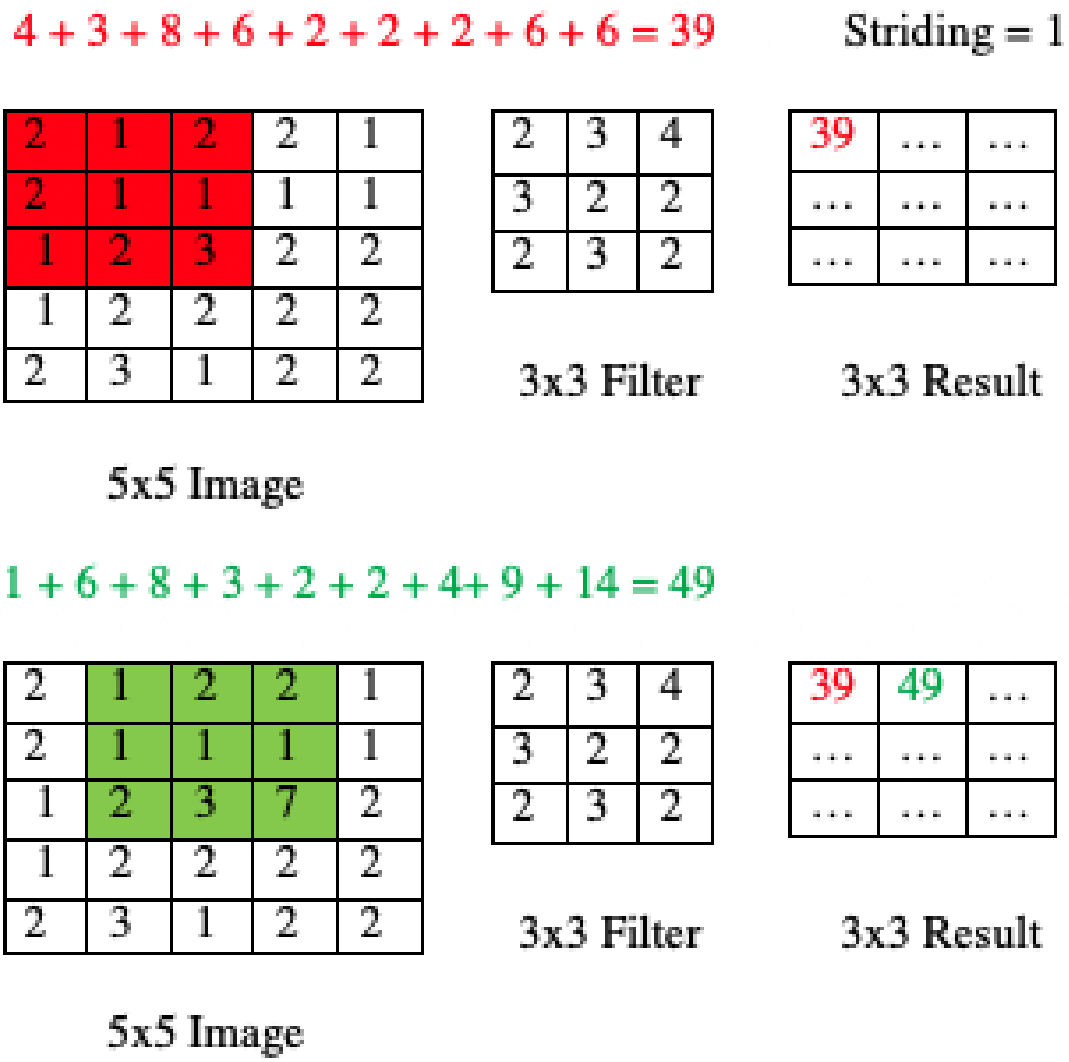


Figure 11 shows a breakdown of the convolution layer.

For simplicity, we will assume that the wave image is in 5×5 format. Applying a 3×3 filter will result in a 3×3 image. Every value in the filter is multiplied by its corresponding value in the original image. With Striding —the amount of movement

over the image — equals one, the filter moves one step to the right and repeats the same process [20] [33].

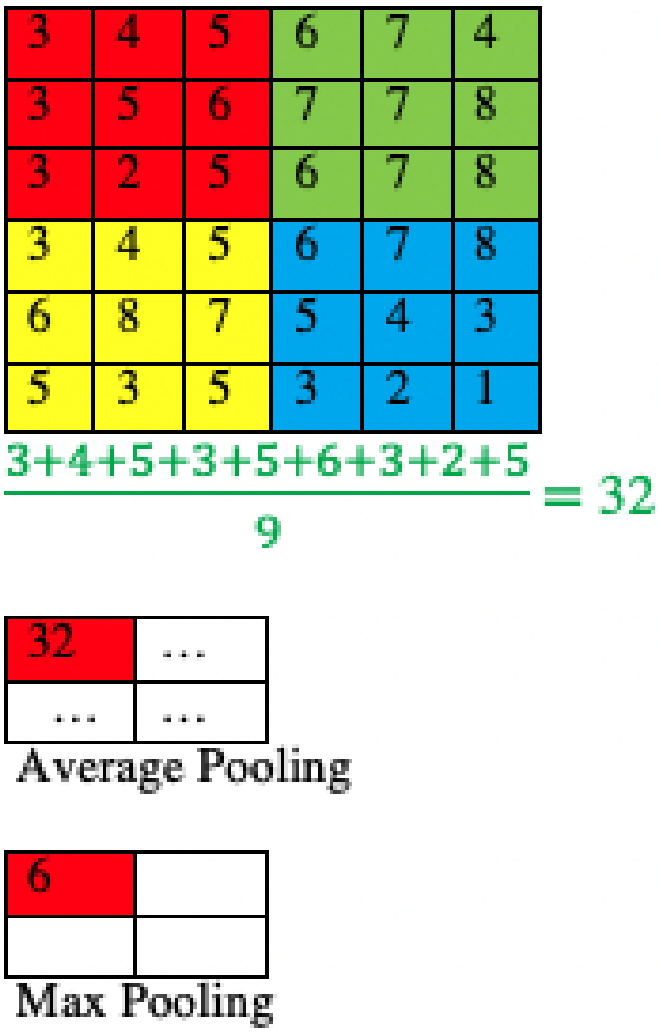


Figure 12 shows a breakdown of the pooling layer.

Here we take a look at the pooling layer. Simply put, we take a small portion of the image, and we apply either max-pooling — taking the maximum number of the portion we select – or average pooling, taking the average of the portion we select and rounding it up [20] [33]. Now that we have a basic visualization of the Neural Network Architectures, we can dig deep into activation functions.

Activation Functions

Each layer of the neural network needs a non-linear function called the activation function before generating the output signal. When there is no non-linear activation function, this neural network layer is transformed linearly. No matter how many layers the network contains, the final output can be expressed by the linear transformation of the input. Therefore, it has no difference from the single-layer neural network [32]. There are three commonly used activation functions:

Sigmoid: Sigmoid activation function:

$$Sigmoid = \frac{1}{1 + e^{-x}}$$

Tanh activation function:

$$Tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

ReLu activation function:

$$ReLu(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

The proposed method in [32] introduced a new activation function that showed promising results in terms of reducing Overfitting. The sigmoid activation function is a near S-shaped curve. The modified-sigmoid activation function is an improvement of the Sigmoid activation function. By defining a hyperparameter, the Modified-sigmoid activation function is shown as follows:

$$ModifiedSigmoid(x) = \begin{cases} x, & -w \leq x \leq w \\ Sigmoid, & x < -w \text{ or } x > w \end{cases}$$

The four different activation functions were compared by [32] using (MNIST) dataset for for training and testing the model in this proposed method.

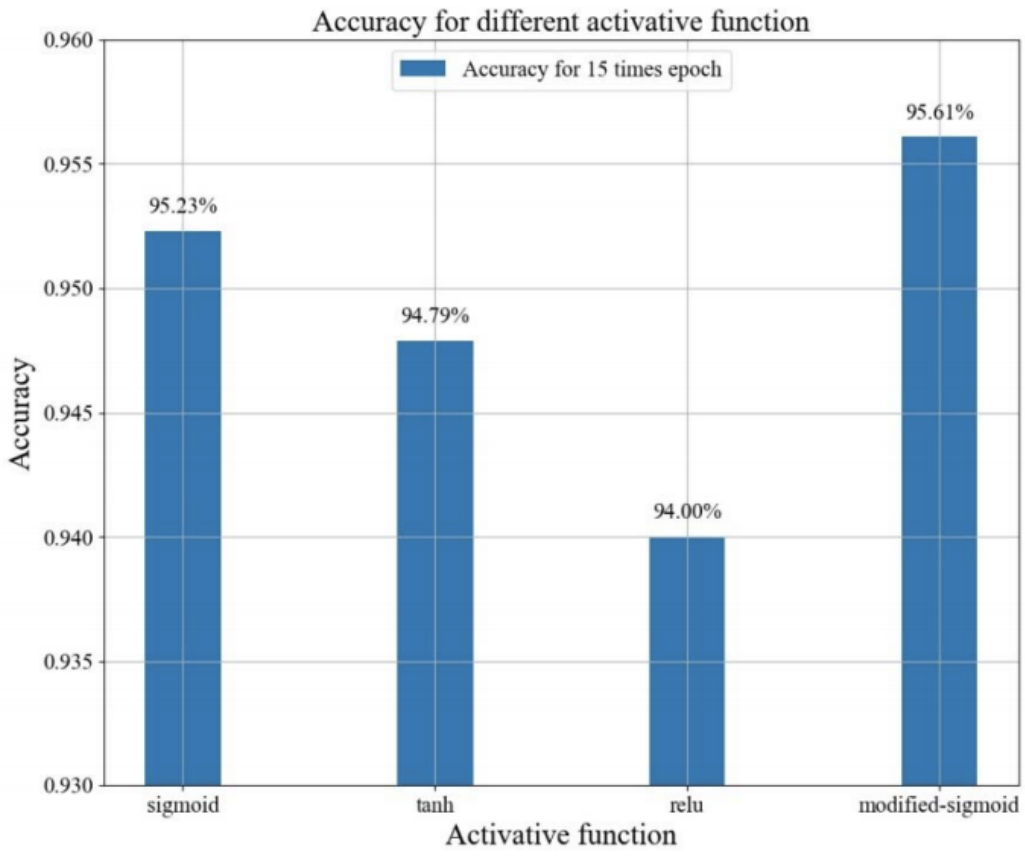


Figure 12: Effect of activation function on model accuracy [32].

As we can see in figure 12, the activation function plays a vital role in preventing the Model from Overfitting. After optimization, it was found in [32] that the neural network possesses the best test accuracy with hyperparameter $w = 2$. Furthermore, the results in [32] show that the Modified-sigmoid function can effectively improve the accuracy of the neural network and prevent the Overfitting of the neural network.

0.2.3 K-fold Cross-Validation

K refers to the number of groups that a given dataset will be split into. "Fold" refers to the number of resulting subsets. Cross-validation helps estimate how the Model is expected to perform when making predictions on unseen data. The general idea is to shuffle the dataset randomly and split it into k groups. We take each group as a test dataset and the remaining groups as the training set. We fit the Model on the training set and evaluate it on the test set. Once we retain the evaluation score, we discard the model [6]. This method reduces bias and computation time as we repeat the process only ten times when the value of k is 10. Every data point is tested precisely once and is used in training k-1 times. As we increase k, the variance of the resulting estimate is reduced as well.

0.2.4 Removing Features

Unjustifiable features that do not fit the Model should be removed. Some algorithms remove irrelevant features by default. Removing irrelevant input costs less computational power. This method improves prediction accuracy by the exclusion of irrelevant variables. The Model to be built is simpler and faster when fewer input variables are used [10].[38] experimented if deep learning can extract latent Multiple Sclerosis (MS) ¬– a neurological disease — lesion (damaged areas of the brain) features that when combined with three user-defined MRI measurements (T2w lesion volume, brain volume, and diffusely abnormal white matter DAWM) and eight user-defined clinical measurements (gender, cerebrum, optic nerve, cerebellum, brain stem, spinal cord, EDDS, and cistype) can accurately predict the chance of conversion from clinically isolated syndrome (CIS), an initial stage of MS, to clinically definite MS (CDMS). [38] used a CNN model fed with lesion masks segmented from MRI images. [38] applied the feature selection method to increase the accuracy of the Model and avoid Overfitting. [38] computed the relative importance of each of the 11 user-defined features for classification between converters and non-converters by permuting the features among the training data and computing the average generalization error. They then selected discriminative features by choosing the features whose relative importance is larger than the median importance. The importance of each feature is shown in figure 13.

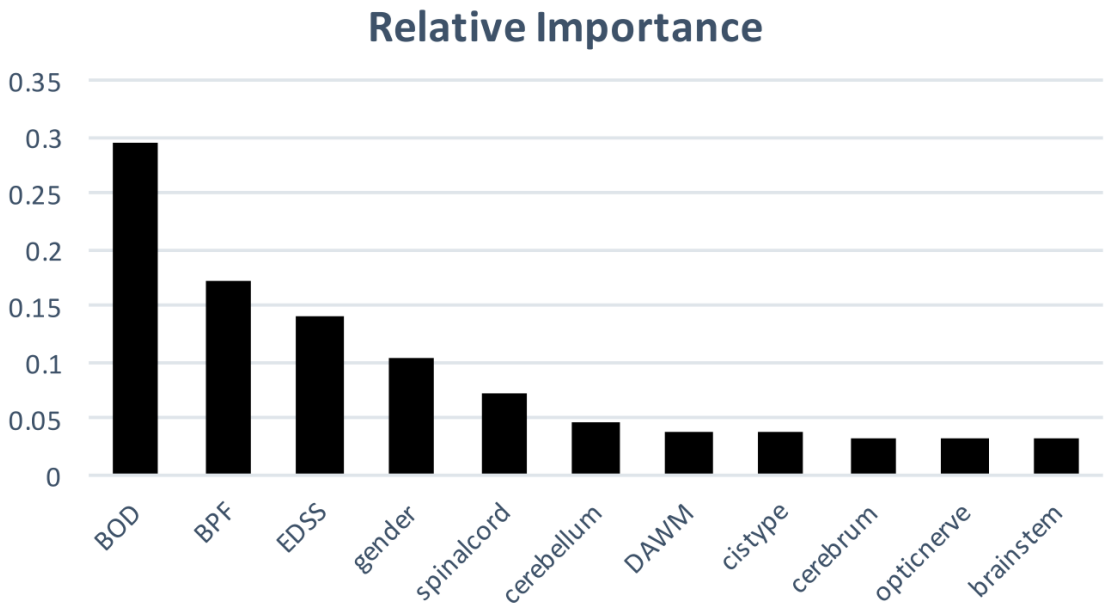


Figure 13: the importance of each feature on the Model's accuracy [38].

The selected features were BOD, BPF, EDSS, gender, spinal cord, and cerebellum. [38] trained the Model using the selected features only and were able to increase the Model's accuracy from 63.6 to 73.8, which amounts to a 10 percent increase. [29] observed that neural networks have many redundant features that are very similar. [29] proposed an algorithm to regularize related features and therefore encourage

features diversity in the model. As a result, [29] eliminated redundant features. This method reduced the computational overhead. [29] was able to achieve a state of the art accuracy and reduce computational overhead as a result of this method

$$J_D(\phi) = \sum_{l=1}^{L} \left(\frac{1}{2} \sum_{i=1}^{n'_l} \sum_{j=1}^{n'_l} \left(\overset{(l)}{\Omega}_{ij} \right)^2 \overset{(l)}{\mathbf{M}}_{ij} \right)$$

To encourage diversity of features, [29] added the J_D term to the loss function $J(\theta; X, y)$. [29] was able to achieve a state of the art accuracy and reduce computational overhead as a result of this method.

0.2.5 Critical Evaluation of Overfitting Methods using Brain Data

Now that we have discussed some methods to address Overfitting, we will implement them and discuss their advantages and drawbacks. As such, we chose a dataset that contains images of MRI Segmentation. The dataset contains images of healthy brains and brains affected by Alzheimer's — a type of dementia that affects brain functions and interferes with regular daily activities. Accordingly, we split the data into three sets. A training set to train our Model, testing set to test our Model once the training is complete, and a validation set to test our Model during the training phase and help it find the weights that can produce the highest level of accuracy. The dataset has four classes; "Non-Demented," "Mild Demented," "Moderate Demented," and "Very Mild Demented." Each of these classes represents a stage of Alzheimer's. Early detection of Alzheimer's is critical because treating it at an early stage is much easier than at advanced stages.

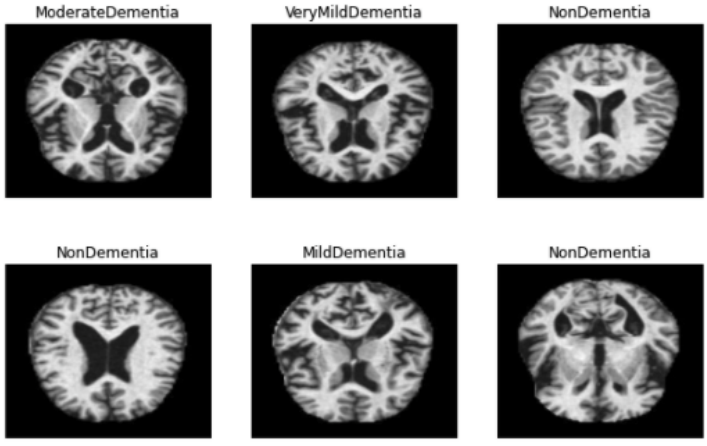


Figure 14 MRI Images

We built a basic CNN model with four convolution layers, four max-pooling layers, and two dense layers. We illustrate the model specification in figure 15.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1 (Conv2D)              (None, 128, 128, 128)     3328

 max_pooling2d (MaxPooling2D  (None, 40, 40, 128)      0
 )

 conv2 (Conv2D)              (None, 40, 40, 64)        204864

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv3 (Conv2D)              (None, 12, 12, 32)        18464

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 conv4 (Conv2D)              (None, 4, 4, 32)          9248

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 32)         0
 2D)

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 64)                2112

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 238,276
Trainable params: 238,276
Non-trainable params: 0
```

Figure 15 CNN Configuration

The Model was only able to achieve 55 % accuracy. Additionally, as shown in Figure 16, the training accuracy kept increasing while the testing accuracy pleated around 50%. Therefore, the Model is experiencing Overfitting. Consequently, we will use the methods discussed earlier to improve the Model's accuracy and reduce Overfitting.

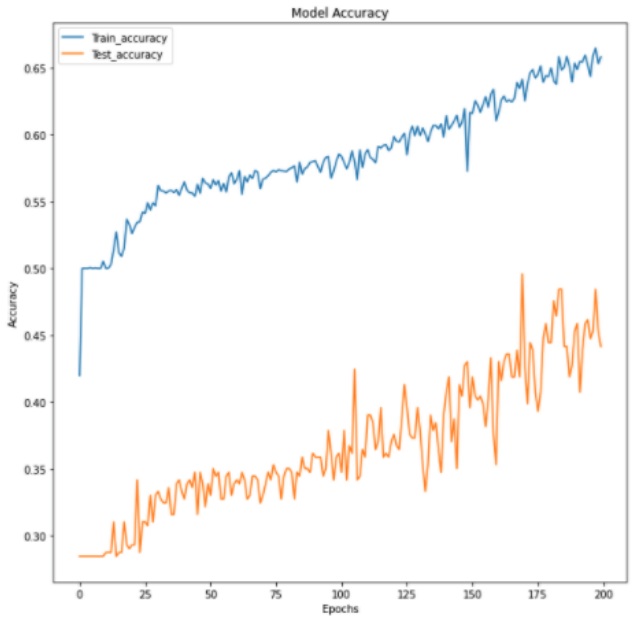


Figure 16 Training vs. Testing accuracy basic CNN model

Data Augmentation and Dropout

Originally the training dataset had a total of 5121 images. 717 MildDemented, 52 ModerateDemented, 2560 NonDemented and 1792 VeryMildDemented. On the other hand, the testing dataset had 1279 images. 179 MildDemented, 12 ModerateDemented, 640 NonDemented and 488 VeryMildDemented. As shown in [22] the left and right regions of the brain are symmetrical. We augmented the data by flipping the image along the horizontal axis. Due to Random Access Memory limitation, we could only produce 77126 images for the training set and 20601 images for the testing set. We tried to balance the dataset by producing more images in the classes with a limited number of images. For example, we only had 12 MildDemented images in the training set. As such, we produced 17925 images of the MildDemented class. Finally, we added two layers of Dropout, each with a Dropout rate of 0.5.

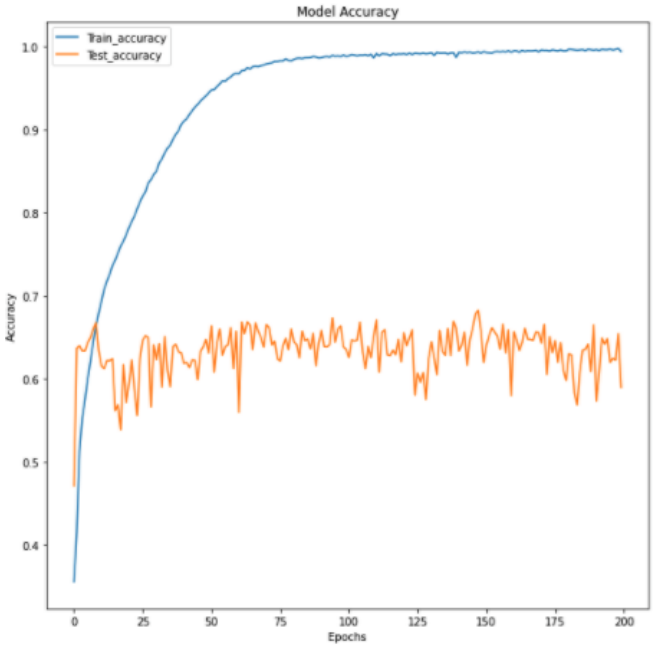


Figure 17 Training vs. Testing accuracy Data Augmentation and Dropout CNN model

We were able to increase the Model's accuracy by 4%. After applying the mentioned Dropout and Data Augmentation techniques, the Model achieved a 59% accuracy. However, as shown in figure 17, the Model is still experiencing Overfitting as the testing accuracy plateaued and the training accuracy increased after reaching roughly Epoch 25.

Data Augmentation and L2 Regularization with and without Dropout

In this section, we trained the Model with Data Augmentation in addition to L2 Regularization. Then, we trained the Model again with Data Augmentation, L2 Regularization, and Dropout regularization.

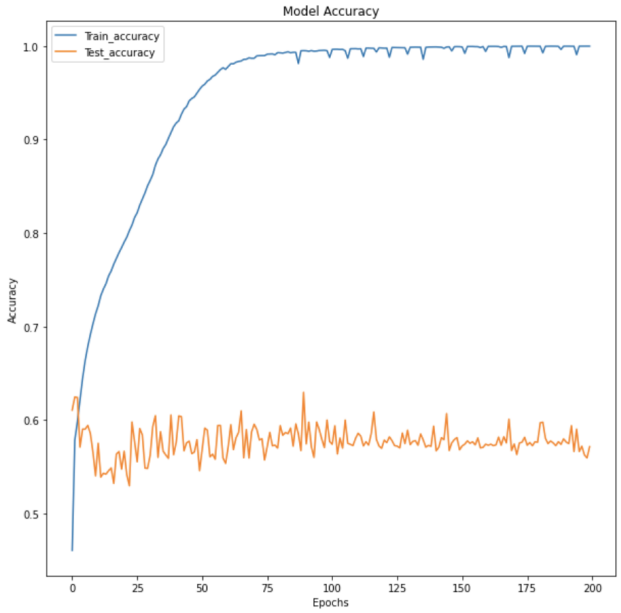


Figure 18 Training vs. Testing accuracy Data Augmentation and L2 Regularization without Dropout CNN model

When training the Model with Data Augmentation and L2 Regularization without Dropout regularization, the Model achieved 57% accuracy. However, compared to training with Data Augmentation and Dropout Regularization, the Model underperformed by 2%. We previously performed 59% accuracy with Data Augmentation and Dropout regularization. As shown in figure 18, the Model is still undergoing Overfitting as the training accuracy is plateauing at around 90%, and the testing accuracy is plateauing around 57%.

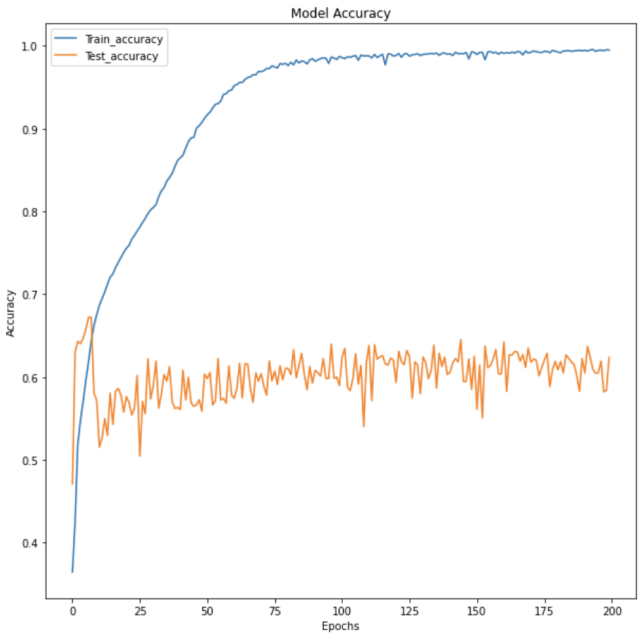


Figure 19 Training vs. Testing accuracy Data Augmentation and L2 Regularization with Dropout CNN model

When training the Model with Data Augmentation and L2 Regularization with Dropout regularization, the Model achieved 62% accuracy. Nevertheless, compared to training with Data Augmentation and Dropout regularization, the Model overperformed by 3%. We previously performed 59% accuracy with Data Augmentation and Dropout regularization. Additionally, the Model overperformed by 5% compared to the Model we trained with merely Data Augmentation and L2 regularization. As shown in figure 19, the Model is still undergoing Overfitting as the training accuracy is plateauing at around 90%, and the testing accuracy is plateauing around 62%.

Early Stopping with Data Augmentation, L2 and Dropout Regularization

In the previous models, we set a specific number of Epoch — the number of passes of the entire training dataset the Model has completed. The Model had to go through the whole training set based on a pre-specified number of epochs regardless of whether its performance improved. Therefore, training without Early Stopping took a long time and did not improve the Model's accuracy. We utilized the Early Stopping method by monitoring the Model's accuracy to save time. Since we are dealing with a classification problem, monitoring the accuracy would be most appropriate instead of monitoring the loss. Once we noticed an improvement in training accuracy, we saved the Model's configuration and returned to it for further training. We halted the training once we noticed that the Model's accuracy had not improved for a long time. We pre-specified how many epochs the Model should go through before we halt the training once we notice no improvement in accuracy. Table 2 shows how the Model improved throughout the training process. Once the Model's accuracy reached 0.63880, it stopped improving for 87 epochs, and that was where we decided to halt the training. The Model may improve if we let it go beyond 87 epochs at the expense of time.

From	To
-inf	0.52968
0.52968	0.57099
0.57099	0.57366
0.57366	0.57832
0.57832	0.58784
0.58784	0.58798
0.58798	0.59177
0.59177	0.60293
0.60293	0.60881
0.60881	0.60973
0.60973	0.63060
0.63060	0.63278
0.63278	0.63613
0.63613	0.63880

Table 2

Overall the Model achieved an accuracy of approximately 64%, Which is an improvement from the previous 62% achieved by the Model trained with Data Augmentation, L2, and Dropout regularization.

Changing the Activation Function and Removing Features

One irrelevant feature that we could have removed to improve the Model's accuracy is the skull from the MRI image. However, as shown in figure 19, the skull was already removed from the dataset. We are currently not aware of any other irrelevant features that we could remove to improve the Model's accuracy.

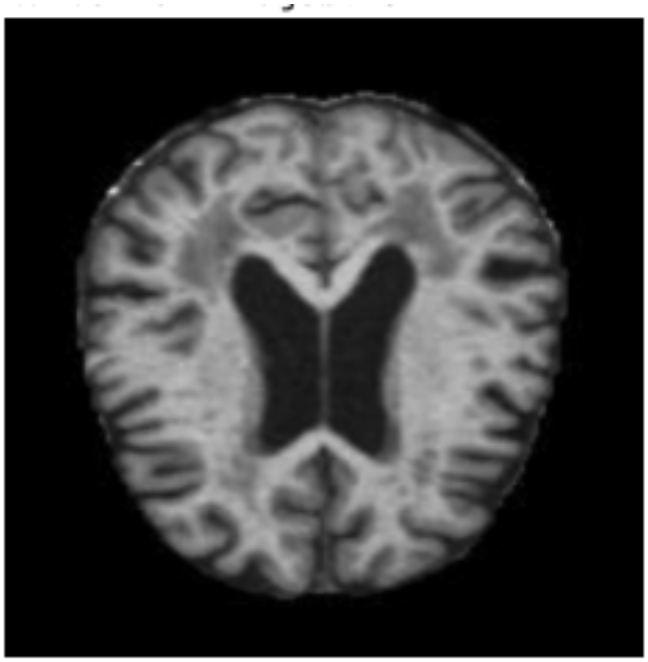


Figure 20 Brain MRI image extracted from the Alzheimer training set.

As shown in Table 3, we trained the Early Stopping with Data Augmentation, L2, and Dropout Regularization model with several different activation functions. The Model improved when using the Hyperbolic Tangent Activation Function, achieving a 66% classification accuracy. However, the accuracy worsened when using other activation functions.

Activation Function	Accuracy
Rectified Linear Unit	63%
Sigmoid	53%
Hyperbolic Tangent	66%
Leaky version of a Rectified Linear Unit	62%
Parametric Rectified Linear Unit	61%
Thresholded Rectified Linear Unit	52%
Softmax	53%

Table 3

Discussion

We reached the highest accuracy when we trained the Model with Early Stopping, Data Augmentation, L2, and Dropout Regularization using the Hyperbolic Tangent Activation Function. One drawback we noticed when using Dropout is a longer training time than training the Model without Dropout. As [28] stated, training a model with Dropout Regularization will significantly increase the training time. Models trained with Dropout Regularization take a long time than models trained without Dropout Regularization. Dropout creates thinned networks after randomly dropping several units and combines them all together. As stated in [28] training a Dropout neural network with n hidden units can be seen as training a collection of 2n different "thinned" models with extensive weight sharing. [28] improved on the traditional Dropout method by omitting to drop units from the neural network that has a positive impact on the Model's accuracy and opting to dropout units that have no potential to improve the Model's accuracy. Units with high activation values positively impact the Model's accuracy, and units with low activation values have a low impact on the Model's accuracy. Selectively adjusting the amount of Regularization on the Model's weights has increased the accuracy of neural networks by reducing Overfitting. [25] showed that it could also mitigate catastrophic forgetting. Catastrophic forgetting occurs when a network learns task A and forgets it once it starts to learn another task B. Catastrophic forgetting occurs because the weights in the network that are important for task A are changed to meet the objectives of task B. To mitigate catastrophic forgetting [25] reduced Regularization on certain weights based on how important they are to previously seen tasks. Similarly, [31] put more weight on the unit outputs that support correct predictions on the training set when they introduced a novel dropout regularization called Spectral Dropout (SD). SD discarded noisy spectral components during the train and test phases. The spectral Dropout speeds up the convergence rate during the training process. As stated in [34], to reduce computation time, many network weights are redundant, and a regularizer can remove it from the network without much loss of performance. Different from the traditional Dropout method, with the moving of convolution filters on the input feature maps, [42] dropped out different units with fixed or random drop rates, introducing more uncertainty. However, experimental results indicated that [42]'s method could prevent overfitting. [34] used a spare 1 regularizer and applied it to the matrix space of network weight to zero redundant weights and removed their unnecessary connections and neurons. [34] hypothesized that reducing the number of unnecessary connections can reduce computation and memory requirements. [25] tested this method on several datasets and was able to obtain state-of-the-art accuracy. We started with 55% accuracy with a basic CNN model, which did not use any methods to mitigate Overfitting. We ended with a 66% accuracy with the Model that used Early Stopping with Data Augmentation, L2, and Dropout Regularization using the Hyperbolic Tangent Activation Function. Given the high complexity of the task and dataset complexity, an 11% increase in accuracy is considered a good improvement. However, there are still signs of Overfitting, and the Model's accuracy can be improved further. Our model is very complex compared to the one used in [52] to predict brain age. [52]

used a lightweight CNN architecture. [52] kept only one convolutional layer before each MaxPool layer to reduce memory requirements. In addition, [52] removed all the fully connected layers, which not only greatly reduces the number of parameters. [52] reduced the GPU memory consumption by liming the channel numbers of the first layer to 32. We could use this idea in our research to reduce GPU memory consumption. A problem we faced when training our model on the Alzheimer dataset. Using data augmentation and regularization techniques, [52] achieved state-of-the-art results. [52] stated that it was intriguing why the lightweight model performs better than the deeper ones, something we can explore in our feature research. [52] noted that the choice of optimizer might affect the model performance. [52] noticed a change in performance when switching from Adam optimizer to Stochastic gradient descent Optimizer. As [44] concluded, the methods we mentioned and experimented with have produced state-of-the-art results in many fields; they have not gained as much ground over human neuroscience datasets. The high dimensionality and large amounts of noise present in neuroscience datasets, coupled with limited data available that can be reasonably obtained from a sample of human subjects, lead to difficulty training deep learning models. Overfitting can occur despite the presence of regularization techniques. [44] introduced the paired trial classification (PTC) method for classifications. PTC reduces the multiclass classification problem to two classes, potentially simplifying the problems and thus presenting a way of making it easier to achieve robust classification performance from limited training data. Instead of classifying a single example at a time into several classes, [44] instead attempt to classify pairs of examples as belonging to either the same class or different classes. [46] demonstrate that DL methods, if implemented and trained according to the common practices, have the potential to outperform standard machine learning methods with lower computational time and space despite being more complex in their architecture and parameterization, using raw or minimally preprocessed input data. [46] states that DL approaches are just beginning to show successes in diagnostic classification and disease prediction domains.

0.3 Development of DL Computational Models Addressing Overfitting

0.4 Applications of DL Handling Overfitting

0.5 Conclusions and Future Directions

Bibliography

[1] Andrey Nikolayevich Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.

[2] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[3] Alan O Sykes. An introduction to regression analysis. *Coase Sandor Institute for Law Economics Working Paper*, 1993.

[4] Thomas J Archdeacon. Correlation and regression analysis: a historian's guide. 1994.

[5] Branislav Kisacanin and Dan Schonfeld. A fast thresholded linear convolution representation of morphological operations. *IEEE transactions on image processing*, 3(4):455–457, 1994.

[6] Andrew Y Ng et al. Preventing" overfitting" of cross-validation data. In *ICML*, volume 97, pages 245–253. Citeseer, 1997.

[7] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[8] Steve Lawrence and C Lee Giles. Overfitting and neural networks: conjugate gradient and backpropagation. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 1, pages 114–119. IEEE, 2000.

[9] Inder M Verma, L Naldini, T Kafri, H Miyoshi, M Takahashi, U Blömer, N Somia, L Wang, and FH Gage. Gene therapy: promises, problems and prospects. In *Genes and Resistance to Disease*, pages 147–157. Springer, 2000.

[10] Juha Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3(Mar):1371–1382, 2003.

[11] Zhao Yue, Zhao Songzheng, and Liu Tianshi. Bayesian regularization bp neural network model for predicting oil-gas drilling cost. In *2011 International Conference on Business Management and Electronic Information*, volume 2, pages 483–487. IEEE, 2011.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[13] Janice M Beitz et al. Parkinson's disease: a review. *Front Biosci (Schol Ed)*, 6(6):65–74, 2014.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[15] Chunyan Xu, Canyi Lu, Xiaodan Liang, Junbin Gao, Wei Zheng, Tianjiang Wang, and Shuicheng Yan. Multi-loss regularized deep neural network. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(12):2273–2283, 2015.

[16] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[17] Murat Kayri. Predictive abilities of bayesian regularization and levenberg–marquardt algorithms in artificial neural networks: a comparative empirical study on social data. *Mathematical and Computational Applications*, 21(2):20, 2016.

[18] Hayrettin Okut. Bayesian regularized neural networks for small n big p data. *Artificial neural networks-models and applications*, pages 28–48, 2016.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[20] 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)*, pages 26–31. IEEE, 2017.

[21] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. Deep learning. 2017.

[22] Ammarah Farooq, SyedMuhammad Anwar, Muhammad Awais, and Saad Rehman. A deep cnn based multi-class classification of alzheimer's disease using mri. In *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*, pages 1–6, 2017.

[23] Pranay Reddy Gankidi and Jekan Thangavelautham. Fpga architecture for deep learning and its application to planetary robotics. In *2017 IEEE Aerospace Conference*, pages 1–9. IEEE, 2017.

[24] Nikhil Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

[25] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[26] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a brain: Fine-tuning by increasing model capacity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2471–2480, 2017.

[27] Zong Meng, Xuyang Zhan, Jing Li, and Zuozhou Pan. An enhancement denoising autoencoder for rolling bearing fault diagnosis. *Measurement*, 130:448–454, 2018.

[28] Alvin Poernomo and Dae-Ki Kang. Biased dropout and crossmap dropout: learning towards effective dropout regularization in convolutional neural network. *Neural networks*, 104:60–67, 2018.

[29] Babajide O Ayinde, Tamer Inanc, and Jacek M Zurada. Regularizing deep neural networks by enhancing diversity in feature extraction. *IEEE transactions on neural networks and learning systems*, 30(9):2650–2661, 2019.

[30] Daniel Jakubovitz, Raja Giryes, and Miguel RD Rodrigues. Generalization error in deep learning. In *Compressed Sensing and Its Applications*, pages 153–193. Springer, 2019.

[31] Salman H Khan, Munawar Hayat, and Fatih Porikli. Regularization of deep neural networks with spectral dropout. *Neural Networks*, 110:82–90, 2019.

[32] Haidong Li, Jiongcheng Li, Xiaoming Guan, Binghao Liang, Yuting Lai, and Xinglong Luo. Research on overfitting of deep learning. In *2019 15th International Conference on Computational Intelligence and Security (CIS)*, pages 78–81. IEEE, 2019.

[33] Yu Li, Chao Huang, Lizhong Ding, Zhongxiao Li, Yijie Pan, and Xin Gao. Deep learning in bioinformatics: Introduction, application, and perspective in the big data era. *Methods*, 166:4–21, 2019.

[34] Rongrong Ma, Jianyu Miao, Lingfeng Niu, and Peng Zhang. Transformed 1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286–298, 2019.

[35] Dan Wang, Phillip WL Tai, and Guangping Gao. Adeno-associated virus vector as a platform for gene therapy delivery. *Nature reviews Drug discovery*, 18(5):358–378, 2019.

[36] Qi Xu, M. Zhang, Z. Gu, and Gang Pan. Overfitting remedy by sparsifying regularization on fully-connected layers of cnns. *Neurocomputing*, 328:69–74, 2019.

[37] Xue Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, 2019.

[38] Youngjin Yoo, Lisa YW Tang, David KB Li, Luanne Metz, Shannon Kolind, Anthony L Traboulsee, and Roger C Tam. Deep learning of brain lesion patterns and user-defined clinical and mri features for predicting conversion to multiple sclerosis from clinically isolated syndrome. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 7(3):250–259, 2019.

[39] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International conference on artificial intelligence and statistics*, pages 4313–4324. PMLR, 2020.

[40] Reza Moradi, Reza Berangi, and Behrouz Minaei. A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947–3986, 2020.

[41] Guangyuan Pan, Yang Wu, Ming Yu, Liping Fu, and Hongsheng Li. Inverse modeling for filters using a regularized deep neural network approach. *IEEE Microwave and Wireless Components Letters*, 30(5):457–460, 2020.

[42] Hengyue Pan, Xin Niu, Rongchun Li, Siqi Shen, and Yong Dou. Dropfilterr: A novel regularization method for learning convolutional neural networks. *Neural Processing Letters*, 51(2):1285–1298, 2020.

[43] Eduard Sariev and Guido Germano. Bayesian regularized artificial neural networks for the estimation of the probability of default. *Quantitative Finance*, 20(2):311–328, 2020.

[44] Jacob M Williams, Ashok Samal, Prahalada K Rao, and Matthew R Johnson. Paired trial classification: A novel deep learning technique for mvpa. *Frontiers in Neuroscience*, 14:417, 2020.

[45] Chenzhong Yin, Xiongye Xiao, Valeriu Balaban, Mikhail E Kandel, Young Jae Lee, Gabriel Popescu, and Paul Bogdan. Network science characteristics of brain-derived neuronal cultures deciphered from quantitative phase imaging data. *Scientific Reports*, 10(1):1–13, 2020.

[46] Anees Abrol, Zening Fu, Mustafa Salman, Rogers Silva, Yuhui Du, Sergey Plis, and Vince Calhoun. Deep learning encodes robust discriminative neuroimaging representations to outperform standard machine learning. *Nature communications*, 12(1):1–17, 2021.

[47] Javier Barbero-Gómez, Pedro-Antonio Gutiérrez, Víctor-Manuel Vargas, Juan-Antonio Vallejo-Casas, and César Hervás-Martínez. An ordinal cnn approach for the assessment of neurological damage in parkinson's disease patients. *Expert Systems with Applications*, page 115271, 2021.

[48] basel99. Parkinson's disease detection, Jul 2021.

[49] Nanziba Basnin, Nazmun Nahar, Fahmida Ahmed Anika, Mohammad Shahadat Hossain, and Karl Andersson. Deep learning approach to classify parkinson's disease from mri samples. In *International Conference on Brain Informatics*, pages 536–547. Springer, 2021.

[50] Kexin Huang, Cao Xiao, Lucas M Glass, Cathy W Critchlow, Greg Gibson, and Jimeng Sun. Machine learning applications for therapeutic tasks with genomics data. *arXiv preprint arXiv:2105.01171*, 2021.

[51] Ujjwal Kumar and Anamitra Bhar. Studying and analysing the effect of weight norm penalties and dropout as regularizers for small convolutional neural networks. *International Journal of Engineering Research Technology*, 2021.

[52] Han Peng, Weikang Gong, Christian F Beckmann, Andrea Vedaldi, and Stephen M Smith. Accurate brain age prediction with lightweight deep neural networks. *Medical image analysis*, 68:101871, 2021.

[53] Adrian Schwarzer, Steven R Talbot, Anton Selich, Michael Morgan, Juliane W Schott, Oliver Dittrich-Breiholz, Antonella L Bastone, Bettina Weigel, Teng Cheong Ha, Violetta Dziadek, et al. Predicting genotoxicity of viral vectors for stem cell gene therapy using gene expression-based machine learning. *Molecular Therapy*, 2021.