A New Regularization Term for Deep Neural Networks With Applications to Biological Data

by

Zerotti Woods

(Under the Direction of Juan B. Gutierrez)

Abstract

In this work, we present a new regularization term that penalizes the conditioning of the weight matrices in a deep neural network. We give a mathematical argument that suggests that in certain situations, the conditioning number of the weight matrices have a direct impact on the error in classification. Empirical evidence suggests that improving the weight matrix associated with the output layer of a matrix improves generalizability when classifying ECG data from a benchmark data-set, and also a novel malaria infection data-set.

Index words: Deep Learning, Regularization, Artificial Intelligence, Machine Learning, Computational Biology

A New Regularization Term for Deep Neural Networks With Applications to Biological Data

by

Zerotti Woods

B.S., Morehouse College, 2014

M.A., The University of Georgia, 2016

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2019

A New Regularization Term for Deep Neural Networks With Applications to Biological Data

by

Zerotti Woods

Approved:

Major Professor: Juan B. Gutierrez

Committee: Caner Kazanci
Daniel Krashen
Qing Zhang

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
May 2019

Dedication

This thesis is dedicated to every teacher, professor, and mentor that had seen things in me that I did not see in myself. Without each of you, I'm sure this would have been possible.

Acknowledgments

I would like to first thank The Biomathematics Research Group. In particular, I would like to thank Elizabeth Trippe, Jessica Brady, Tao Sheng, Yi Yan, Diago Moncada, and Saeid Safaei. Each of these individuals put in countless hours to contribute to this research and I learned a tremendous amount from each of them. I also would like to acknowledge my advisor, Dr. Juan B. Gutierrez. He had a vision to put together one of the most diverse labs that exists at UGA. With his vision came ideas that no one individual or one discipline could have done alone. Because of the team that Juan assembled, and his leadership as a colleague and not a boss, I was given the opportunity to grow as a scientist in an environment that many others did not have.

This project was funded in part by Federal funds from the US National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services under contract #HHSN272201200031C, 2012-2017, which supported the Malaria Host-Pathogen Interaction Center (MaHPIC). This work was also supported by the Defense Advanced Research Projects Agency and the US Army Research Office through the program Technologies for Host Resilience - Host Acute Models of Malaria to study Experimental Resilience (THoR's HAMMER), DARPA contract #W911NF-16-C-0008, 2016-2019. All experiments were performed at the Yerkes National Primate Research Center and are supported in part by the National Institutes of Health, Office of the Director (P51OD011132). The content of this work is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or other funding organizations.

I would like to thank all the members of the MaHPIC-HAMMER consortium.

Lastly, I would like to thank my family and friends for all of the support that they gave me throughout this process.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Review

1.1 Background of machine learning

The promise that has been shown recently for machine learning makes one excited about its potential. There have been reports of many successes with machine learning in areas such as computer vision, speech recognition, biology, drug discovery, genomics and a host of other areas [2, 25, 28, 56]. Machine learning is not a new concept; there are pioneering papers for the subject that date back as far as the 1950s [58]. The subject seemed to fall out of favor over the years but, with the advances in computing capabilities in recent years and the explosion of large datasets, we are now better positioned than we were in the past to explore the capabilities of machine learning.

One characterization of machine learning is, "a composition of multiple processing layers to learn representations of data with multiple levels of abstraction" [28]. Neural networks are popular machine learning models that have been studied extensively throughout the literature. Figure 1.1 shows a diagram of a fully connected neural network with 2 input units, 3 neurons on the first hidden layer, 2 neurons on the second hidden layer and one output unit.

A mathematical description inspired by Saarine *et al.* (1993) [46] of a two hidden layer feedforward Neural Network is now given:

- a := Total number of input units.
- b := Total number of first hidden layer neurons.
- c := Total number of second hidden layer neurons.

- m := The number of data points.
- n := $b(a+c+1)+2c$ total number of parameters.
- N_k := The k^{th} neuron $1 \leq k \leq b+c$.

We define an a-b-c feedforward neural network as the following:
For the i^{th} data point $\mathbf{x}^i = (x_1^i, x_2^i, \ldots, x_a^i)$

$$N_l = g\bigg(\sum_d w_{(l-1)(a+1)+1+d} x_d^i + w_{(l-1)(a+1)+1}\bigg), \tag{1.1}$$

where $1 \leq l \leq b$, $1 \leq d \leq a$ and $g(\bullet, \bullet)$ is an "activation" function of the user's choosing.

$$N_{b+p} = f\bigg(\sum_q w_{t+(p-1)(b+1)+1+q} N_q + w_{t+(q-1)(b+1)+1}\bigg), \tag{1.2}$$

where $1 \leq p \leq c$,$1 \leq q \leq b$, $t = p(h+1)$ and $f(\bullet, \bullet)$ is an "activation" function of the users choosing. So the output of an a-b-c feedfoward neural network for input $\mathbf{x}^i$ and weight vector $\mathbf{w}$ is

$$F(\mathbf{x}^i, \mathbf{w}) = \sum_r w_{u+r} N_{b+r} \tag{1.3}$$

where $u = t + c(b+1)$ and $1 \leq r \leq c$.

According to Deng and Yu (2014) [13], a definition of deep learning is "a class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification." Although there are many reports of success for deep learning, the understanding of how and why it works so well is still underdeveloped [31]. It is a well-known result that feedforward neural networks have universal approximating abilities for functions under reasonable assumptions [4, 11, 20, 31, 39]. In particular Leshno *et al.* (1993) [30] proved the following theorem:

Theorem 1.1.1 (Leshno *et al.* (1993))**.** *Let M denote the set of functions which are L^∞ on any compact subset of $\mathbf{R}^n$ and the closure of the set of points of discontinuity have Lebesgue measure zero.*

Two Hidden Layer Neural Network with two input units and one output unit

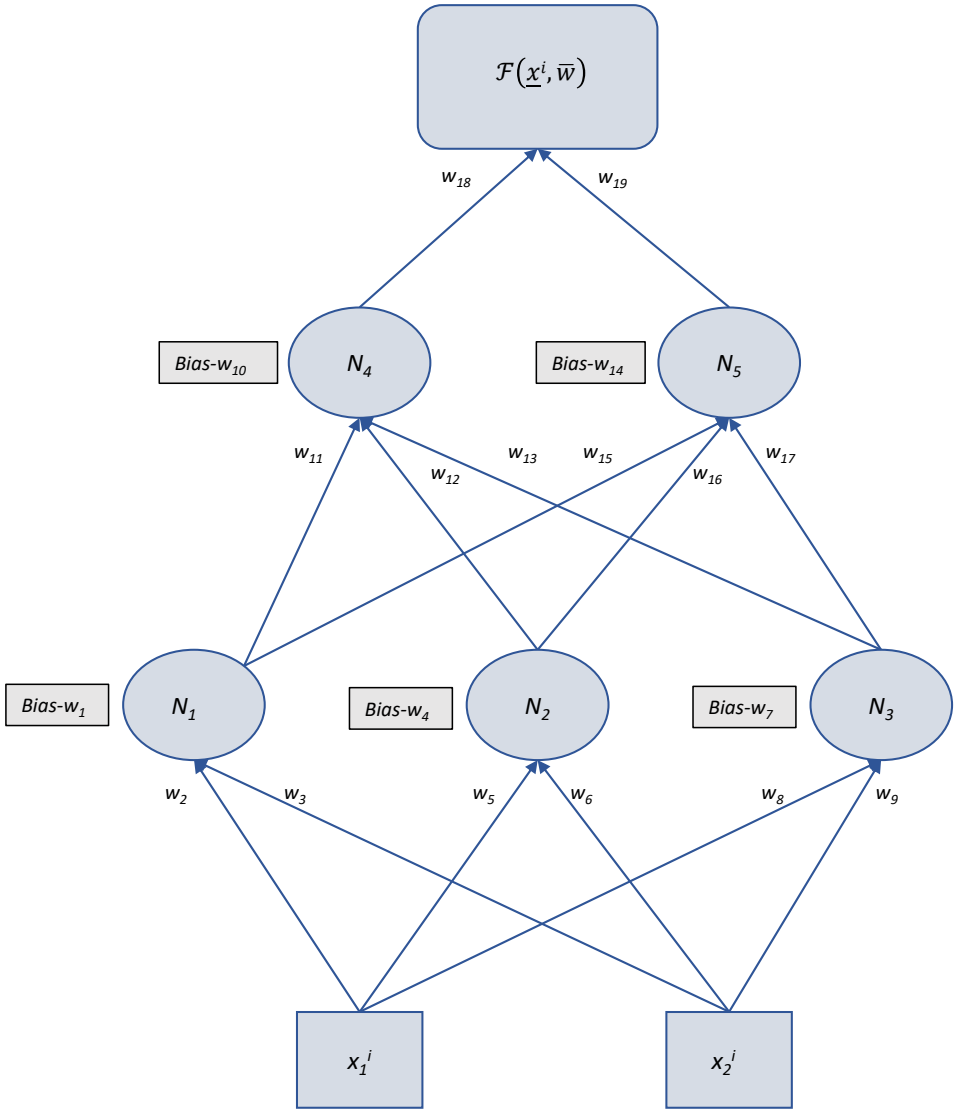


Figure 1.1: Two hidden layer feedfoward neural network with 2 input units, 3 neurons on the first hidden layer, 2 neurons on the second hidden layer, and one output neuron.

For any $\sigma \in M$ *span*$\{\sigma(\mathbf{w} \bullet \mathbf{x} + \Theta) : \mathbf{w} \in \mathbf{R}^n, \Theta \in \mathbf{R}\}$ *is dense in* $C(\mathbf{R}^n)$ *if and only if* σ *is not an algebraic polynomial (a.e)*

In supervised learning, deep learning usually attempts to learn features of a given data set, to be able to make predictions on similar data that the system has not yet seen. After training a model, we usually seek some metric to determine how well a model is performing on data that it has not been trained on, (commonly called test data). Many times data-sets are skewed, which makes accuracy (the fraction of predictions that the model got correct of the entire list of predictions) a misleading metric many times. Consider an example where we create a model that would classify a data set $\{\mathbf{x}\}$ into two classes $\{0, 1\}$. Now assume that 90% of the data is in the 0 class. If our model predicts that all data belongs to the 0 class, then we get the misleading result that our model is 90% accurate. Many times on skewed data sets, a better metric to use is the F_1 score.

Definition 1.1.2.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$
$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$
$$F_1 = \frac{Precision \times Recall}{Precision + Recall}$$

The F_1 score is the harmonic average of precision and recall. This statistic is used for binary classifiers but is used many times in multiclass classifiers by calculating the F_1 score for each class individually. The F_1 score is a metric that is between 0 and 1 with 0 being the worst and 1 being the best. Going back to our example of the model that calculated 0 for every data point in the skewed data set $\{\mathbf{x}\}$ we see that the F_1 score is .47. This statistic is used for scoring models on skewed data-sets throughout the literature [17, 43, 59].

1.2 Convolutional Neural Networks

In 1959, Hubel and Wiesel [21] performed a series of experiments on a cat's cortex. With the use of electrodes that were placed on the cat's brain, they monitored the response of different

neurons when the cats were given different visual stimuli. Some of the results that were observed are that there is a hierarchical organization to the neurons. So there are different cells responsible for response to different stimuli. For example, there are simple cells that responded to light, there are complex cells that respond to light, orientation, and movement, and hypercomplex cells that respond to movement with an endpoint. This brought rise to the Neurocognitron in 1980 [15]. This model simulated the ideas of Hubel and Wiesel and created a model that alternated between simple layers and complex layers. The model is credited to be the first step towards developing the convolutional neural network (CNN) framework. In 1998 Lecun *et al.* [29] was the first to demonstrate gradient-based learning on a CNN.

Because of computational limitations, the popularity of the Convolutional Neural Networks didn't begin to rise until 2012 with "Alexnet" [26]. The purpose of Krizhevsky's work was to show that it is possible to efficiently train a deep CNN and also to show that CNNs could dramatically surpass the state of the art at that time. The vast majority of the literature for CNNs are in the fields of image processing, computer vision, and natural language processing. There has also however been many reports of successes in many areas of human health [8, 17, 42]. A brief explanation of the different components of a convolutional neural network inspired by Aghdam and Heravi (2017) [16] is now given.

Convolutional Layers

Convolutional layers are the central pieces to a CNN. Convolutional layers take as input a three-dimensional representation of the data set (width, height, depth). It is easy to extend to one-dimensional data by making the third dimension of the data 1. The parameters of the layer include a set of trainable **filters** (also called **kernels**) that only look at small bits of the data at any given time. The language used often is that filters have a small receptive field. The creator of the CNN decides the amount of data that is considered at any given

time by the filters. The filters are convolved across the input volume by computing the dot product of the elements of the filter and the input volume. The filter then moves over by an amount that is chosen by the user. The amount that the filter moves is called the stride of the filter. The smaller the stride, the more overlap a particular filter haves as it passes through the data. This process produces a two-dimensional activation map. The user also chooses the number of filters to use in every layer. Each activation map is stacked in the third dimension, so the output of the convolutional layer is a three-dimensional volume.

After the three dimensional volume if formed the application of a nonlinear activation function is usually applied to each element in the volume. Rectified Linear Unit (ReLU) is used many times for CNNs in the literature. ReLU is defined as

$$f(x) = ReLU(x) = \max(0, x). \tag{1.4}$$

There are many other choices of activation functions including,

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{1.5}$$

$$f(x) = |tanh(x)| = \left|\frac{e^x - e^{-x}}{e^x + e^{-x}}\right|, \tag{1.6}$$

$$\sigma(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}. \tag{1.7}$$

Functions (1.5), (1.6), and (1.7) are all saturating nonlinear functions. Krizhevsky *et al.* (2012) [26] found that these functions train several times slower than the nonsaturating function in (1.4). Because of the speed of training that comes with using nonsaturating activation functions, ReLU has become the activation function of preference in many deep neural networks.

Convolutional layers have the feature of local connectivity. This is one key difference in the convolutional neural network from the fully connected neural network. Convolutional neural networks take advantage of the spatial structure of the data. This idea also comes from the research that Hubel and Wiesel did [22]. The idea is that spatially close data is highly correlated. In fully connected neural networks the spatial component is ignored. If

the input of a fully connected neural network is rearranged then nothing is affected since the weights are connected to every input.

One other feature of the convolutional layer is weight sharing. Each filter slides across the entire input. The filter has the same weights and bias to analyze the entire input dimension, in contrast to a fully connected neural network where a diferent weight is connected to every component of the input. This reduces the number of parameters needed to define the layer. The idea is that if a filter is good at detecting a feature in one part of the data, then it should also be good at detecting a feature at another part of the data. This feature also gives the opportunity to train much deeper and complex networks because using fewer parameters in each layer makes computation cost decrees giving one the opportunity to use computing power in deeper layers.

Pooling Layer

Another component of CNNs is the pooling layer. The purpose of the pooling layers is to achieve spatial invariance by reducing the resolution of the feature maps [23]. It is a way to downsample your data in a non-linear fashion. Max pooling has become the most popular pooling layer. It partitions the data into subsets, and for each subset, max pooling outputs the maximum value of the subset. This process is done for every depth slice in the input volume of the pooling layer, so this leaves the depth of the input volume unchanged. Other examples of pooling layer include average pooling which partitions the data into subsets and for each subset outputs the average of each subset, but Hutchison *et al.* (2010) [23] showed that max pooling is far superior for capturing invariances, so other pooling layers are not used as often in state of the art CNNs.

One concern with pooling layers is the quick dimension reduction of the data. This dimension reduction can be attractive in some sense because it reduces the number of trainable parameters which speeds up computation, but this can be a concern because information may be lost too quickly and performance suffers. One way to control this is to zero-pad the

borders of the input volume. Zero-padding essentially adds zeros to the borders of the input volume to make the dimension larger which in turn padding ensures that the output of the pooling layer does not have a dimension that is so small that important information is lost. Pooling also improves generalization in neural networks which is also shown in Hutchison *et al.* (2010) [23].

Fully Connected Layer

Finally, most CNNs have fully connected layers. They are usually the last layers of a CNN. At this point, the neurons are not sparsely connected anymore and now go back to being fully connected. Fully connected layers assist in classification in many CNNs [8, 17, 26, 54, 59]. When faced with a classification problem with k classes throughout the literature the activation function for the last dense layer is a k dimensional $softmax$ function

$$softmax(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}}, \tag{1.8}$$

for $j = 1, \ldots, k$ and $\mathbf{x} = (x_1, \ldots, x_k)$.

1.3 Residual Networks

Works from Simonyan and Zisserman (2015) and also Szegedy *et al.* (2015) [49, 54] recently showed evidence that the depth of a neural networks is important to its overall performance. There are many examples of deep neural networks that enjoy increased performance as compared to their shallow counterparts [17, 26, 34, 49, 54]. In the work of He *et al.* [19] the natural question "*is learning better networks as easy as stacking more layers?*" was asked. Counterintuitively the works of He and Sun (2014), He *et al.* (2014), and Srivastava *et al.* (2015) [18, 19, 53] showed that in many instances, neural networks' performance start to degrade as the network becomes deeper. Figure 1.2 shows an example of this phenomena.

Consider a shallow neural network $F(\Theta, \mathbf{x})$ and its deeper counterpart $G(\Phi, \mathbf{x})$ that is constructed by adding more layers to $F(\Theta, \mathbf{x})$. With this construction, it would seem as

Training Error and Test Error on CIFAR-10 With 20-Layer and 56-Layer Network

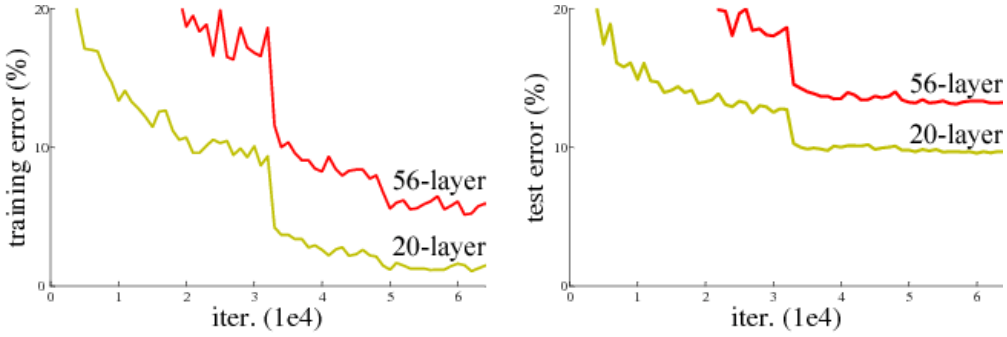


Figure 1.2: Image taken from He *et al.* (2015)

though the performance of $G(\Phi, \mathbf{x})$ should be no worse then the performance of $F(\Theta, \mathbf{x})$ since we could make all layers in $G(\Phi, \mathbf{x})$ that are not in $F(\Theta, \mathbf{x})$ the identity map. He *et al.* [19] points out that in consideration of this constructive example it seems that the degradation problem experienced with deep neural networks lies in the optimization of the network. This example points out that in many cases, gradient descent has a hard time optimizing deeper networks.

He *et al.* [19] presents a solution to the above problem. Let $H(\mathbf{x})$ denote the desired mapping of the input vector $\mathbf{x}$. Instead of trying to find a neural network that produces $H(\mathbf{x})$ directly, we allow the stacked nonlinear layers to fit the residual mapping

$$F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}. \tag{1.9}$$

Reformulating the problem, we would like stacked nonlinear layers to fit

$$H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}. \tag{1.10}$$

We can achieve this with neural networks by introducing skip connections. Figure 1.3 is an example of skip connections.

Skip connections are the building blocks of Residual Neural Networks (RESNET). He *et*

Skip connections: The Building Blocks of Residual Networks

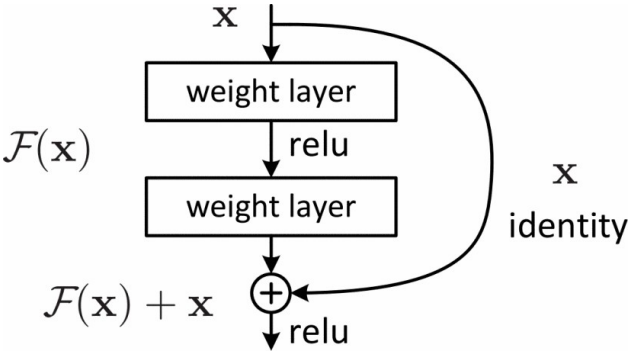


Figure 1.3: Image taken from He *at al.* (2015)

al. [19] hypothesized that it is easier to optimize the residual mapping than to optimize the original mapping. The idea is that if the identity mapping is optimal, then it would be easier to push the residual to zero then to fit the identity map with stacked nonlinear functions. One attractive feature of RESNET is that it adds no more parameters to its plain counter and no more computational complexity.

He *et al.* (2015) [19] found that RESNET dramatically outperformed its' plain counterparts. This framework also gives the intuitive result that deeper networks perform better than shallow networks. It also showed that in some cases convergence was faster in RESNET even when performance was similar in shallow networks. Figure 1.4 shows a comparison of a RESNET framework compared to a plain deep neural network, and Figure 1.5 shows the performance of plain networks of different depths compared to RESNET of different depths. In Figure 1.5 the thin lines represent the training error, and the thicker lines represent the test errors.

1.4 Regularization in Neural Networks

The generalization ability of a neural network refers to the ability of a neural network to be able to predict things about a data set that it was not trained on assuming the new

Residual Network Compared to Plain Counterparts

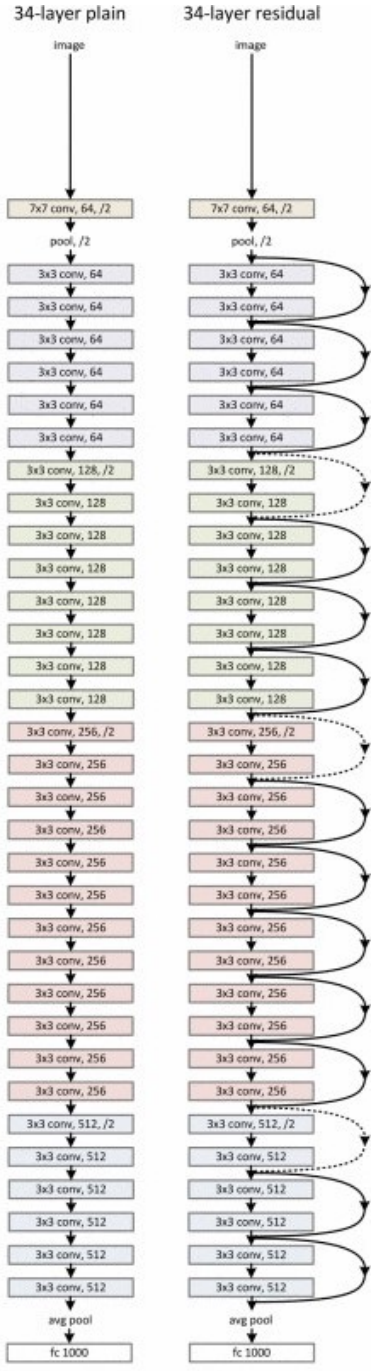


Figure 1.4: Image taken from He *at al.* (2015)

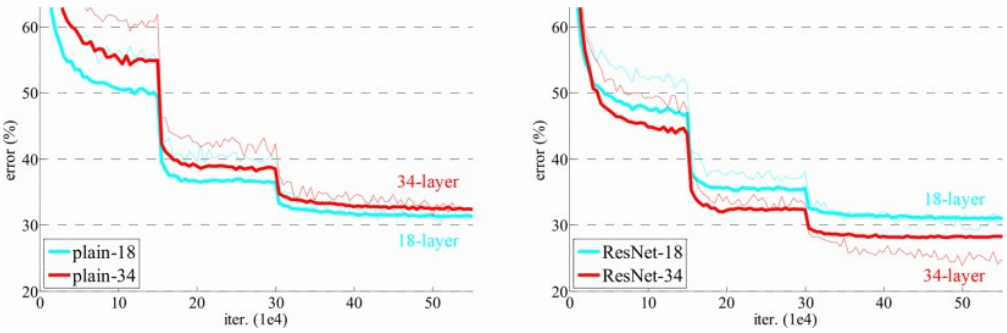


Figure 1.5: Image taken from He *at al.* (2015)

data is similar to the training data. Regularization of a neural network is defined to be any modification to a neural network to reduce its generalization error. The phenomenon of achieving low error on a test set and high generalization error is referred to as overfitting. The study of reducing generalization error without underfitting the model has been of great interest. The following are algorithms that have been presented in the literature to reduce overfitting.

L_1 and L_2 Normalization

The complexity of an unregularized neural network is fairly understood to increase as a function of its size and depth [35]. A neural network F with parameters Θ is said to shatter a data-set $\{x_1, x_2, \ldots, x_n\}$ if, for all assignments of labels to those points, there exists a θ such that the model f makes no errors when evaluating that set of data points. The VC dimension of a model F is the maximum cardinal D such that a data-set $\{x_1, x_2, \ldots, x_n\}$ of cardinality D can be shattered by F [48]. With hard-threshold activations, the VC-dimension, of the class of functions realizable with a feed-forward network is equal, up to logarithmic factors, to the number of edges in the network which corresponds to the number of parameters [48].

The problem with this is that with more complex functions comes more opportunities to overfit data.

Consider a neural network $F(\Theta, \mathbf{x})$ with n neurons and with corresponding loss function $L(\Theta, \mathbf{x})$. Define the hypothesis class $\mathcal{H}$ as all possible realizations of F that we are searching for. A regularization function is a mapping $\Omega : \mathbb{R}^d \to \mathbb{R}$ that intuitively measures the "complexity" of a model. The regularized loss minimization rule produces

$$argmin_{\Theta}(L(\Theta, \mathbf{x}) + \Omega(\Theta)). \tag{1.11}$$

Minimizing 1.11 gives a balance between a low cost function and a "less complex" model according to Shalev-Shwartz and Ben-David (2014) [48]. One function that is used often is

$$\Omega(\Theta) := \lambda ||\Theta||_2^2, \tag{1.12}$$

where $\lambda > 0$ is a scaler and $||\bullet||_2$ is the l_2 norm given by $||\Theta||_2 = \sqrt{\sum_i \sum_j \Theta_{i,j}^2}$. We can look at the l_2 norm of the perameter Θ as a measure of "complexity". One intuitive way to define a sequence of nested hypothesis classes $\mathcal{H}_1 \in \mathcal{H}_2 \in \mathcal{H}_3 \ldots$ where $\mathcal{H}_i$ is all neural networks with the same architecture of $F(\Theta, x)$ and $||\Theta||_2 < i$. This regularization is called *Tikhonov regularization* or *weight decay*.

Krogh and Hertz (1992), Srivastava *et al.* (2014), Zeiler and Fergus (2013) [27, 52, 65] have all shown examples of successes with normalization with weight decay. The speed and simplicity of the regularization term are what has made it so popular in the past. It has been useful but in recent times with newer techniques weight decay has not been used so much in the literature.

A straightforward modification that we can do to the weight decay minimization term changes the norm from l_2 to l_1. So we now define

$$\Omega_1(\Theta) := \lambda ||\Theta||_1, \tag{1.13}$$

and analyze

$$argmin_{\Theta}(L(\Theta, x) + \Omega_1(\Theta)) \tag{1.14}$$

This regularization term is called Lasso and was presented by Tibshirani (1995) [64]. The key difference presented in the work of Tibshirani (1995) [64] is that Lasso defines a continuous shrinking operation that can produce coefficients of exactly 0. The result is sparsity in data and can lead to a sparser representation in the input data set and also a neural network with fewer neurons. Sparsity is attractive in the sense that it allows for less memory demands and also gives a better feature selection in the input data. There has been many reports of lasso outperforming weight decay including Ng (2004), Srivastava (2014), Tibshirani (1996) [36, 52, 64]. Srivastava (2014) [52] also reported results when they combined lasso and weight decay.

Dropout

Ensemble methods use multiple learning models to obtain better prediction power than that of one single model. Ensemble methods have been shown many times to outperform any of the single classifiers in the ensemble [38, 40, 45]. Srivastava (2014) [52] stated that "with unlimited computation, the best way to regularize a fixed-size model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data." Also according to Srivastava *et al.* (2014), [52] ensemble methods are most useful when models in the ensemble have different architectures and are trained on different data. This is unfeasible many times in practice because of limited computation capacity, limited data, and limited time.

Dropout is a technique the prevents overfitting and also approximates combining exponentially many models to increase prediction power. The name "dropout" comes from dropping out neurons in a stochastic manner while training a neural network. Dropout makes neuron learn with randomly chosen samples of other neurons. This forces the neuron to learn more independently and not depend on other neurons to learn because there is no guarantee that the neuron will remain in the system. A neural network with n neurons in it can be seen as a collection of 2^n "thinned" possible neural networks according to Srivastava (2014)

[52]. The idea is to estimate combining these 2^n thinned models to make predictions at test time. Let L be the number of hidden layers in a neural network. Let $l \in \{1, \ldots, L\}$. Let $z^{(l)}$ be the vector of inputs of layer l. Let $y^{(l)}$ denote the output of layer l with the convention that $y^{(0)} = x$ is the network's input, and $W^{(l)}, b^{(l)}$ be the trainable weights and bias for layer l. Then in a standard neural network for any neuron i in layer l we have

$$z_i^{l+1} = W_i^{(l+1)} y^{(l)} + b_i^{(l+1)}, \tag{1.15}$$

$$y_i^{(l+1)} = f(z_i^{l+1}), \tag{1.16}$$

for some activation function f.

In order to apply dropout we do the following:

$$r_j^{(l+1)} \sim Bernoulli(p), \tag{1.17}$$

$$\widetilde{y}^{(l+1)} = r^{(l+1)} \cdot y^{(l+1)}, \tag{1.18}$$

$$z_i^{l+1} = W_i^{(l+1)} \widetilde{y}^{(l+1)} + b_i^{(l+1)}, \tag{1.19}$$

$$y_i^{(l+1)} = f(z_i^{l+1}). \tag{1.20}$$

Where $r^{(l)}$ is a vector of independent Bernoulli random variables, each having probability p of being 1 and $\bullet$ denotes componentwise multiplication. The choice of p is a hyperparameter however Srivastava (2014) [52] notes that in practice, most times in the earlier parts of the network p should be close to one and goes down as you get deeper in the network. Notice that adding dropout to a layer in a neural network effectively eliminates some neurons in that layer by making them zero. Forward and backpropagation are done in a very similar manner with the only acceptation that it is done only on the thinned networks. This gives the effect of training 2^n thinned neural networks that share weights but training them very rarely if any at all. At test time the original neural network is used with the modification that the weights used in the trained neural network are $W_{test}^{(l)} = pW^{(l)}$. This step is an approximation of averaging the 2^n thinned models that ensure that for any neuron the expected output under the distribution used to drop neurons during training is the same as the actual output

at test time.

Dropout was shown to improve generalization performance on a wide range of data domains. These domains included computer vision, speech recognition, document classification, and computational biology [52]. It surpassed the generalizing capabilities of every regularization technique that used except for Bayesian Neural Networks. This outperformance was expected because Bayesian Neural Networks takes a more accurate approach for model averaging. It is important to note that according to Srivastava (2014) [52] Bayesian Neural Networks are much slower to train and hard to scale to large scale problems.

Batch Normalization

Ioffe and Szeged (2015) [24] described a new technique called Batch Normalization to deal with the problem of *internal covariate shift* during the training process of neural networks. In this work *internal covariate shift* is defined to be when the input distribution to a learning system changes. It is a known result from LeCun *et al.*(1998),Wiesler and Ney (2011) [29, 62] that when training a neural network that training converge faster if one first *whiten* the input values by doing a linear transformation of subtracting the mean and divide by the standard deviation of the input data and also decorrelating the data.

Consider a neural network that computes

$$l = F_2(F_1(\mathbf{x}, \Theta_1), \Theta_2), \tag{1.21}$$

with Θ_1 and Θ_2 being the parameters to optimize with objective function J, and F_1 and F_2 are some arbitrary activation functions. Wiesler and Ney (2011) [62] showed that the optimization process would benefit greatly by whitening. Ioffe and Szeged (2015) [24] considered

$$l_2 = F_2(\mathbf{u}, \Theta_2) \tag{1.22}$$

as a sub-network of l with $u = F_1(x, \Theta_1)$ as the input of l_2. This work pointed out that using the result of Wiesler and Ney (2011) [62] for l_2 we see that the training process also benefits from whitening u. This work noted that there could be some complications with

not allowing the optimization algorithm used to optimize the network to take into account the normalization step that was described. To address this issue, they wanted to make sure that for any parameter value, the network always produced activations with the desired distribution. To achieve this, they considered whitening the layers input in a differentiable way. This work also noted that trying to whiten the layer's input is expensive and not everywhere differentiable. Because of this observation, they introduced two simplifications:

- Instead of whitening the features in layer inputs and outputs jointly, they normalize each scalar feature independently, by making it have the mean zero and variance 1
- Since mini-batches are used training, each mini-batch produces estimates of the mean and variance of each activation.

One other thing noted in Ioffe and Szeged (2015) [24] is that simply normalizing each input of a layer may change the representation power of the layer. An example of this is if we normalize the inputs of a sigmoid it constrains them to only the linear like part of the nonlinearity which would lose some of the power of the sigmoid function. To correct this they introduce to learnable parameters $\gamma^{\langle k \rangle}, \beta^{\langle k \rangle}$ for each activation $x^{\langle k \rangle}$, where $x = (x^{\langle 1 \rangle}, \ldots, x^{\langle d \rangle})$. These parameters scale and shift the normalized value

$$y^{\langle k \rangle} = \gamma^{\langle k \rangle} \widetilde{x}^{\langle k \rangle} + \beta^{\langle k \rangle}, \tag{1.23}$$

where $\widetilde{x}^{\langle k \rangle} = \frac{x^{\langle k \rangle} - E[x^{\langle k \rangle}]}{\sqrt{Var[x^{\langle k \rangle}]}}$

Introducing these new parameters allows the model to learn the optimal mean and variance of the model and restored the capabilities of the layer. Using the above observations Ioffe and Szeged (2015) [24] created the following algorithm

Algorithm 1 Batch Normalization Transform

INPUT:

Values of x over a mini batch:$B = \{x_{1...m}\}$; Parameters to be learned: γ, β, ϵ(numerical stability)

OUTPUT:

$\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ (mini batch mean)

$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$ (mini-batch variance)

$\widetilde{x}^{\langle k \rangle} \leftarrow \frac{x-\mu_\beta}{\sqrt{\sigma_\beta^2+\epsilon}}$ (Normalize)

$y_i \leftarrow \gamma\widetilde{x} + \beta \equiv BN_{\gamma,\beta}(x_i)$ (scale and shift)

Ioffe and Szeged (2015) [24] also points out that even though when creating Batch Normalization, regularization of the network in the traditional sense was not the goal, Batch Normalization regularizes the network. It comes from the fact that noise is introduced in the system by estimating the mean and variance of the inputs by the means and variances of mini-batches. In their experiments, they found that when comparing between the "inception" model Szeged *et al.* (2015) [54] and the "inception" model after Batch Normalization they were able to gain 2.2% on the training set while dropping the number of epochs tremendously.

1.5 Ill conditioning In Neural Networks

The condition number of a function is defined as:

Definition 1.5.1. *The condition number of a differentiable function* $H(\mathbf{x}) : \mathbf{R}^n \to \mathbf{R}^m$ *is*

$$\kappa(H(\mathbf{x})) = \frac{||x||||J(\mathbf{x})||}{||H(\mathbf{x})||}. \tag{1.24}$$

Such that $J(\mathbf{x})$ *is the Jacobian of the function* $H(\mathbf{x})$ *and* $||\bullet||$ *is a norm on* $\mathbf{R}^m$

Intuitively this measures how much the output of $H(\mathbf{x})$ changes for a small change in $\mathbf{x}$. The above definition is canonical if given a small change in $\mathbf{x}$, we think of the ratio of the

relative change in output of $H(\mathbf{x})$, $[||H(\mathbf{x}+\delta\mathbf{x})-H(\mathbf{x})||/||H(\mathbf{x})||]$ and the relative change in $\mathbf{x}$ $[||\delta\mathbf{x}||/||\mathbf{x}||]$, doing so we get

$$\frac{||\mathbf{x}||}{||H(\mathbf{x})||}\frac{||H(\mathbf{x}+\delta\mathbf{x})-H(\mathbf{x})||}{||\delta\mathbf{x}||}. \tag{1.25}$$

If we take the limit as $||\delta\mathbf{x}|| \to 0$ we get (1.24).

When considering the linear equation

$$A\mathbf{x} = \mathbf{b}. \tag{1.26}$$

with $A \in \mathbb{R}^{n\times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. The conditioning number gives a bound on the inaccuracy of the solution $\mathbf{x}$ in approximation. In the linear case, the conditioning number is defined to be the maximum ratio of the relative error in $\mathbf{x}$ to the relative error in $\mathbf{b}$. If we assume that A is nonsingular and let $\mathbf{e}$ be the error in $\mathbf{b}$ when we approximate $\mathbf{x}$, then the error in the approximation of $\mathbf{x}$ is $A^{-1}\mathbf{e}$. Taking the ratio of the relative errors in $\mathbf{x}$ and in $\mathbf{b}$ yields,

$$\frac{\frac{||A^{-1}e||}{||A^{-1}b||}}{\frac{||e||}{||b||}} = \left(\frac{||A^{-1}e||}{||e||}\right)\left(\frac{||b||}{||A^{-1}b||}\right). \tag{1.27}$$

So we would like to find the max of 1.27 over all non zero $\mathbf{e}$ and $\mathbf{b}$.

$$max_{\mathbf{e},\mathbf{b}\neq 0}\left\{\left(\frac{||A^{-1}\mathbf{e}||}{||\mathbf{e}||}\right)\left(\frac{||\mathbf{b}||}{||A^{-1}\mathbf{b}||}\right)\right\} = \tag{1.28}$$

$$max_{\mathbf{e}\neq 0}\left\{\left(\frac{||A^{-1}\mathbf{e}||}{||\mathbf{e}||}\right)\right\}max_{\mathbf{b}\neq 0}\left\{\left(\frac{||\mathbf{b}||}{||A^{-1}\mathbf{b}||}\right)\right\} = \tag{1.29}$$

$$max_{\mathbf{e}\neq 0}\left\{\left(\frac{||A^{-1}\mathbf{e}||}{||\mathbf{e}||}\right)\right\}max_{\mathbf{x}\neq 0}\left\{\left(\frac{||A\mathbf{x}||}{||\mathbf{x}||}\right)\right\} = \tag{1.30}$$

$$||A^{-1}|| \cdot ||A||. \tag{1.31}$$

This derivation shows that the conditioning number of a matrix A is defined as

$$\begin{cases} \kappa(A) = ||A^{-1}|| \cdot ||A|| & det(A) \neq 0 \\ \infty & det(A) = 0. \end{cases} \tag{1.32}$$

It can be shown immediately that

$$\kappa(A) = ||A^{-1}|| \cdot ||A|| \geq ||A^{-1} \cdot A|| = ||I|| \geq 1 \tag{1.33}$$

The generalization of the conditioning number to non square matrices is also of interest. Let $A \in \mathbb{R}^{m \times n}$ with the range of A denoted by $\mathcal{R}_A$. Define the dual norm of a vector $\mathbf{x} \in \mathbb{R}^n$ as follows

$$||\mathbf{x}||_D = sup\{\mathbf{x}^T z : ||z|| = 1\}. \tag{1.34}$$

Also define

$$\alpha(A) := min\{||A^T\mathbf{x}||_D : \mathbf{x} \in \mathcal{R}_A, ||\mathbf{x}||_D = 1\} \tag{1.35}$$

Demko (1986) [12] defines the generalized conditioning number of the rectangular matrix A as

$$\kappa(A) = ||A|| \cdot \alpha(A). \tag{1.36}$$

It is easily verifiable that if A is a square matrix, then the generalized conditioning number is the same as the conditioning number defined in 1.32. It is also important to notice that when using the l_2 norm as defined in 1.12 then $\alpha(A) = A^\dagger$ where $A^\dagger$ is called the Moore-Penrose inverse. $A^\dagger$ has the following properties:

$$AA^\dagger A = A \tag{1.37}$$

$$A^\dagger AA^\dagger = A^\dagger \tag{1.38}$$

$$(AA^\dagger)^* = AA^\dagger \tag{1.39}$$

$$(A^\dagger A)^* = A^\dagger A \tag{1.40}$$

There have been studies on the conditioning number and the way it relates to neural networks. Saarinen (1993), Sjoberg and Viberg (1997) [46, 51] showed that the Jacobian of neural networks can be ill-conditioned and that this ill-conditioning often happens in specific neural network architectures. This ill-conditioning leads to slow training because the ill-conditioning in the Jacobian causes any optimization algorithm that uses the Jacobian for search directions, to only use partial information of the possible search directions. Saarinen

(1993) [46] proposed changes in architecture to deal with the problem of ill-conditioning in the Jacobian, while Sjoberg and Viberg (1997) [51] presented a new optimization technique that would assist in speeding up training with ill-conditioned neural networks.

Singh *et al.* [50] presented empirical evidence that the conditioning of the weight matrices in neural networks can help in the performance of a neural network on adversarial data. A new regularization term is presented to influence the weight matrices to become orthonormal. They showed positive results in most cases when training models on the MNIST [1] and F-MNIST [1] handwriting data-set.

The goal of this work is to present an argument that well-conditioned weight matrices assist in the classification of noisy data. This work also presents a new regularization term that influences the weight matrices to have better conditioning numbers but not try to make them orthonormal. The remainder of this work is structured as follows: chapter 2 is a presentation of an argument that better-conditioned weight matrices lead to better classification on noisy data, as well as present a new regularization term to influence the weight matrices to have better conditioning numbers. Chapter 3 gives results as well as outline experiments performed to test the regularization term as well as give a description of a benchmark data set and a novel data used in the experiments. Chapter 4 is a discussion and future work, and chapter 5 is the conclusion.

Chapter 2

Training Neural Networks with Noisy Data

2.1 Statement of the Problem

In any data set there is understood to be some element of noise that comes in the data. Assume that we have a data set $\{\mathbf{x_i}\}$ $1 \leq i \leq m$ with m data points. So assume that our data set is of the form

$$\mathbf{x_i} = \mathbf{g_i} + \epsilon_{x_i} \tag{2.1}$$

such that $\mathbf{g_i}$ is the true signal and ϵ_{x_i} is the associated noise. Assume that we have a fully-connected neural network to classify our data set into k classes. So we have

$$F(\mathbf{x_i}, \mathbf{w}) : \mathbf{R}^n \to \mathbf{Z}_k \tag{2.2}$$

$$F(\mathbf{x_i}, \mathbf{w}) = f_v(W^{\langle v \rangle} f_{v-1}(W^{\langle v-1 \rangle} f_{v-2}(W^{\langle v-2 \rangle} f_{v-2}(\dots f_1(W^{\langle 1 \rangle}\mathbf{x^i} + \gamma_i)\dots) + \gamma_{v-2}) + \gamma_{v-1}) + \gamma_v), \tag{2.3}$$

with f_l being an activation function of the users choice and $W^{\langle l \rangle}$ is the weight matrix associated with the l^{th} layer $1 \leq l \leq v$.

Notice that we execute a linear operator in every layer of the neural network before executing an activation function. Let the output of layer l be defined as follows:

$$\mathbf{z_i}^{\langle l \rangle} = W^{\langle 1 \rangle}\mathbf{a_i}^{\langle l-1 \rangle} + \gamma_\mathbf{l}, \tag{2.4}$$

where $\mathbf{a_i}^{\langle l-1 \rangle} = f_{\langle l-1 \rangle}(\mathbf{z_i}^{\langle l-1 \rangle})$ for some activation function $f_{\langle l-1 \rangle}$ of user's choice. For $l = 1$ we will have the convention that $\mathbf{z_i}^{\langle 0 \rangle} = \mathbf{a_i}^{\langle 0 \rangle} = \mathbf{x_i}$

When training this network, since the data has noise in it, we also get a perturbation in the weight vector, the $z's$ and the $a's$. Let

- $\mathbf{h_i}^{\langle l \rangle}$ be the target output of layer l,
- $\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}$ be the perturbation of the output of the l^{th} layer,
- $\mathbf{\Omega}^{\langle \mathbf{l} \rangle}$ be the target weight matrix associated with layer l,
- $\epsilon_{W^{\langle l \rangle}}$ be the perturbation of the weight matrix associated with the l^{th} layer,
- $\mathbf{b_l}$ be the target bias for layer l,
- ϵ_{γ_l} be the perturbation of the bias vector associated with layer l,
- $\zeta_{\mathbf{i}}^{\langle l-1 \rangle}$ be the target activation of layer $l-1$ (by applying an activation to the output of layer $l-1$),
- $\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}$ be the perturbation of the activation vector associated with layer $l-1$.

So we have:

$$\mathbf{z_i}^{\langle l \rangle} = \mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}, \tag{2.5}$$

$$\mathbf{a_i}^{\langle l-1 \rangle} = \zeta_{\mathbf{i}}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}, \tag{2.6}$$

$$W^{\langle l \rangle} = \mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}}, \tag{2.7}$$

$$\gamma_{\mathbf{l}} = \mathbf{b_l} + \epsilon_{\gamma_l}, \tag{2.8}$$

with our target system being

$$\mathbf{h_i}^{\langle l \rangle} = \mathbf{\Omega}^{\langle \mathbf{l} \rangle} \zeta_{\mathbf{i}}^{\langle l-1 \rangle} + \mathbf{b}_l. \tag{2.9}$$

But because of noise our system gives us

$$\mathbf{z_i}^{\langle \mathbf{l} \rangle} = \mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} \tag{2.10}$$

$$= W^{\langle 1 \rangle} \mathbf{a_i}^{\langle l-1 \rangle} + \gamma_{\mathbf{l}} \tag{2.11}$$

$$= (\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})(\zeta_{\mathbf{i}}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}) + \mathbf{b_l} + \epsilon_{b_l}. \tag{2.12}$$

A natural question is can we bound the relative error of $\mathbf{z_i}^{\langle l \rangle}$?

2.2 A Bound on the Relative Error of $\mathbf{z_i}^{\langle l \rangle}$

One can assume with out loose of generality that $\gamma_l = 0$ by subtracting γ_l on both sides of equation (2.4) and analyzing the resulting vector. So by setting $\gamma = 0$ equation 2.4 yields,

$$\mathbf{z_i}^{\langle \mathbf{l} \rangle} = \mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = (\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})(\zeta_\mathbf{i}^{\langle l \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l} \rangle}}), \tag{2.13}$$

with the corresponding target system being,

$$\mathbf{h_i}^{\langle l \rangle} = \mathbf{\Omega}^{\langle \mathbf{l} \rangle} \zeta_\mathbf{i}^{\langle l-1 \rangle}. \tag{2.14}$$

Consider the case when we assume that $\mathbf{\Omega}^{\langle \mathbf{l} \rangle}$ is an invertible square matrix. So by left multiplying both sides of equation (2.14) by the inverse of $\mathbf{\Omega}^{\langle \mathbf{l} \rangle}$ equation (2.14) implies that

$$(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}^{\langle l \rangle} = \zeta_\mathbf{i}^{\langle l-1 \rangle}. \tag{2.15}$$

Similar to above we will assume that $W^{\langle 1 \rangle}$ is an invertable square matrix. When we use the form in equation 2.7 after multiplication we will get,

$$(W^{\langle 1 \rangle})^{-1} = (\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})^{-1} \tag{2.16}$$

Miller (1981) [33] proved the following 2 lemmas and theorem,

Lemma 2.2.1 (Miller)**.** *Let* $\mathbf{G}$ *and* $\mathbf{G} + \mathbf{H}$ *be nonsingular matrices where* $\mathbf{H}$ *is of rank one. Let* $g = tr(\mathbf{H}\mathbf{G}^{-\mathbf{1}})$*. Then* $g \neq -1$ *and*

$$(\mathbf{G} + \mathbf{H})^{-\mathbf{1}} = \mathbf{G}^{-\mathbf{1}} - \frac{1}{1+g}\mathbf{G}{-}\mathbf{1}\mathbf{H}\mathbf{G}^{-\mathbf{1}}.$$

Lemma 2.2.2 (Miller)**.** *Let* $\mathbf{G}$ *and* $\mathbf{G} + \mathbf{H}$ *be invertible matrices. If* $\mathbf{H}$ *has positive rank* r*, then there exist a decomposition of* $\mathbf{H} = \mathbf{E_1} + \mathbf{E_2} + \cdots + \mathbf{E_r}$ *such that* $\mathbf{E_k}$ *has rank one for all* $1 \leq k \leq r$ *and the "partial sums"* $\mathbf{C_{k+1}} = \mathbf{G} + \mathbf{E_1} + \mathbf{E_2} + \cdots + \mathbf{E_k}$ *is nonsingular with* $\mathbf{C_1} = \mathbf{G}$*.*

Theorem 2.2.3 (Miller)**.** *Let* $\mathbf{G}$ *and* $\mathbf{G} + \mathbf{H}$ *be non singular matrices and let* H *have positive rank* r*. Let* $\mathbf{H} = \mathbf{E_1} + \mathbf{E_2} + \cdots + \mathbf{E_r}$ *and* $\mathbf{C_{k+1}} = \mathbf{G} + \mathbf{E_1} + \mathbf{E_2} + \cdots + \mathbf{E_k}$ *as described in Lemma 2.2.2 Then*

$$\mathbf{C_{k+1}^{-1}} = \mathbf{C_k^{-1}} - \nu_\mathbf{k} \mathbf{C_k^{-1}} \mathbf{E_k} \mathbf{C_k^{-1}}$$

for $k = 1, \ldots, r$

with $\nu_{\mathbf{k}} = \frac{1}{tr(\mathbf{C_k^{-1}E_k})}$

and in particular

$$(\mathbf{G} + \mathbf{H})^{-\mathbf{1}} = \mathbf{C_r^{-1}} - \nu_{\mathbf{r}}\mathbf{C_r^{-1}E_kC_r^{-1}}.$$

A simple corollary of Theorem 2.2.4 is now presented.

Corollary 2.2.4. *Let* $\mathbf{G}$, $\mathbf{G} + \mathbf{H}$, E_k, *and* C_{k+1} *be as defined in Theorem 2.2.4. Then*

$$((\mathbf{C_{k+1}})^{-\mathbf{1}} = \mathbf{G^{-1}} + \mathbf{\Gamma}$$

for some matrix Γ *that will have dependency on* E_k

Proof. We proceed by induction. When $k = 1$

$$\mathbf{C_2^{-1}} = \mathbf{G^{-1}} - \nu_{\mathbf{1}}\mathbf{G^{-1}E_1G^{-1}}.$$

Assume true for arbitrary $k = m - 1$

$$\begin{aligned}\mathbf{C_{m+1}^{-1}} &= \mathbf{C_m^{-1}} - \nu_{\mathbf{k}}\mathbf{C_m^{-1}E_mC_m^{-1}} \\ &= \mathbf{G^{-1}} + \mathbf{\Gamma_1} - \nu_{\mathbf{k}}\mathbf{C_m^{-1}E_mC_m^{-1}}.\end{aligned}$$

Choosing $\mathbf{\Gamma}$ to be $\mathbf{\Gamma_1} - \nu_{\mathbf{k}}\mathbf{C_m^{-1}E_mC_m^{-1}}$ we obtain the result. □

Returning to equation (2.17) we see that since the only matrix over the real numbers with rank 0 is the 0 matrix we have,

$$(\mathbf{\Omega^{\langle 1\rangle}} + \epsilon_{W^{\langle l\rangle}})^{-1} = (\mathbf{\Omega^{\langle 1\rangle}})^{-1} + \mathbf{\Gamma} \tag{2.17}$$

Where Γ will have dependency on the decomposition of $(\ \mathbf{\Omega^{\langle 1\rangle}})^{-1}$ and the matrix $\epsilon_{W^{\langle l\rangle}}$.

Definition 2.2.5.

$$\epsilon_{\mathbf{\Omega^{\langle 1\rangle}}} := \Gamma$$

We are interested in the relative error of $\mathbf{z_i}^{\langle \mathbf{l} \rangle}$. Notice that

$$\mathbf{z_i}^{\langle l \rangle} = \tag{2.18}$$

$$\mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = \tag{2.19}$$

$$((\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})(\zeta_\mathbf{i}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}) \tag{2.20}$$

Left multiplying equation 2.20 and 2.21 by the inverse of $(\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})$ results in

$$((\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}}))^{-1}(\mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}) = \tag{2.21}$$

$$((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})(\mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}) = \tag{2.22}$$

$$(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\mathbf{h_i}^{\langle l \rangle} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\mathbf{h_i}^{\langle l \rangle} + ((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = \tag{2.23}$$

$$\zeta_\mathbf{i}^{\langle l-1 \rangle} + \epsilon_{\mathbf{z_i}^{\langle \mathbf{l-1} \rangle}} \tag{2.24}$$

Solving for $((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}$ gives the result

$$((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = \tag{2.25}$$

$$(W^{\langle 1 \rangle})^{-1}\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = \tag{2.26}$$

$$\zeta_\mathbf{i}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\mathbf{h_i}^{\langle l \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\mathbf{h_i}^{\langle l \rangle} \tag{2.27}$$

We will now invert $(W^{\langle 1 \rangle})^{-1}$ to put it back on the right side of the equal sign

$$\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}} = \tag{2.28}$$

$$((W^{\langle 1 \rangle})(\zeta_\mathbf{i}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\mathbf{h_i}^{\langle l \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\mathbf{h_i}^{\langle l \rangle}) \tag{2.29}$$

In order to find a bound on the relative error, we enter norm space.

$$\left\|\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}\right\| = \tag{2.30}$$

$$\left\|((W^{\langle 1 \rangle})(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| \leq \tag{2.31}$$

$$\left\|((W^{\langle 1 \rangle})\right\| \left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} + \epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| \leq \tag{2.32}$$

$$\left\|((W^{\langle 1 \rangle})\right\| \left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) = \tag{2.33}$$

$$\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle} + \epsilon_{W^{\langle l \rangle}})\right\| \left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) = \tag{2.34}$$

$$\left\|((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1}\right\| \left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) = \tag{2.35}$$

$$\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) \left(((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}) - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \right) ((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1} \right\|$$

$$\left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) = \tag{2.36}$$

$$\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} ((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1}\right\|$$

$$\left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) \leq \tag{2.37}$$

$$\left(\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\| + \left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\right\| \left\|((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1}\right\| \right)$$

$$\left(\left\|(\zeta_{\mathbf{i}}{}^{\langle l-1 \rangle} - (\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \mathbf{h_i}{}^{\langle l \rangle})\right\| + \left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) \tag{2.38}$$

By subtracting equation 2.37 and 2.40 by equations $\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\right\| \left\|((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1}\right\|$, using equation 2.15, and dividing by appropriate terms, 2.37-2.40 is equivalent to the following inequality:

$$\left\|((\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1} + \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}})^{-1}\right\| \left(\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) \leq \tag{2.39}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|}{1 - \left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle}) \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\right\|} \left(\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}} \mathbf{h_i}{}^{\langle l \rangle}\right\| \right) \tag{2.40}$$

To analyze the relative error of $\mathbf{z_i}^{\langle \mathbf{l} \rangle}$ we divide both sides by the norm of $\mathbf{h_i}^{\langle \mathbf{l} \rangle}$

$$\frac{\left\|\epsilon_{\mathbf{z_i}^{\langle \mathbf{l} \rangle}}\right\|}{\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|} \leq \tag{2.41}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}} - \epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\mathbf{h_i}^{\langle l \rangle}\right\|}{\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|}\right) \leq \tag{2.42}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}\right\| + \|\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|\left\|\mathbf{h_i}^{\langle l \rangle}\right\|}{\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|}\right) \leq \tag{2.43}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}\right\|}{\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|} + \frac{\|\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|\left\|\mathbf{h_i}^{\langle l \rangle}\right\|}{\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|}\right) = \tag{2.44}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}\right\|}{\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\|\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|} + \frac{\|\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|\left\|\mathbf{h_i}^{\langle l \rangle}\right\|}{\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\|\left\|\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|}\right) \leq \tag{2.45}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}\right\|}{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\mathbf{h_i}^{\langle \mathbf{l} \rangle}\right\|} + \frac{\|\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}{\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\|}\right) = \tag{2.46}$$

$$\frac{\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\|\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\right\|}{1-\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}\left(\frac{\left\|\epsilon_{\mathbf{a_i}^{\langle \mathbf{l-1} \rangle}}\right\|}{\left\|\zeta_{\mathbf{i}}^{\langle l-1 \rangle}\right\|} + \frac{\|\epsilon_{\mathbf{\Omega}^{\langle \mathbf{l} \rangle}}\|}{\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\|}\right) \tag{2.47}$$

We notice that one thing that we can do to try to make the relative error of $\mathbf{z_i}^{\langle \mathbf{l} \rangle}$ on the same magnitude as the relative error of $\zeta_{\mathbf{i}}^{\langle \mathbf{l} \rangle}$ and $(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}$ is to make the conditioning number of $(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})$ to be as small as possible.

2.3 A New Regularization Term

Let L_{class} be the original loss function associated with a neural network $F(\mathbf{x}, \Theta)$. We will make the strong assumption that $\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\| \leq \left\|W^{\langle l \rangle}\right\|$ and also $\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\right\| \leq \left\|(\mathbf{W}^{\langle \mathbf{l} \rangle})^{-1}\right\|$. This assumption equates to the error term associated with the weights and the error term associated with inverse, having a positive correlations with the weight matrix and inverse. In the case of the l_2 operator norm, the error term shifts the singular values of the weight matrix and inverse matrix in a significant way. We suggest a regularization term to be added to the system's loss function to penalize large conditioning number of $W^{\langle l \rangle}$, and by our assumption,

this penalizes the conditioning number of $(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})$

$$L_{cond} = \lambda_l \kappa(W^{\langle l \rangle}), \tag{2.48}$$

Where λ_l are hyperparameters to the system and can differ from layer to layer. We only analyzed the case in which the weight matrix is square. This means that we only add regularization to a layer that meets our conditions. To meet this, we must have 2 layers $l, l+1$ of $F(\mathbf{x}, \Theta)$ that are fully connected and both layers have the same number of neurons in it. In this case, $\mathbf{W}^{\langle \mathbf{l+1} \rangle}$ is a square matrix, and this is the matrix that we influence to have a good conditioning number. Our total lost is now defined as follows:

$$L_{total} = L_{class} + L_{cond} \tag{2.49}$$

Chapter 3

Experiments and Results

3.1 Atrial Fibrillation

According to Lip *et al.* (2016)[32] atrial fibrillation (AF) is a disorder of the hearts electrical conduction system that leads to fast and irregular heart rhythms. It is the most common sustained cardiac arrhythmia, occurring in 12% of the general population according to Deng and Yu (2014) [14]. According to the Center of Disease Control and Prevention, there are an estimated 2.7-6.1 million people in the United States that live with AF and as the population grows older with advancements in medicine, the number of people living with AF is expected to increase. Adults that are 40 years or older have a one in four risk of developing AF in a lifetime. The complications associated with AF include a four- to fivefold increased risk of stroke [63] and a two- to threefold increased risk of heart failure [60]. The Symptoms of AF include palpitations, fatigue, dizziness, light-headedness, and dyspnoea but many patients are asymptomatic [47].

Diagnosing AF is often done through the readings of an Electrocardiogram (ECG). An ECG is a noninvasive test where electrodes are placed on the skin to detect the electrical activity of the heart. The heart works as a pump to deliver blood through out the body. Each beat of your heart is triggered by an electrical impulse that is generated at the top of the heart in the sinoatrial node and then travels to the atrioventricular node. Then the signal travels through the bundle of His to the ventricles at the bottom of the heart. An electrocardiogram records the timing and strength of these signals as they travel through your heart. Figure 3.1 gives a diagram of the heart. The ECG features that are associated with AF are complete irregularity of the RR intervals, an absence of P-waves and coarse or

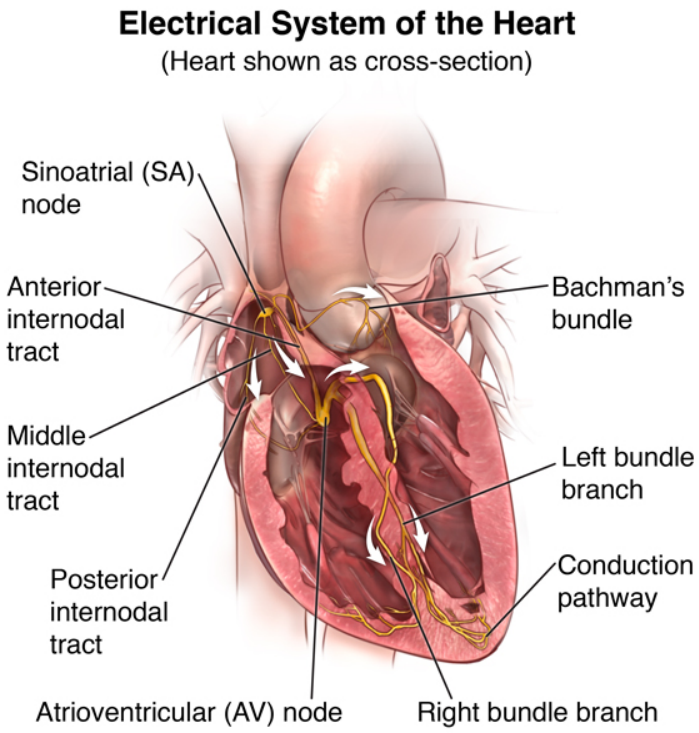


Figure 3.1: Image source: John Hopkins University

fine fibrillation waves in the baseline [32]. A comparison between a patient with a normal heart beat and a patient with AF is given in Figure 3.2.

The diagnosis of AF has also proven to be difficult when absent from symptoms because AF is often paroxysmal [32]. This means that the irregularities in the heart can be short-lived and many times overlooked. Clifford *et al.* (2017) [9] point out that AF detection remains problematic for numerous reasons. The limitations of other studies to classify AF include the only classification of normal and AF because many non-AF rhythms exhibit irregular RR intervals that may be similar to AF, picking data that is often clean and can't be applied in real life scenarios, and only using data from a small number of patients. The following challenge was given in 2017 to encourage research in trying to classify AF reliably.

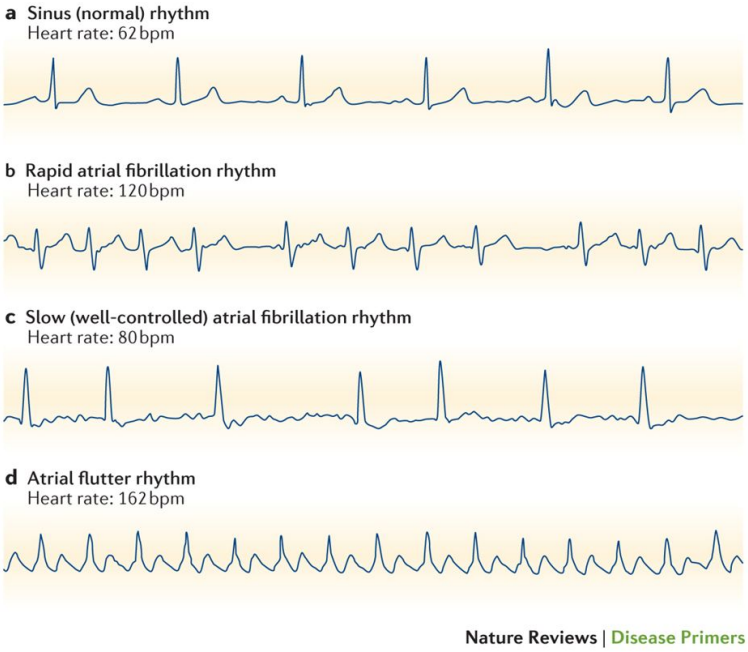


Figure 3.2: Comparison of Normal and AF ECG images. Image source: Li and Yuan (2017)

3.2 PhysioNet Challenge

The PhysioNet Challenge [9] data-set consists of a total of 12,186 ECG recordings that were donated by AliveCor. Each recording was from a self-administered ECG reading which leaves room for user error. The recordings ran for a mean of 32.5 seconds, a minimum length of 9 seconds and a maximum length of 61 seconds. The data is stored at 300hz and converted into MATLAB V4 files (each including a .mat file containing the ECG and a .hea file containing the waveform information.) Four classes of data were considered for this challenge: normal rhythm, AF rhythm, other rhythms, and noisy recordings. Initially, 10% of the labels were visually verified by the challenge organizers. They found that there were discrepancies in the labels, and with a "Mid -Challenge Bootstrap" algorithm along with the help of 8 independent expert ECG analysts the challenge organizers the final version of the labels were agreed upon. The test data was distributed to the public and contained 8,528

	Normal	AF	Other	Noisy	Total
Normal	N_n	N_a	N_o	N_p	$\sum N$
AF	A_n	A_a	A_o	A_p	$\sum A$
Other	O_n	O_a	O_o	O_p	$\sum O$
Noise	P_n	P_a	P_o	P_p	$\sum P$
Total	$\sum n$	$\sum a$	$\sum o$	$\sum p$	

Table 3.1: Definition of parameters for scoring used in equations. Source: Clifford *et al.* (2017)

records with similar features as the entire data-set, and the test data has been kept private and contains 3,658 records with similar features as the entire data-set. The training data contained 5,154 ECG readings of normal heartbeats, 771 readings of AF, 2,557 readings that were other rhythm types, and 46 readings that are classified as noise.

The data set is skewed heavily towards normal ECG signals. Because of this, the scoring in this challenge was done with a F_1 measure. In the following are the definition of the four types of F_1 were taken from Clifford *et al.* (2017) [9] and Table 3.1 gives the definition of each parameter.

$$Normal : F_{1n} = \frac{2N_n}{\sum N + \sum n} \tag{3.1}$$

$$AF : F_{1a} = \frac{2A_a}{\sum A + \sum a} \tag{3.2}$$

$$Other : F_{1o} = \frac{2O_o}{\sum O + \sum o} \tag{3.3}$$

$$Noise : F_{1p} = \frac{2P_p}{\sum P + \sum p} \tag{3.4}$$

The final score for an entry in the challenge was

$$F_1 = \frac{F_{1n} + F_{1a} + F_1 o}{3} \tag{3.5}$$

The winner of this challenge was the work described in Teijeiro *et al.* [57]. The final F_1 score was .83. In this work extensive preprocessing and area expertise was used to extract clinical meaningful features for classification. Hannun *et al.* (2019) [17] reported an F1 score on the

Physionet challenge of .83 as well. The difference with the two approaches is substantial. In the approach in Hannun *et al.* (2019) [17] they use no area expertise and no prepossessing. This is an end-to-end deep learning approach to deep learning that has become more popular in the previous years. The method used a deep convolutional neural network that takes data of any length and no other ECG features. The network have shortcut connections similar to that of Residual Neural networks. The network has 34 layers that consist of 16 residual blocks with 2 convolutional layers per block. The convolutional layers all have filter length 16. Each convolutional layer have $32 \cdot 2^k$ filters where k starts at zero and increases by one every 4th residual block. Batch noramlization was applied to every convolutional layer. Every other residual block has a maxpooling layer to sub sample the data by a factor of 2. Finally dropout with $p = 0.2$ was added between the convolution layers and after the activation function was applied. The final layer was a fully connected layer with a soft-max activation to predict the 4 classes. Figure 3.3 shows the diagram given in the original work of the model.

To test our new regularization term we made a few modifications to the model in Hannun *et al.* (2019) [17]. We first added another residual block to the network. Adding the block seemed to improve on the result presented in the original paper. We also added another dense layer before the output layer that has the same dimensions as the output layer. The activation function used on this layer was relu. This last modification gives us a square weight matrix that we added our regularization term to. We added another skip connection between the two dense layers. Figure 3.6 gives a diagram of the proposed neural network.

Figure 3.3: Deep Neural Network Model for ECG Classification

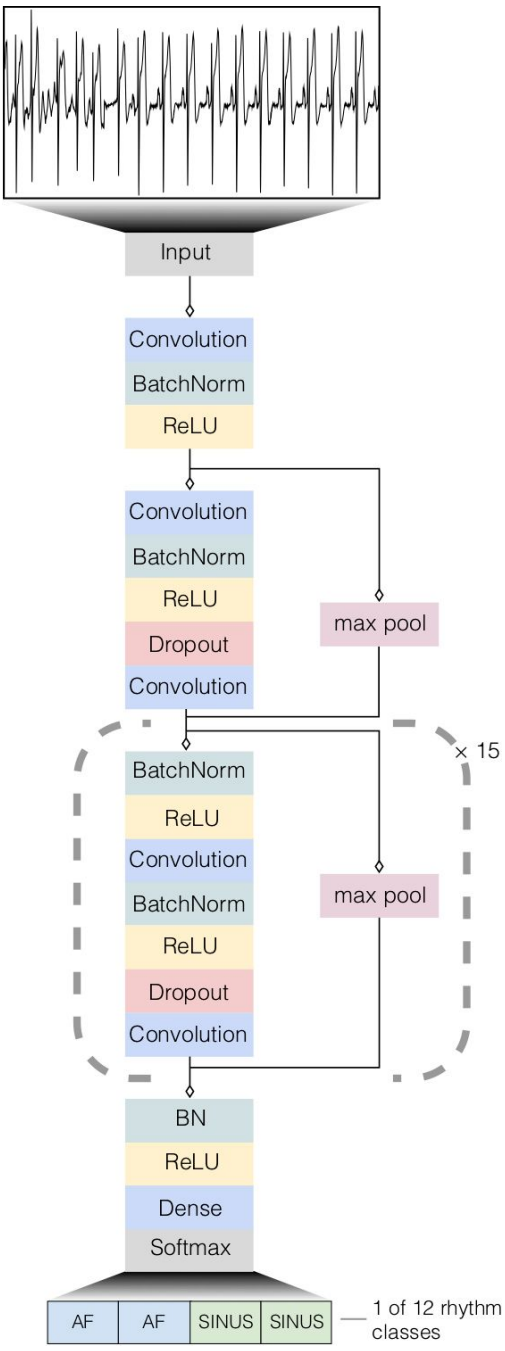
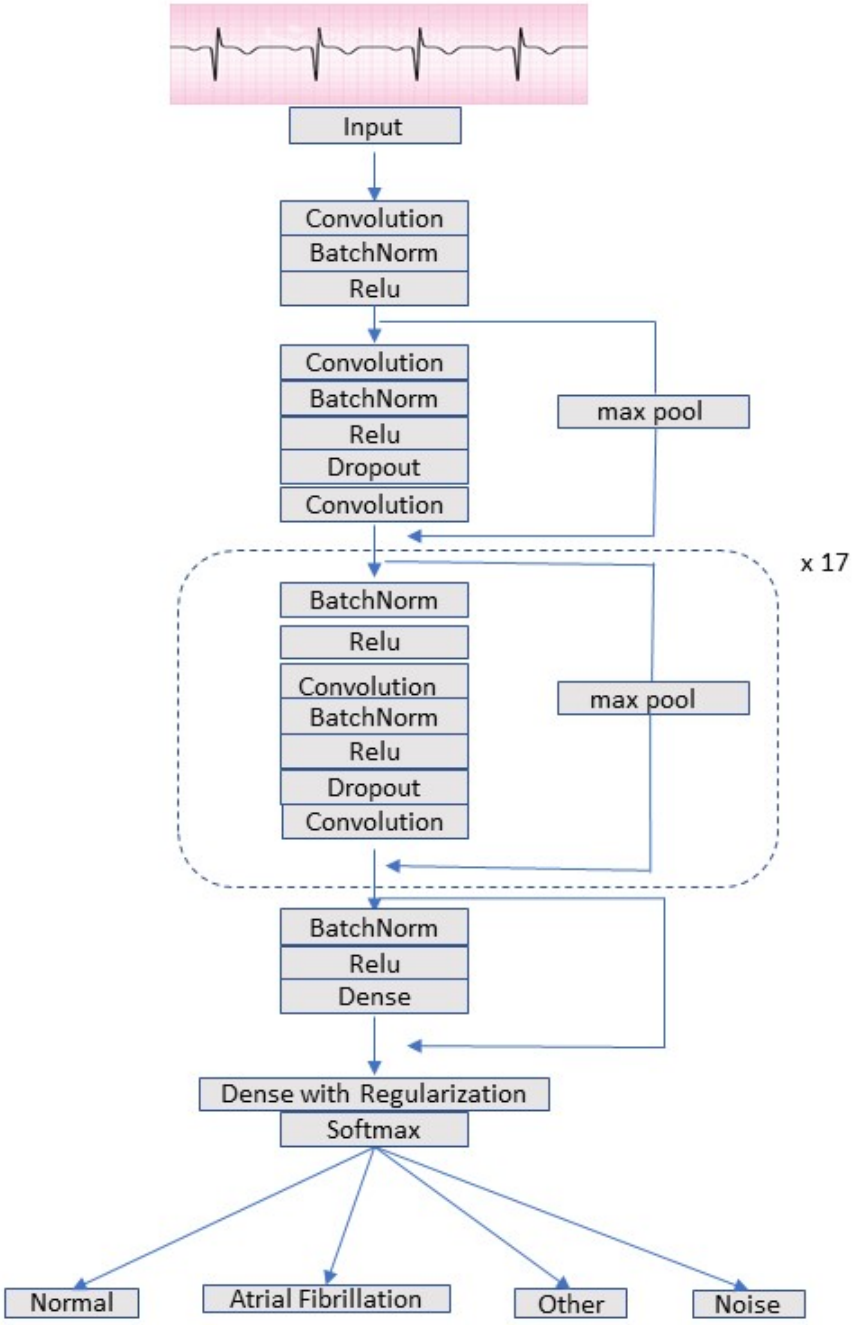


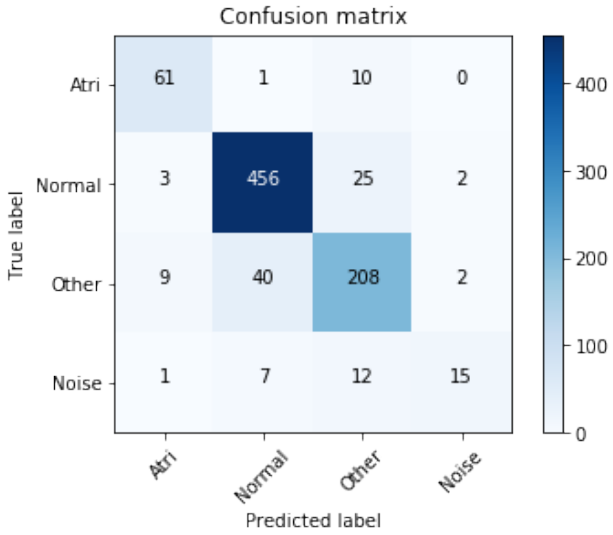
Image source: Image taken from Hannun *et al.* (2019)

Figure 3.4: Diagram of Proposed Deep Neural Network Model for ECG Classification



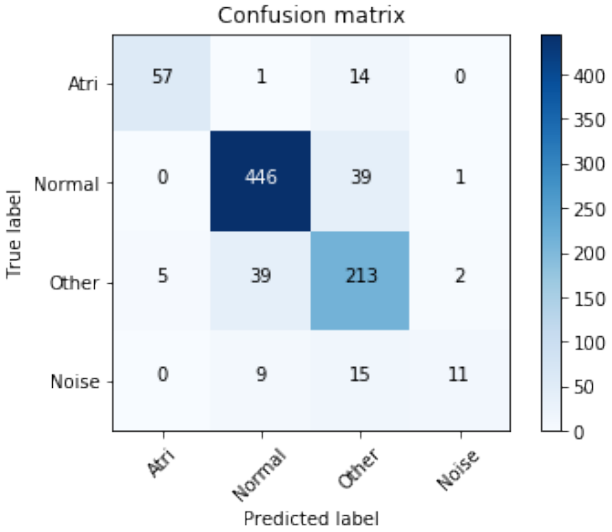
To train the model we used 90% of the Physionet training data and 10% for the development set. The norm that we choose was the l_2 norm. We only regularized the output layer, so we used $\lambda = .1$. We saved the model after every epoch for the regularized model and also for the model that is not regularized, and took the best model in both instances. To measure the performance of the models, we plotted the confusion matrix, we took the F_1 as defined in the challenge on the development set and also submitted the models to be tested on the private data from the challenge.

Figure 3.5: Confusion Matrix for Model With Regularization



When taking the combined F_1 score as done in the challenge on the development we got .8554. Using the same model without regularization we get a combined F_1 score of .8496. Tables 3.2 and 3.3 gives different statistics for the different labels of the data-set. Figure 3.5 and 3.6 gives the confusion matrices for the model with regularization and without regularization.

Figure 3.6: Confusion Matrix for Model With No Regularization



	Precision	Recall	$F_1 score$	Support
Normal	.901	.918	.909	486
AF	.919	.792	.851	72
Other	.758	.822	.789	259
Noise	.786	.314	.449	35

Table 3.2: Statistics from model without regularization

	Precision	Recall	$F_1 score$	Support
Normal	.905	.938	.921	486
AF	.824	.847	.836	72
Other	.816	.803	.809	259
Noise	.789	.429	.556	35

Table 3.3: Statistics from model with regularization

3.3 Novel Telemetry Data

The World Health Organization [7] reported that in 2017 an estimated 219 million cases of malaria occurred worldwide with a 95% confidence interval of 203-262 million. Fifteen countries in sub-Saharan Africa and India account for 80% of the global malaria burden. In 2017 an estimated 435,000 deaths were caused by malaria with about 61% of the deaths

being from children ages 5 and below. In 2017 an estimated 3.1 billion dollars was invested in malaria control and elimination.

Malaria is a disease that affects humans and other animals and is caused by parasites that are transmitted to humans through the bites of infected mosquitoes. Malaria is caused by *Plasmodium* parasites. The parasites are spread through the female *Anopheles* mosquitoes. These mosquitoes mainly bite from dusk to dawn. Five parasite species cause human malaria:

- *Plasmodium* falciparum
- *Plasmodium* vivax
- *Plasmodium* malariae
- *Plasmodium* ovale
- *Plasmodium* knowlesi

Plasmodium falciparum and *Plasmodium* vivax are the most common according to the World Health Organization and *Plasmodium* falciparum is the most deadly. *Plasmodium* knowlesi is a parasite of old world monkeys and was thought to be rare in humans until recent reports [10, 61]. Partial immunity is developed over multiple exposures, which reduces the risk of severe complications in malaria. This is one explanation of why the highest mortality rate for malaria is with children under the age of 5.

Symptoms in a non-immune individual usually appear 7 days or more after the infectious bite. The first symptoms of malaria symptoms include fever, headache, chills, and vomiting. These symptoms may be mild and hard to recognize as malaria. In children, one or more of the following symptoms are common with a severe malaria infection according to the World Health Organization: severe anemia, respiratory distress concerning metabolic acidosis, or cerebral malaria. When partial immunity is present malaria can also become asymptomatic.

As described by the Center for Disease Control and Prevention [41] the life cycle of malaria involves cyclical infection of humans and *Anopheles* mosquitoes. The cycle begins

with the mosquito injecting a human with anticoagulant saliva together with the parasite. The parasite immediately migrates to the liver where it grows and multiplies for an estimated 5-7 days in humans. After this, the parasites come out of the liver, and the blood stage begins. The blood stage is when the onset of symptoms happens. Parasites began to invade red blood cells where they grow and multiply destroying the red blood cell and releasing parasites to continue this process. During this process, the parasites become gametocytes, which occur in male and female forms. Another mosquito comes and bites the infected human. This makes make the mosquito pick up both male and female forms of the parasite. The parasites mate in the mosquito's gut, and after 10-18 days the parasite migrates to the mosquito's saliva glands in the form of sporozoites to began the cycle again. A diagram of this cycle is given in Figure 3.7

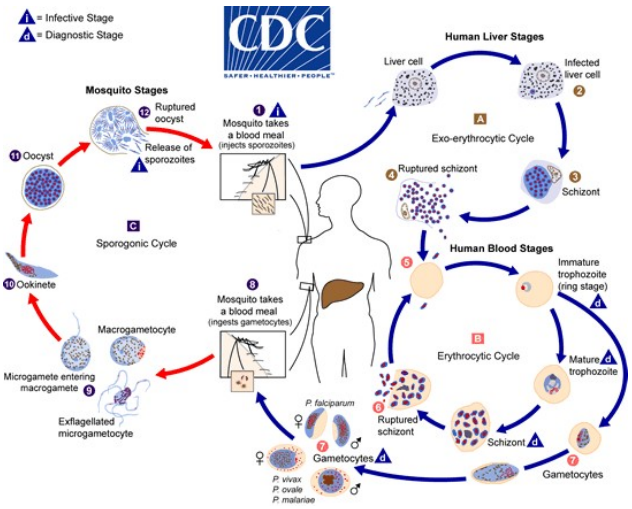


Figure 3.7: Malaria Life Cycle: Source: The Center for Disease Control

The World Health Organization reports that early diagnosis and treatment of malaria reduces the disease and prevents death. The best-known treatment for malaria is artemisinin-based combination therapy (ACT) [5, 37] which are combinations of an artemisinin derivative and another structurally unrelated and more slowly eliminated antimalarial. To diagnose malaria the World Health Organization recommends microscopy or rapid diagnostic tests. Polymerase chain reaction (PCR) is also a method to detect malaria and has been reported

to be slightly more sensitive than smear microscopy, but because the results from PCR take longer than microscopy, PCR is usually used as a way to confirm that a patient has malaria and not as a diagnostic tool. Current techniques that are used by clinicians to diagnose malaria are used after symptoms arise [55]. There has been considerable interest in the study of the liver stage of malaria [44]. This stage is clinically silent, so there is no presence of symptoms yet. Diagnostic tools for the liver stage will change the landscape for treating malaria and give rise to the opportunity of treating malaria before symptoms are present.

2012 the Malaria Host-Pathogen Interaction Center (MaHPIC) was established to characterize host-pathogen interactions during malaria infections of non-human primates (NHP) and clinical studies. MaHPIC collects and analyzes comprehensive data on how a Plasmodium parasite infection produces changes in host and parasite gene expression, proteins, lipids, metabolism, and the host immune response. The experiment design for three novel malaria experiments provided in [3] are now given

Experiment 30: A Pilot experiment for Macaca mulatta infected with Plasmodium knowlesi Malayan strain sporozoites to produce and integrate clinical, hematological, parasitological, omics, telemetric and histopathological measures of acute primary infection.

Telemetry devices (DSI, model L11) were surgically implanted in two malaria-naive, male *Macaca mulatta* non-human primates that were approximately 3 years old. The telemetry devises had blood pressure sensors, accelerometer, temperature sensors and electrocardiogram (ECG) leads. After a resting period of two weeks, the telemetry devices were turned on, and physiological data was captured continuously. In this experiment, ECG and blood pressure were collected at a rate of 1kHz. Temperature was collected at 1hz, and three perpendicular axes of accelerometer data were collected all at a rate of 10hz. Two weeks after the devices were turned on the *mulattas* were inoculated intravenously with cryopreserved *P. knowlesi* Malayan strain salivary gland sporozoites, obtained from Anopheles dirus infected

with parasites from the Pk1A+ clone. The *P. knowlesi* sporozoites were produced, isolated and cryopreserved at the Centers for Disease Control and Prevention. After inoculation, the macaques were profiled for clinical, hematological, parasitological, immunological, functional genomic, lipidomic, proteomic, metabolomic, telemetric and histopathological measurements.

The experiment was designed for pathology studies, with terminal necropsies 8 days apart. The anti-malarial drug artemether was sub curatively administered selectively to one subject during the primary parasitemia to suppress clinical complications. Capillary blood samples were collected daily for the measurement of complete blood counts, reticulocytes, and parasitemias. Capillary blood samples were collected every other day to obtain plasma for metabolomic analysis. Venous blood and bone marrow samples were collected at five timepoints for functional genomic, proteomic, lipidomic, and immunological analyses. Physiological data were continuously captured via telemetry. The Emory University Institutional Animal Care approved the experimental design and protocols for this study and Use Committee (IACUC) and the MRMC Office of Research Protection Animal Care and Use Review Office (ACURO). A figure for experimental design is given in Figure 3.8.

Experiment 06: Macaca mulatta infected with Plasmodium knowlesi sporozoites to produce and integrate clinical, hematological, parasitological, omics, telemetric, and histopathological measures of acute primary infection.

Telemetry devices (DSI, model L11) were surgically implanted in four malaria-naive, male *Macaca mulatta* non-human primates that were approximately 5 years old. The telemetry devises had blood pressure sensors, accelerometer, temperature sensors and electrocardiogram (ECG) leads. After a resting period of two weeks, the telemetry devices were turned on, and physiological data was captured continuously. In this experiment ECG was collected at a rate of 1kH, blood pressure was collected at 500hz, temperature was collected at 1hz, and three perpendicular axes of accelerometer data were collected all at a rate of 10hz.

Two weeks after the devices were turned on, the macaques were inoculated intravenously with cryopreserved P. knowlesi Malayan strain salivary gland sporozoites, obtained from Anopheles dirus infected with parasites from the Pk1A+ clone and previously tested in E30 for their infectivity of macaques. The sporozoite stocks used were produced, isolated and cryopreserved at the Centers for Disease Control and Prevention, and then stored at Yerkes. After inoculation clinical, hematological, parasitological, immunological, functional genomic, proteomic, and metabolomic measurements were taken throughout the experiment. The experiment was designed with pathology studies and thus terminal necropsies, which were scheduled at the log phase of the infections or at the peak of parasitemias. Capillary blood samples were collected daily for the measurement of complete blood counts, reticulocytes, and parasitemias. Capillary blood samples were collected every other day to obtain plasma for metabolomics analyses. Venous blood and bone marrow samples were collected at five timepoints for functional genomic, targeted proteomic, targeted metabolomics, and immunological analyses. The Emory University Institutional Animal Care approved the experimental design and protocols for this study and Use Committee (IACUC) and the MRMC Office of Research Protection Animal Care and Use Review Office (ACURO). A figure for experimental design is given in Figure 3.9.

Experiment 07A and 07B: Macaca fascicularis infected with Plasmodium knowlesi sporozoites to produce and integrate clinical, hematological, parasitological, omics, telemetric and histopathological measures of acute primary infection.

Experiment 07A was an experiment where a cohort was inoculated with *P. knowlesi* sporozoites but for unknown reasons no blood-stage parasitemia ever occurred. Because of this, the cohort was reinoculated with cryopreserved P. knowlesi sporozoites. So E07A refers to the data collected from the failed inoculation and E07B refers to the successful reinoculation.

Telemetry devices (DSI, model L11) were surgically implanted in 7 malaria-naive, male long-tailed *Macaca fascicularis* non-human primates that were approximately 5 years old. The telemetry devises had blood pressure sensors, accelerometer, temperature sensors and electrocardiogram (ECG) leads. After a resting period of two weeks, the telemetry devices were turned on, and physiological data was captured continuously. In this experiment ECG was collected at a rate of 1kH, blood pressure was collected at 500hz, temperature was collected at 1hz, and three perpendicular axes of accelerometer data were collected all at a rate of 10hz. To save battery life after the failed inoculation from E07A the devices were turned off to preserve battery life. One *Macaca mulatta* was introduced to the experiment between E07A and E07B and had no telemetry implant inserted in it. At the start of E07B, the telemetry devices were turned back on. Ten days after, all animals were inoculated intravenously with cryopreserved P. knowlesi Malayan strain salivary gland sporozoites, obtained from Anopheles dirus infected with parasites from the Pk1A+ clone and previously tested in E30 for their infectivity of macaques. All experiments were conducted similarly to E30 and E06. A diagram of Experiment E07 is given in Figure 3.11. A diagram of the telemetry hardware and set up is given in Figure 3.10

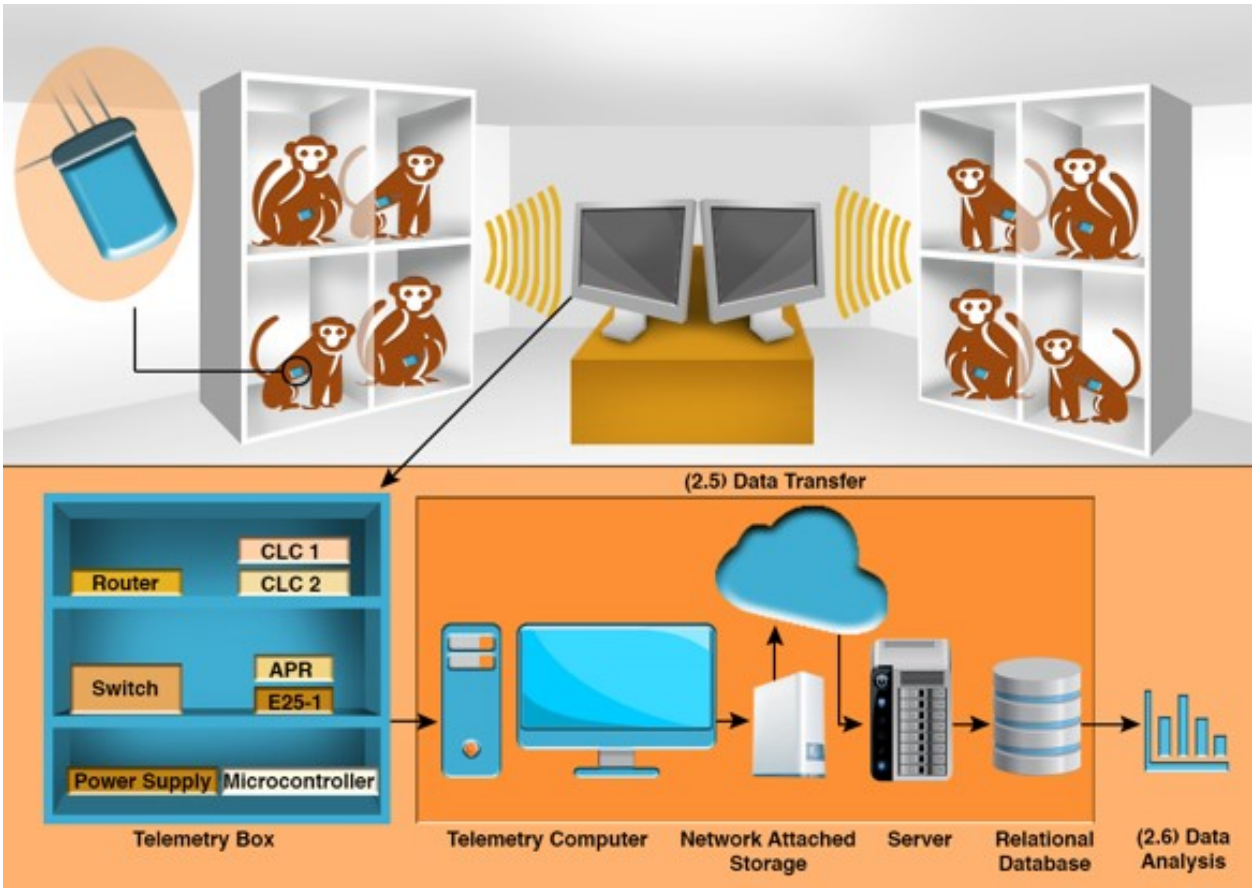
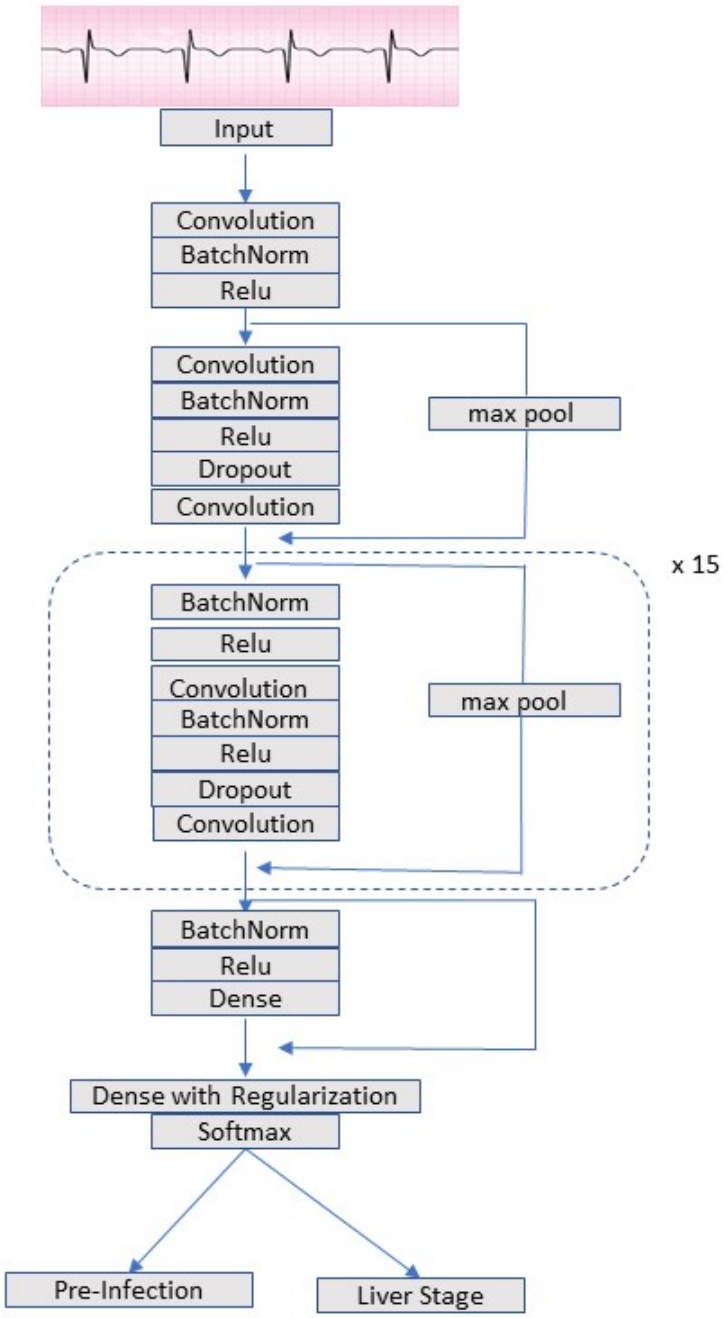


Figure 3.10: Telemetry Set Up. Source: Biomathematics Research Group

We considered all thirteen subjects from E30, E06, and E07 that had telemetry implants present during the experiments. We only considered E07B because this was the infection that had a successful blood stage. We are interested in getting a classification between pre-infection and the liver stage of the P. knowlesi infection from the three experiments. We considered only ECG data from each experiment. We considered four days before inoculation and four days after inoculation for all of the experiments and all of the animals. Each day was segmented into hourly intervals of data. Finally, we segmented the hour intervals into 10-second intervals, and these are the observations that we feed to our model to try to get a classification. We used the model that is similar to that in Figure 3.4 with $\lambda = .1$ the same as in the 2017 Physionet challenge. We modified to this data set. Figure 3.12 gives a diagram of the model.

Figure 3.12: Diagram of Proposed Deep Neural Network Model for Telemetry Classification

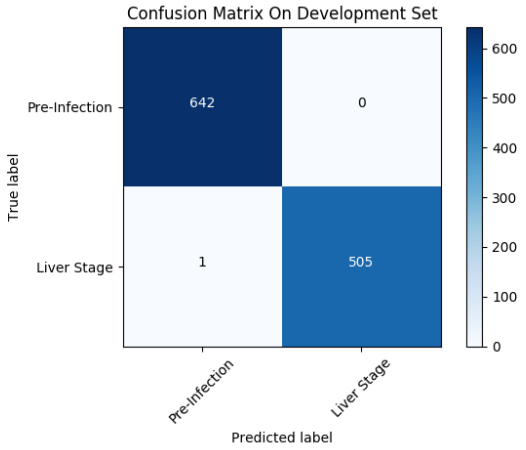


We trained our model using 4 days of hour 23 data before inoculation, and 4 days of liver stage hour 23 for all of the subjects that had telemetry implants. The data was split into 10-sec increments which gave a total of 11,481 observations. We used a random 90% of these observations to train a neural network and a random 10% as the development set to see if we can get classification between pre-infection and liver stage. After 7 epoch an accuracy of 99% was achieved on the development set with the top performing model. The confusion matrix and F_1 statistics for the development set is given in Table 3.4 and Figure 3.13

	Precision	Recall	F_1 score	Support
Pre-Infection	.998	1.000	.999	642
Liver Stage	1.000	.998	.999	506

Table 3.4: Statistics from Development Set (10% of Hour 23)

Figure 3.13: Confusion Matrix for Development Set (10% of Hour 23)



After training on hour 23 we tested the model on each hour to see which hours have the best classification power. The confusion matrices and F_1 statistics for these experiments are found in the appendix. The code that was created for this classification problem is also found in the appendix.

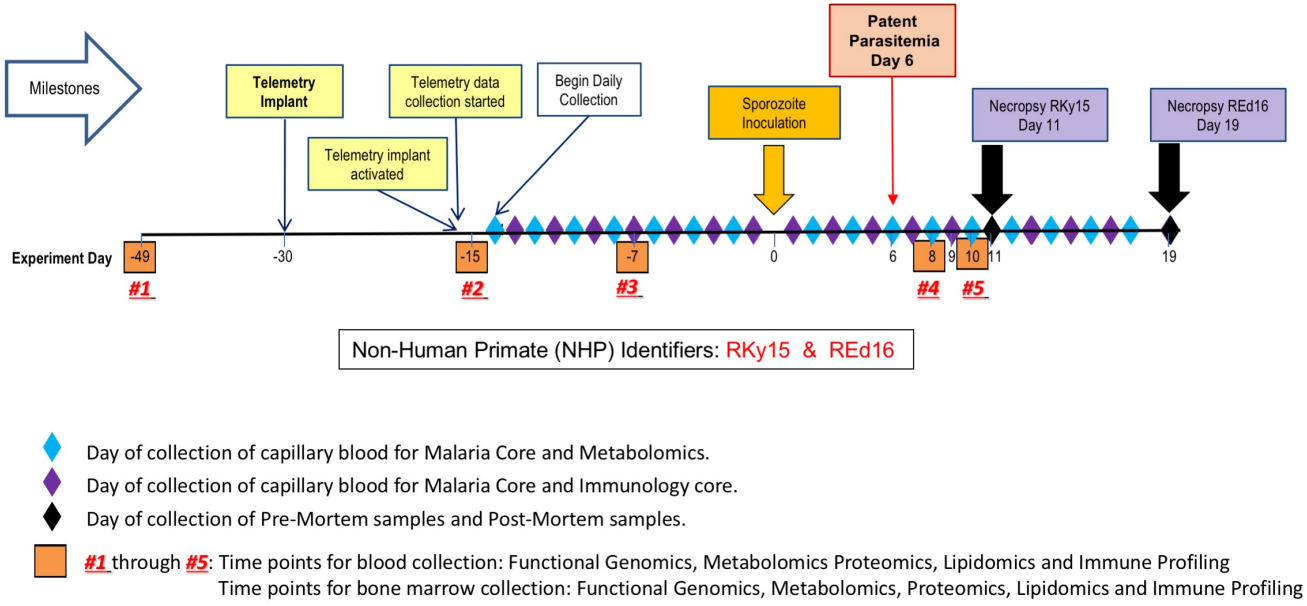
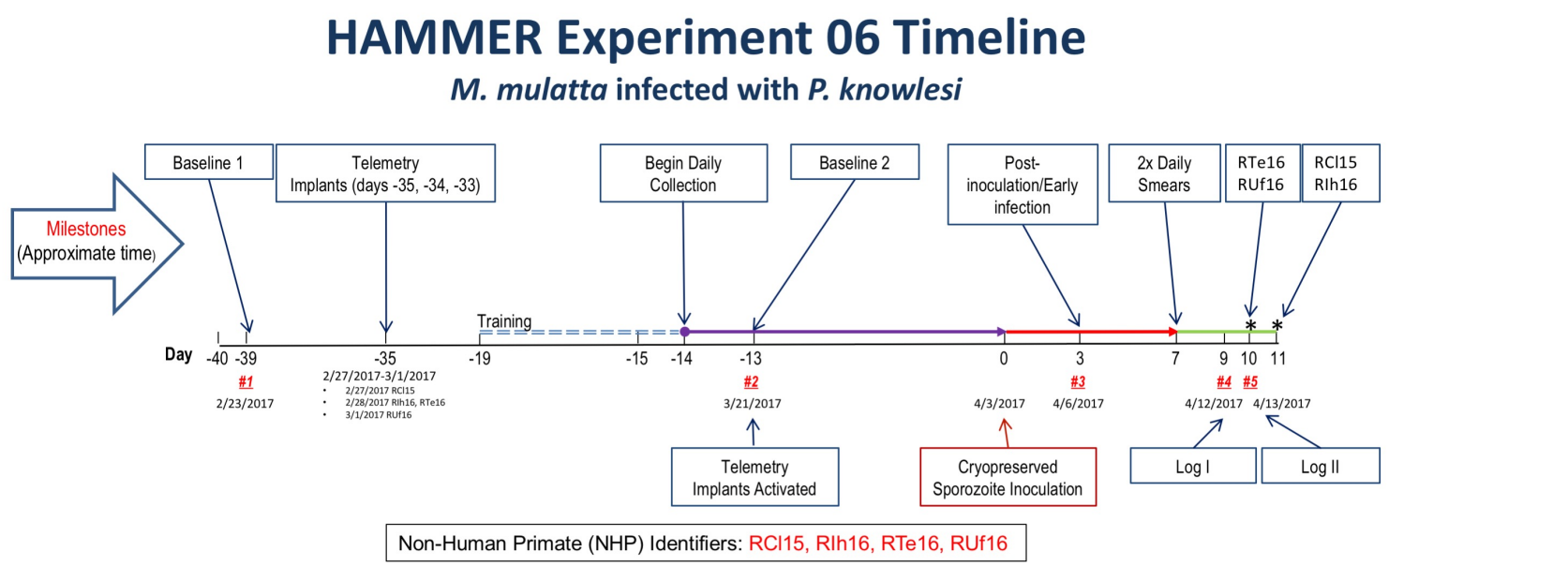


Figure 3.8: E30 Experiment Set Up: Source: Aurrecoechea C, et al. (2008)

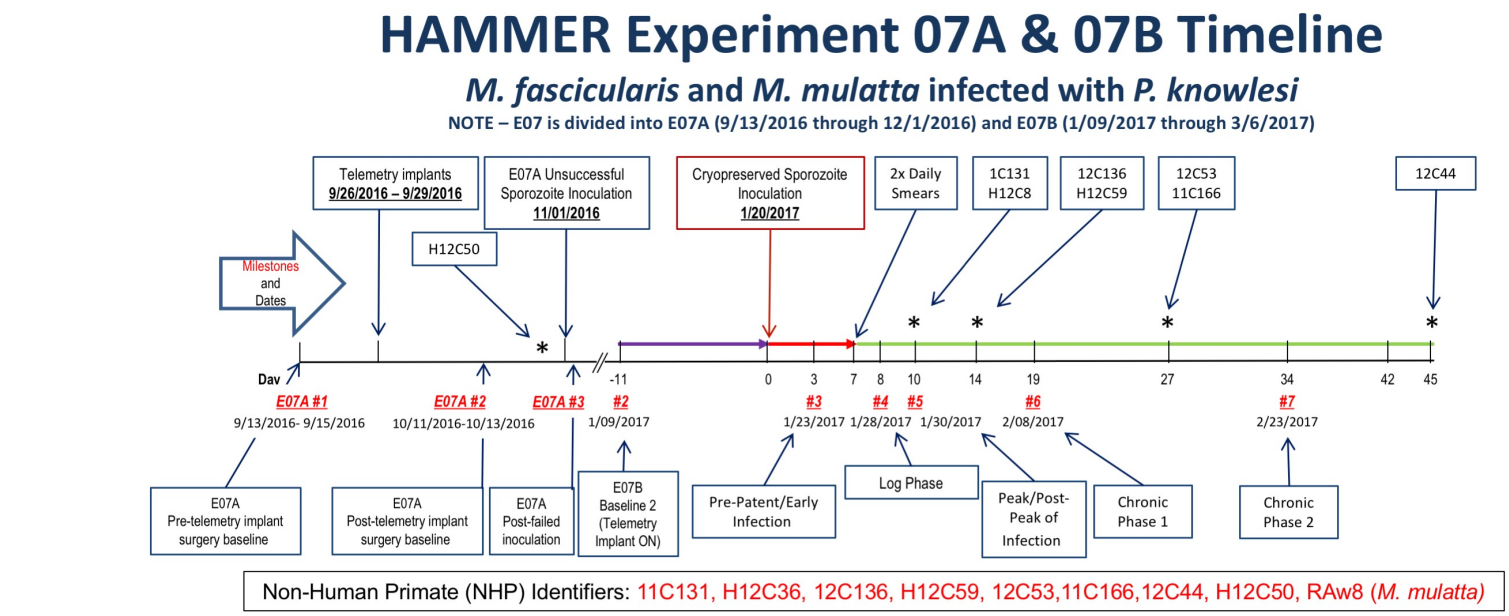


- Daily capillary blood collection for Clinical analysis (CBC, chemistry, and reticulocyte counts), and every other day capillary blood collection for Biocrates prior to sporozoite inoculation.
- Daily capillary blood collection for Clinical analysis (CBC , chemistry, reticulocyte counts, and parasitemia calculations), and every other day capillary blood collection for Biocrates after sporozoite inoculation through Day 7.
- Twice daily capillary blood collection for Clinical analysis (parasitemia calculations at 8:00 am collection and CBC , chemistry, reticulocyte counts, and parasitemia calculations at 1:00 pm collection) until Necropsy
- **#1** through **#5** time point sample collection of capillary blood, venous whole blood, and bone marrow collection for Clinical analysis (CBC , chemistry, reticulocyte counts, and parasitemia calculations), Functional Genomics, Lipidomics, Untargeted Metabolomics, Immune Profiling , Biocrates, SOMAscan, NanoSight, Pathology, and Metagenomics (air, fur, and rectal). Saliva collection for Biocrates.

* • Necropsy – Capillary blood collection for Immune Profiling. Venous whole blood collection for Clinical analysis (CBC , chemistry, reticulocyte counts, and parasitemia calculations) and Immune Profiling. Bone Marrow Aspirate collection for Clinical analysis (CBC), Functional Genomics, Immune Profiling, Biocrates, and SOMAscan. CSF collection for Proteomics, and Biocrates. Urine collection for Biocrates, Untargeted Metabolomics, and Urinalysis. Tissue collection for Pathology and Functional Genomics.

- All remaining sample volumes were cryopreserved.

Figure 3.9: E06 Experiment Set Up: Source: Aurrecoechea C, et al. (2008)



- Daily capillary blood collection for Clinical analysis (CBC, chemistry, and reticulocyte counts), and every other day capillary blood collection for Biocrates prior to sporozoite inoculation.
- Daily capillary blood collection for Clinical analysis (CBC , chemistry, reticulocyte counts, and parasitemia calculations), and every other day capillary blood collection for Biocrates after sporozoite inoculation through Day 7.
- Twice daily capillary blood collection for Clinical analysis (parasitemia calculations at 8:00 am collection and CBC , chemistry, reticulocyte counts, and parasitemia calculations at 1:00 pm collection) until Necropsy
  - ***E07A #1*** through ***#3*** time point sample collection of capillary blood, venous whole blood, and bone marrow collection for Clinical analysis (CBC, chemistry, reticulocyte counts, and parasitemia calculations), Functional Genomics, Lipidomics, Untargeted Metabolomics, Immune Profiling , Biocrates, SOMAscan, NanoSight, Pathology, and Metagenomics (air, fur, and rectal). Saliva collection for Biocrates.
  - ***E07B #2*** through ***#7*** time point sample collection of capillary blood, venous whole blood, and bone marrow collection for Clinical analysis (CBC, chemistry, reticulocyte counts, and parasitemia calculations), Functional Genomics, Lipidomics, Untargeted Metabolomics, Immune Profiling , Biocrates, SOMAscan, NanoSight, Pathology, and Metagenomics (air, fur, and rectal). Saliva collection for Biocrates.
- * Necropsy – Capillary blood collection for Immune Profiling. Venous whole blood collection for Clinical analysis (CBC , chemistry, reticulocyte counts, and parasitemia calculations) and Immune Profiling. Bone Marrow Aspirate collection for Clinical analysis (CBC), Functional Genomics, Immune Profiling, Biocrates, and SOMAscan. CSF collection for Biocrates. Urine collection Biocrates, Untargeted Metabolomics, and Urinalysis. Tissue collection for Pathology and Functional Genomics.
- All remaining sample volumes were cryopreserved.

Figure 3.11: E07 Experiment Set Up: Source: Aurrecoechea C, et al. (2008)

Chapter 4

Discussion

4.1 Regularization

Many regularization terms have as their only goal to make a neural network less complex. We created a regularization term that would achieve this goal but also has a separate goal of minimizing the relative error of the output vector in a layer of the system. When combining this with dropout [52], heuristically during training this will sample and train a set of neural networks that have well-conditioned weight matrices. We compared our regularization method to the l_2 regularization and also l_1 on the Physionet Data-set. Our method outperformed both methods. The confusion matrix and also the F_1 statistics are given in the appendix.

The results show that only controlling the norm of the weight matrix is not enough for regularization. In this work we made an assumption on the weight matrix that $\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})\right\| \leq \left\|W^{\langle l \rangle}\right\|$ and also $\left\|(\mathbf{\Omega}^{\langle \mathbf{l} \rangle})^{-1}\right\| \leq \left\|(\mathbf{W}^{\langle \mathbf{l} \rangle})^{-1}\right\|$. This assumption is strong, but it seems not to damage the results when applied to data in a meaningful way. The assumption of the weight matrix being square is also a strong assumption. For future research, it is of interest to see if this assumption could be relaxed. It is of the author's opinion that in this case penalizing the generalized conditioning number would yield similar results.

4.2 Telemetry Data

This work used data only from pre-infection and the early parts of a malaria infection from two different species of non-human primates. It is important to note that *Plasmodium*

knowlesi is usually fatal within a week of blood stage for *Macaca mulatta* but *Macaca fascicularis* usually only experience a mild infection [6]. No information about clinical outcomes or the severity of the infections was used in this experiment. The question of "if there is a signature of malaria in the heart" is of interest because the ECG is a nonevasive and inexpensive process that can be implemented in clinical settings.

Our results seem to suggest that there may be a signature for malaria that exists in the heart even in the early parts of infection. In the results of this experiment, we see that our model performed the best on hour 21. We also see that in general, we have more classification power at night. This phenomenon was expected because during the day the animals had human interactions. We also speculate that because the animals are moving more during the day that noises such as muscle contractions and stimulation from other animals caused the classification power to degrade.

To check the efficacy of our pipeline, we did a random permutation of the labels in hour 23 and proceeded to train on this noised data set. As expected when doing this the model did not converge giving consistent results that the pipeline is sound. The results of training on the noised data set is also in the appendix.

Monitoring heart function has become more efficient in recent years. Many smartwatches can monitor heart function. Algorithms such as that in this work open the door to possibilities of more efficient early diagnoses of the malaria parasite. In future work, it is of interest to see what happens when training on different parts on the data set to figure out which hour is best to train on for classification. It is also of interest to see of this classification could be present in other diseases that have a clinically silent stage. More clinical experiments of this nature are needed to obtain more extensive and diverse data-sets.

Chapter 5

Conclusion

In deep learning, the task of understanding how to build models with as much generalization capabilities as possible is still in its early stages. The need for regularization is prevalent when data sets are particularly small. With computational power increasing at such a rapid pace, we can build models with really high expressive capabilities. Because of this when building models for complex yet small data sets, the question of overfitting is always daunting. Dropout seems to be a regularization that has lots of promise, but as we see throughout the literature and in this work as well, many times dropout can be improved upon.

In this work, we introduced a new regularization technique that is based on the conditioning number of the weight matrices of a neural network. We also gave a mathematically rigorous explanation of why this regularization should yield positive results on noisy data sets. We showed the potential of this regularization term by applying it to a deep neural network for the classification of ECG signals. In the Physionet challenge, we obtained better results than that of state of the art.

We also were able to distinguish between pre-infection and the liver stage of a *P. knowlesi* infection in two spices of non-human primates. This gives new insight on ways to diagnose the malaria disease much sooner which aligns with the recommendations of the World Health Organization on how to effectively treat malaria. This also shows the promise of our method and the promise of end to end deep learning in computational biology. This approach requires no expertise in a specific subject because no features were extracted from the data set. This also reduces the chance of human-imposed biases because it eliminates the opportunity for one to determine what feature is essential and what feature is not.

For future work, it is of interest to test these and similar algorithms on different and larger data sets. It is also of interest to find ways to relax assumptions made on the structure of the neural network and the weight matrices. We believe that the generalized conditioning number will play a role in regularization when weight matrices are nonsquare. This generalization will make the regularization term presented in this work apply to a broader range of neural network structures.

Bibliography

[1] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. page 21.

[3] C. Aurrecoechea. PlasmoDB: a functional genomic database for malaria parasites. Nucleic Acids Res.

[4] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[5] A. Bhattarai, A. S. Ali, S. P. Kachur, A. Mrtensson, A. K. Abbas, R. Khatib, A.-w. Al-mafazy, M. Ramsan, G. Rotllant, J. F. Gerstenmaier, F. Molteni, S. Abdulla, S. M. Montgomery, A. Kaneko, and A. Bjrkman. Impact of Artemisinin-Based Combination Therapy and Insecticide-Treated Nets on Malaria Burden in Zanzibar. *PLOS Medicine*, 4(11):e309, Nov. 2007.

[6] G. Butcher, G. Mitchell, and S. Cohen. Plasmodium knowlesi infections in a small number of non-immune natural hosts (Macaca fascicularis) and in rhesus monkeys (M. mulatta). *Transactions of the Royal Society of Tropical Medicine and Hygiene*, 104(1):75–77, Jan. 2010.

[7] C. By-Nc-Sa. World malaria report 2018. page 210.

[8] Z. Cang and G.-W. Wei. TopologyNet: Topology based deep convolutional and multi-task neural networks for biomolecular property predictions. *PLOS Computational Biology*, 13(7):e1005690, July 2017.

[9] G. Clifford, C. Liu, B. Moody, L.-w. Lehman, I. Silva, Q. Li, A. Johnson, and R. Mark. AF Classification from a Short Single Lead ECG Recording: the Physionet Computing in Cardiology Challenge 2017. Sept. 2017.

[10] J. Cox-Singh, T. M. E. Davis, K.-S. Lee, S. S. G. Shamsul, A. Matusop, S. Ratnam, H. A. Rahman, D. J. Conway, and B. Singh. Plasmodium knowlesi Malaria in Humans Is Widely Distributed and Potentially Life Threatening. *Clinical Infectious Diseases*, 46(2):165–171, Jan. 2008.

[11] G. Cybenkot. Approximation by superpositions of a sigmoidal function. page 12.

[12] S. Demko. Condition numbers of rectangular systems and bounds for generalized inverses. *Linear Algebra and its Applications*, 78:199–206, June 1986.

[13] L. Deng and D. Yu. Deep Learning: Methods and Applications. May 2014.

[14] Developed with the special contribution of the European Heart Rhythm Association (EHRA), Endorsed by the European Association for Cardio-Thoracic Surgery (EACTS), Authors/Task Force Members, A. J. Camm, P. Kirchhof, G. Y. H. Lip, U. Schotten, I. Savelieva, S. Ernst, I. C. Van Gelder, N. Al-Attar, G. Hindricks, B. Prendergast, H. Heidbuchel, O. Alfieri, A. Angelini, D. Atar, P. Colonna, R. De Caterina, J. De Sutter, A. Goette, B. Gorenek, M. Heldal, S. H. Hohloser, P. Kolh, J.-Y. Le Heuzey, P. Ponikowski, F. H. Rutten, ESC Committee for Practice Guidelines (CPG), A. Vahanian, A. Auricchio, J. Bax, C. Ceconi, V. Dean, G. Filippatos, C. Funck-Brentano, R. Hobbs, P. Kearney, T. McDonagh, B. A. Popescu, Z. Reiner, U. Sechtem, P. A. Sirnes, M. Tendera, P. E. Vardas, P. Widimsky, Document Reviewers, P. E. Vardas, V. Agladze, E. Aliot, T. Balabanski, C. Blomstrom-Lundqvist, A. Capucci, H. Crijns,

B. Dahlof, T. Folliguet, M. Glikson, M. Goethals, D. C. Gulba, S. Y. Ho, R. J. M. Klautz, S. Kose, J. McMurray, P. Perrone Filardi, P. Raatikainen, M. J. Salvador, M. J. Schalij, A. Shpektor, J. Sousa, J. Stepinska, H. Uuetoa, J. L. Zamorano, and I. Zupan. Guidelines for the management of atrial fibrillation: The Task Force for the Management of Atrial Fibrillation of the European Society of Cardiology (ESC). *European Heart Journal*, 31(19):2369–2429, Oct. 2010.

[15] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr. 1980.

[16] H. Habibi Aghdam and E. Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, Cham, 2017.

[17] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, 25(1):65, Jan. 2019.

[18] K. He and J. Sun. Convolutional Neural Networks at Constrained Time Cost. *arXiv:1412.1710 [cs]*, Dec. 2014. arXiv: 1412.1710.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[20] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989.

[21] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

[22] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[23] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, D. Scherer, A. Mller, and S. Behnke. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In K. Diamantaras, W. Duch, and L. S. Iliadis, editors, *Artificial Neural Networks ICANN 2010*, volume 6354, pages 92–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[24] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, Feb. 2015. arXiv: 1502.03167.

[25] M. Janzamin, H. Sedghi, and A. Anandkumar. Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods. *arXiv:1506.08473 [cs, stat]*, June 2015. arXiv: 1506.08473.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[27] A. Krogh and J. A. Hertz. A Simple Weight Decay Can Improve Generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992.

[28] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436, May 2015.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient Based Learning Applied to Document Recognition. 1998.

[30] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, Jan. 1993.

[31] Y. Li and Y. Yuan. Convergence Analysis of Two-layer Neural Networks with ReLU Activation. page 11.

[32] G. Y. H. Lip, L. Fauchier, S. B. Freedman, I. Van Gelder, A. Natale, C. Gianni, S. Nattel, T. Potpara, M. Rienstra, H.-F. Tse, and D. A. Lane. Atrial fibrillation. *Nature Reviews Disease Primers*, 2:16016, Mar. 2016.

[33] K. S. Miller. On the Inverse of the Sum of Matrices. *Mathematics Magazine*, 54(2):67–72, 1981.

[34] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the Number of Linear Regions of Deep Neural Networks. page 9.

[35] B. Neyshabur, R. Tomioka, and N. Srebro. Norm-Based Capacity Control in Neural Networks. page 26.

[36] A. Y. Ng. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Twenty-first international conference on Machine learning - ICML '04*, page 78, Banff, Alberta, Canada, 2004. ACM Press.

[37] F. Nosten and N. J. White. Artemisinin-Based Combination Treatment of Falciparum Malaria. *The American Journal of Tropical Medicine and Hygiene*, 77(6_Suppl):181–192, Dec. 2007.

[38] D. Opitz and R. Maclin. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198, Aug. 1999.

[39] X. Pan and V. Srikumar. Expressiveness of Rectifier Networks. *arXiv:1511.05678 [cs]*, Nov. 2015. arXiv: 1511.05678.

[40] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[41] C.-C. f. D. C. a. Prevention. CDC - Malaria - About Malaria - Biology, Jan. 2019.

[42] T. V. Pyrkov, K. Slipensky, M. Barg, A. Kondrashin, B. Zhurov, A. Zenin, M. Pyatnitskiy, L. Menshikov, S. Markov, and P. O. Fedichev. Extracting biological age from biomedical data via deep learning: too much of a good thing? *Scientific Reports*, 8, Mar. 2018.

[43] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, M. P. Lungren, and A. Y. Ng. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. *arXiv:1711.05225 [cs, stat]*, Nov. 2017. arXiv: 1711.05225.

[44] K. E. Rankin, S. Graewe, V. T. Heussler, and R. R. Stanway. Imaging liver-stage malaria parasites. *Cellular Microbiology*, 12(5):569–579, 2010.

[45] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, Feb. 2010.

[46] S. Saarinen, R. Bramley, and G. Cybenko. Ill-Conditioning in Neural Network Training Problems. *SIAM Journal on Scientific Computing*, 14(3):693–714, May 1993.

[47] I. Savelieva and A. J. Camm. Clinical Relevance of Silent Atrial Fibrillation: Prevalence, Prognosis, Quality of Life, and Management. *Journal of Interventional Cardiac Electrophysiology*, 4(2):369–382, June 2000.

[48] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, Cambridge, 2014.

[49] K. Simonyan and A. Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. page 14, 2015.

[50] M. Singh, A. Sinha, and B. Krishnamurthy. Neural Networks in Adversarial Setting and Ill-Conditioned Weight Space. *arXiv:1801.00905 [cs, stat]*, Jan. 2018. arXiv: 1801.00905.

[51] J. Sjoberg and M. Viberg. Separable non-linear least-squares minimization-possible improvements for neural net fitting. In *Neural Networks for Signal Processing VII. Proceedings of the 1997 IEEE Signal Processing Society Workshop*, pages 345–354, Sept. 1997.

[52] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overtting. page 30, 2014.

[53] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway Networks. *arXiv:1505.00387 [cs]*, May 2015. arXiv: 1505.00387.

[54] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, June 2015. IEEE.

[55] N. Tangpukdee, C. Duangdee, P. Wilairatana, and S. Krudsood. Malaria Diagnosis: A Brief Review. *The Korean Journal of Parasitology*, 47(2):93–102, June 2009.

[56] A. L. Tarca, V. J. Carey, X.-w. Chen, R. Romero, and S. Drghici. Machine Learning and Its Applications to Biology. *PLOS Computational Biology*, 3(6):e116, June 2007.

[57] T. Teijeiro, C. A. Garcia, D. Castro, and P. Flix. Arrhythmia Classification from the Abductive Interpretation of Short Single-Lead ECG Records. Sept. 2017.

[58] A. M. Turing. Computing Machinery and Intelligence. *Mind, New Series*, 59(236):433–460, 1950.

[59] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3462–3471, July 2017. arXiv: 1705.02315.

[60] Wang Thomas J., Larson Martin G., Levy Daniel, Vasan Ramachandran S., Leip Eric P., Wolf Philip A., DAgostino Ralph B., Murabito Joanne M., Kannel William B., and Benjamin Emelia J. Temporal Relations of Atrial Fibrillation and Congestive Heart Failure and Their Joint Influence on Mortality. *Circulation*, 107(23):2920–2925, June 2003.

[61] N. White. Sharing malarias. *The Lancet*, 363(9414):1006, Mar. 2004.

[62] S. Wiesler and H. Ney. A Convergence Analysis of Log-Linear Training. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 657–665. Curran Associates, Inc., 2011.

[63] P. A. Wolf, R. D. Abbott, and W. B. Kannel. Atrial fibrillation as an independent risk factor for stroke: the Framingham Study. *Stroke*, 22(8):983–988, Aug. 1991.

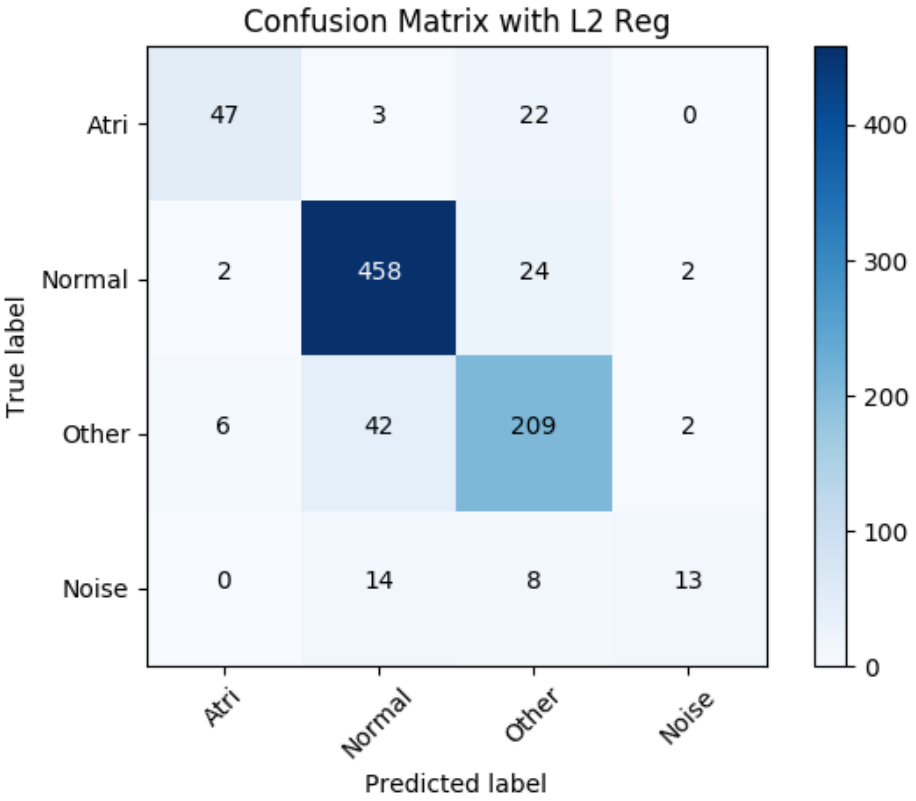
[64] R. T. R. work(s):. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[65] M. D. Zeiler and R. Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *arXiv:1301.3557 [cs, stat]*, Jan. 2013. arXiv: 1301.3557.

Appendix

5.1 Physionet Results with l_2 and l_1 Regularization

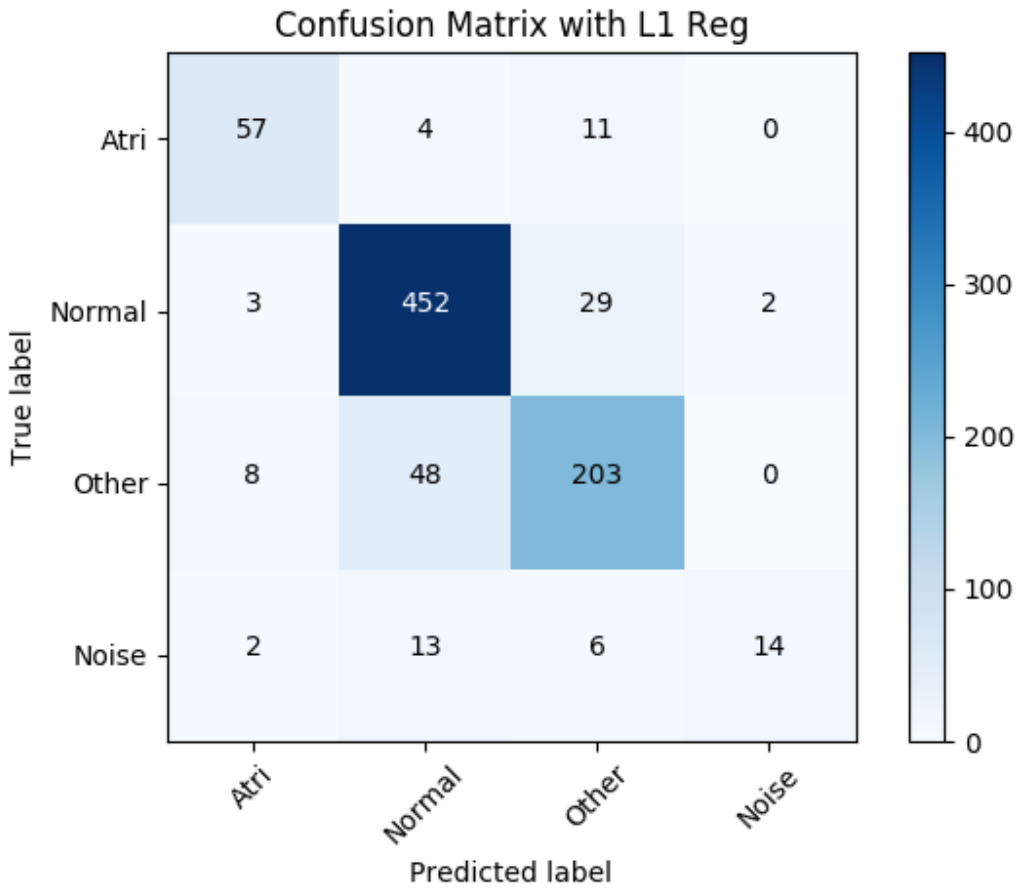
Figure 5.1: Confusion Matrix for Model With l_2 Regularization



	Precision	Recall	F_1 score	Support
Normal	.886	.942	.913	486
AF	.855	.653	.740	72
Other	.795	.807	.801	259
Noise	.765	.371	.500	35

Table 5.1: Statistics from model with l_2 Regularization

Figure 5.2: Confusion Matrix for Model With l_1 Regularization

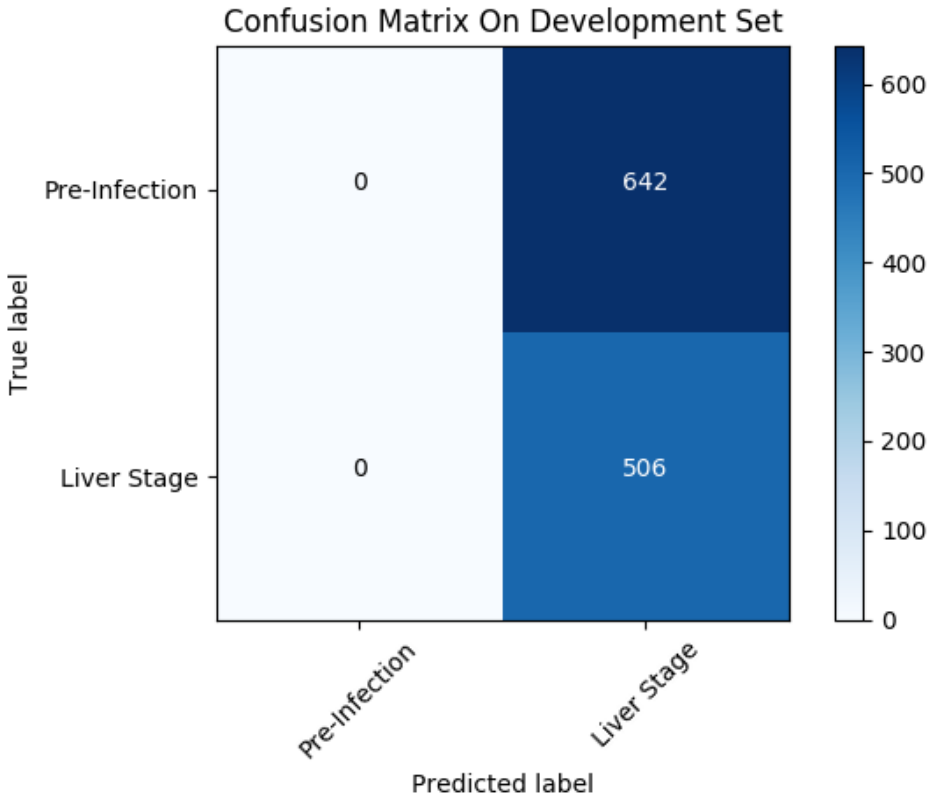


	Precision	Recall	F_1 score	Support
Normal	.874	.930	.901	486
AF	.814	.792	.803	72
Other	.815	.784	.799	259
Noise	.875	.400	.549	35

Table 5.2: Statistics from model with l_1 Regularization

5.2 Statistics on Development set when training on Data Set With Random Permutation of Labels

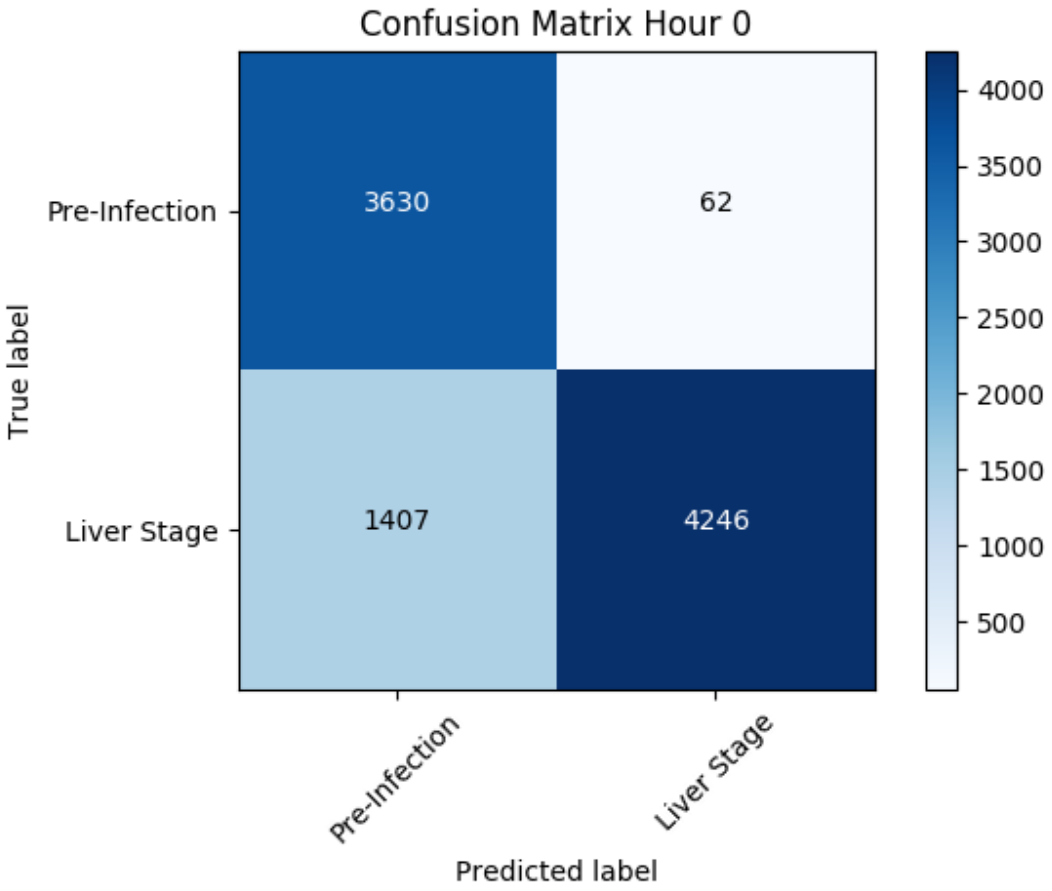
Figure 5.3: Confusion Matrix of Development set when training on Data Set With Random Permutation of Labels



	Precision	Recall	F_1 score	Support
Pre-Infection	.000	.000	.000	642
Liver Stage	.441	1.000	.612	506

Table 5.3: Statistics from Model Development Set When Trained on Data With Randomly Permuted Label

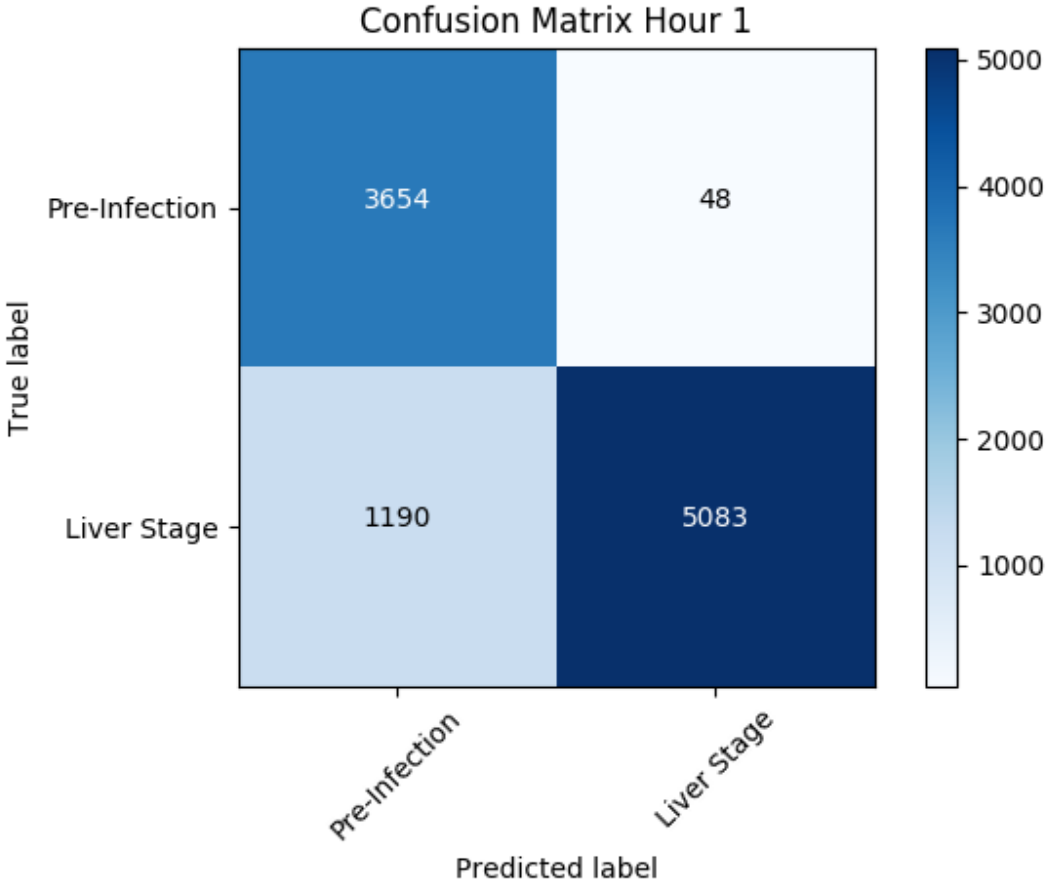
Figure 5.4: Confusion Matrix for Hour 0



	Precision	Recall	F_1 score	Support
Pre-Infection	.721	.983	.832	3692
Liver Stage	.986	.751	.853	5653

Table 5.4: Statistics from Hour 0

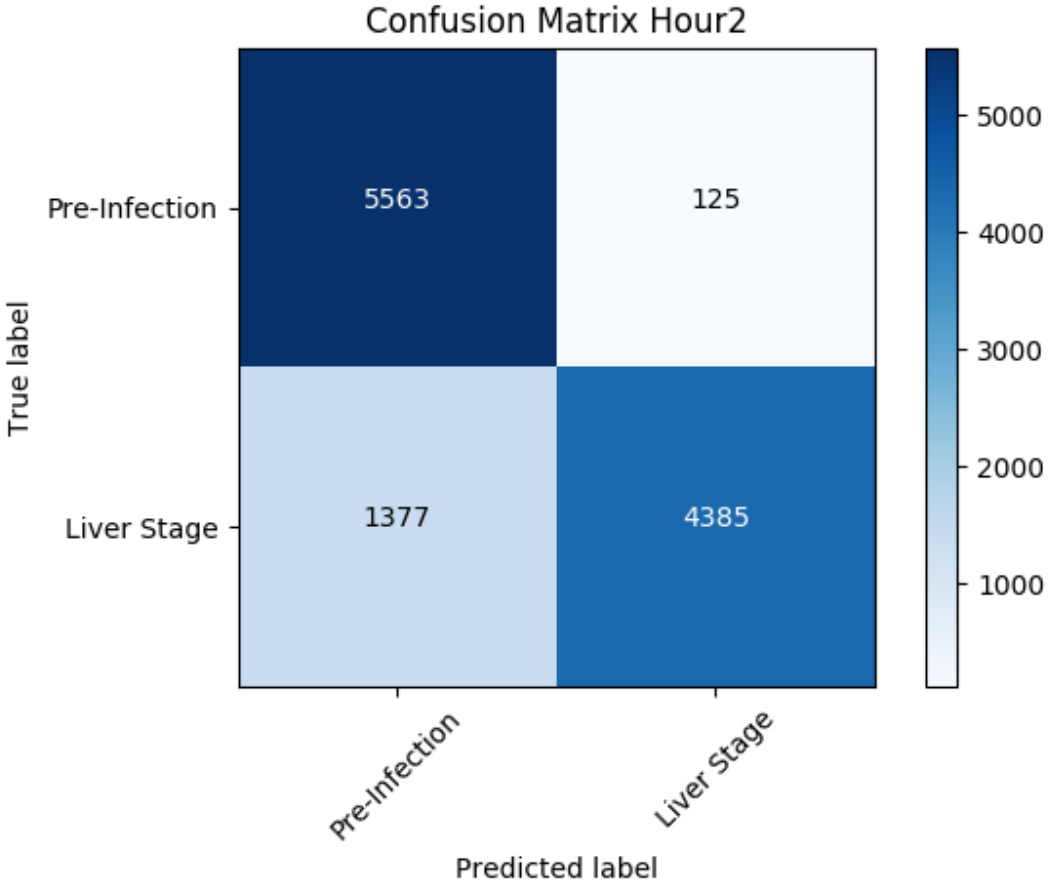
Figure 5.5: Confusion Matrix for Hour 1



	Precision	Recall	F_1 score	Support
Pre-Infection	.754	.987	.855	3702
Liver Stage	.991	.810	.891	6273

Table 5.5: Statistics from Hour 1

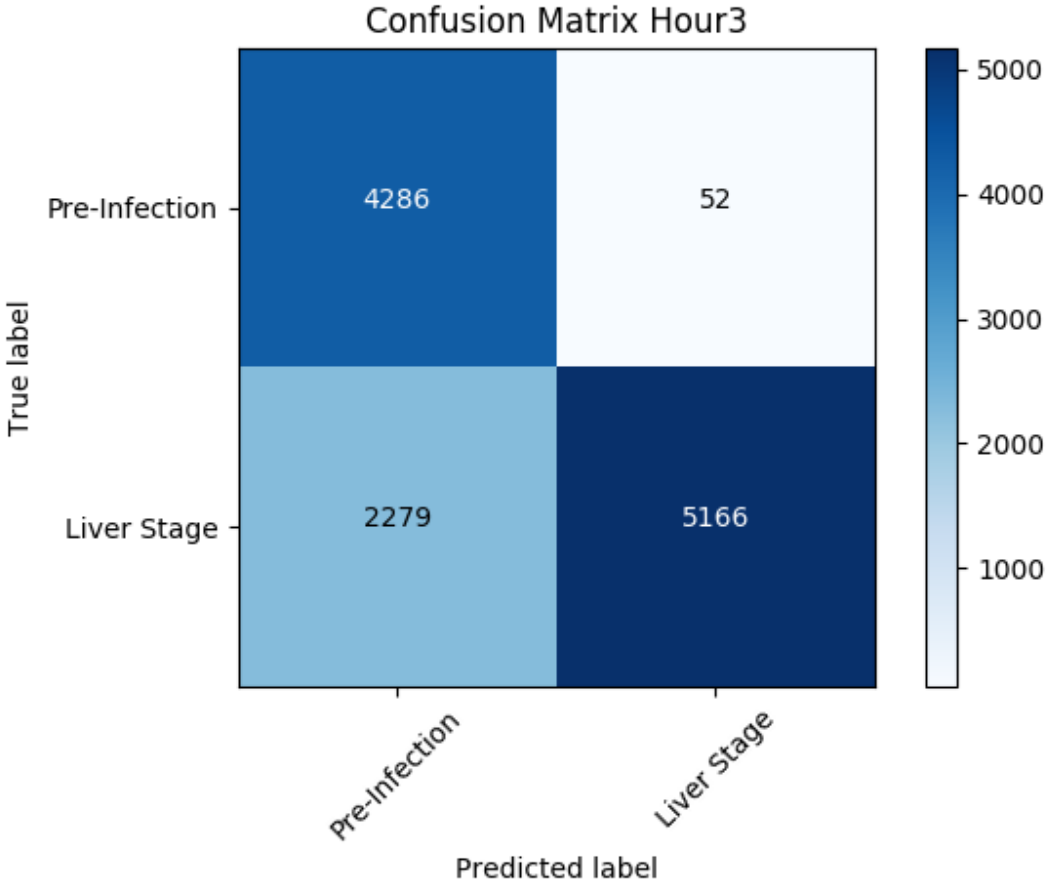
Figure 5.6: Confusion Matrix for Hour 2



	Precision	Recall	F_1 score	Support
Pre-Infection	.802	.978	.881	5688
Liver Stage	.972	.761	.854	5762

Table 5.6: Statistics from Hour 2

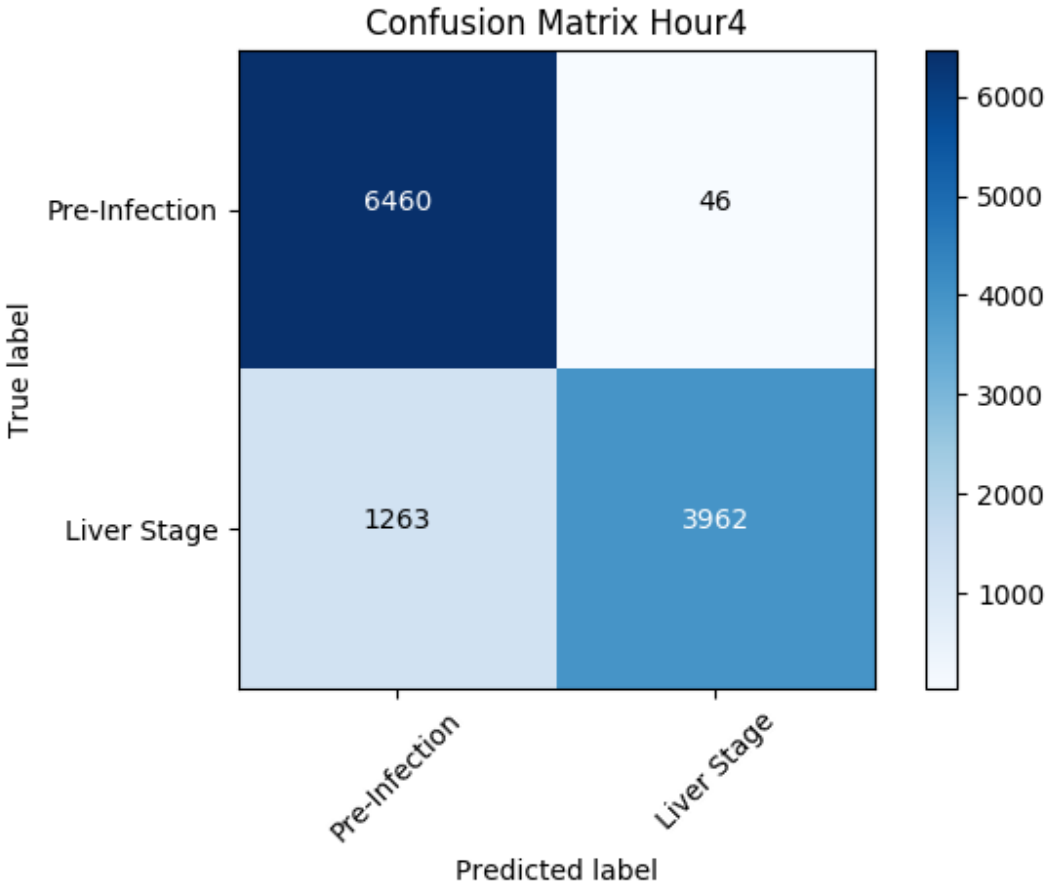
Figure 5.7: Confusion Matrix for Hour 3



	Precision	Recall	F_1 score	Support
Pre-Infection	.653	.988	.786	4338
Liver Stage	.990	.694	.816	7445

Table 5.7: Statistics from Hour 3

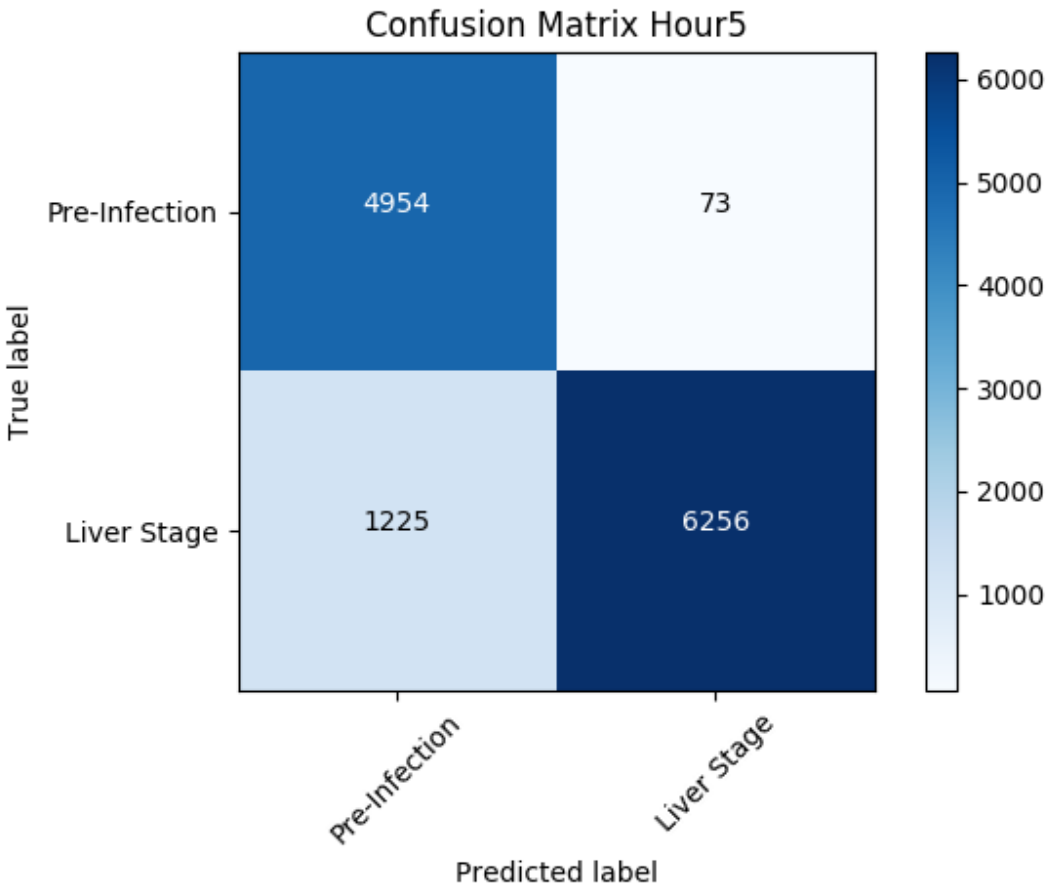
Figure 5.8: Confusion Matrix for Hour 4



	Precision	Recall	F_1 score	Support
Pre-Infection	.836	.993	.908	6506
Liver Stage	.989	.758	.858	5225

Table 5.8: Statistics from Hour 4

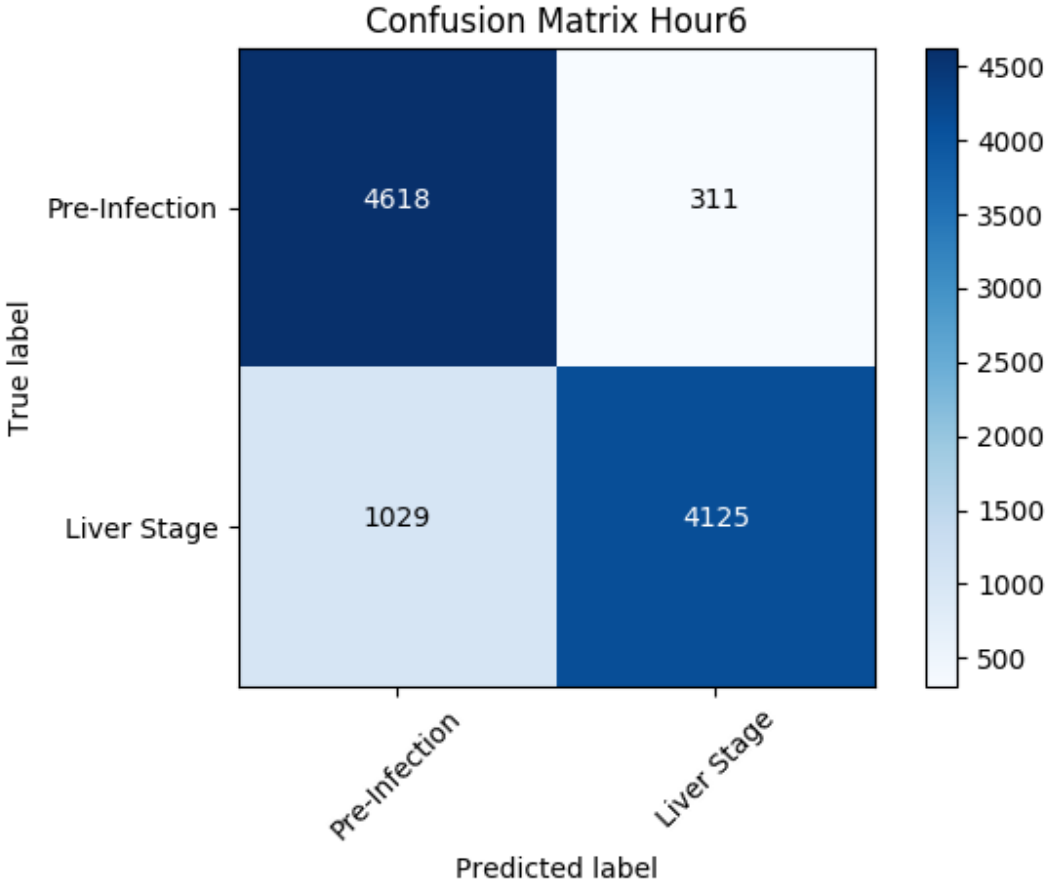
Figure 5.9: Confusion Matrix for Hour 5



	Precision	Recall	F_1 score	Support
Pre-Infection	.802	.985	.884	5027
Liver Stage	.988	.836	.906	7481

Table 5.9: Statistics from Hour 5

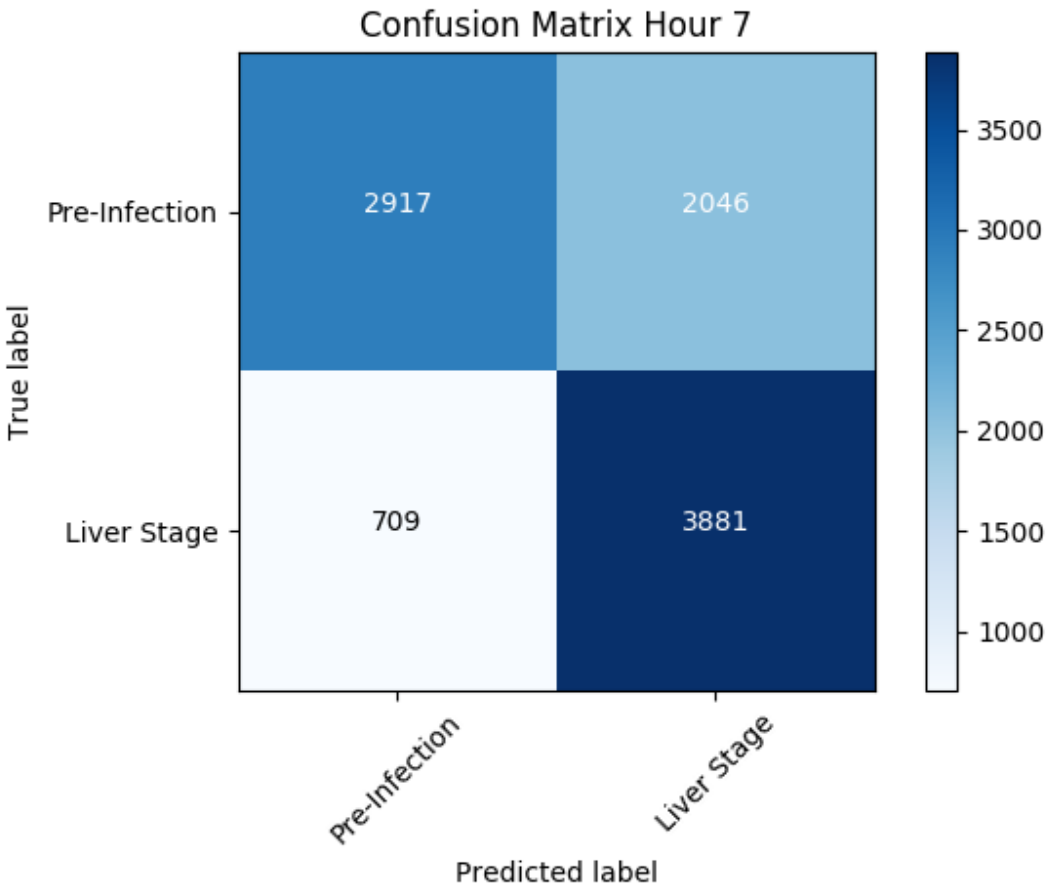
Figure 5.10: Confusion Matrix for Hour 6



	Precision	Recall	F_1 score	Support
Pre-Infection	.818	.937	.873	4929
Liver Stage	.930	.800	.860	5154

Table 5.10: Statistics from Hour 6

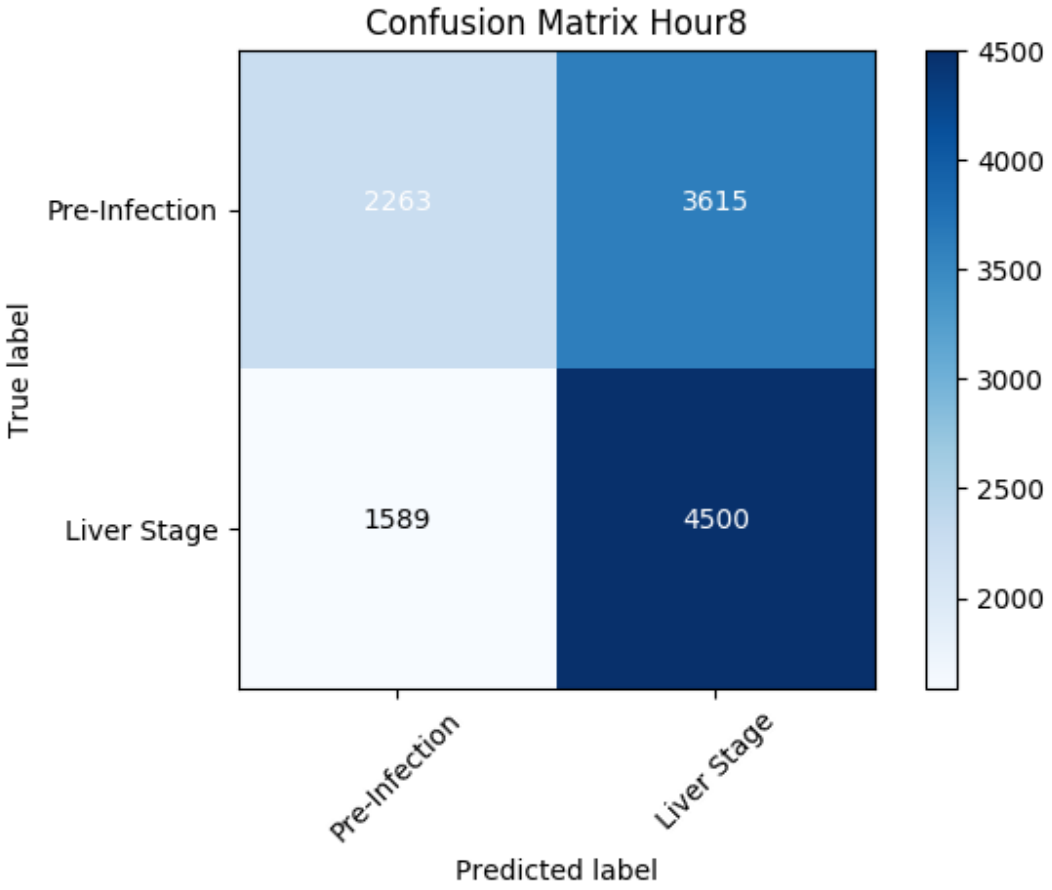
Figure 5.11: Confusion Matrix for Hour 7



	Precision	Recall	F_1 score	Support
Pre-Infection	.804	.588	.679	4963
Liver Stage	.655	.846	.738	4590

Table 5.11: Statistics from Hour 7

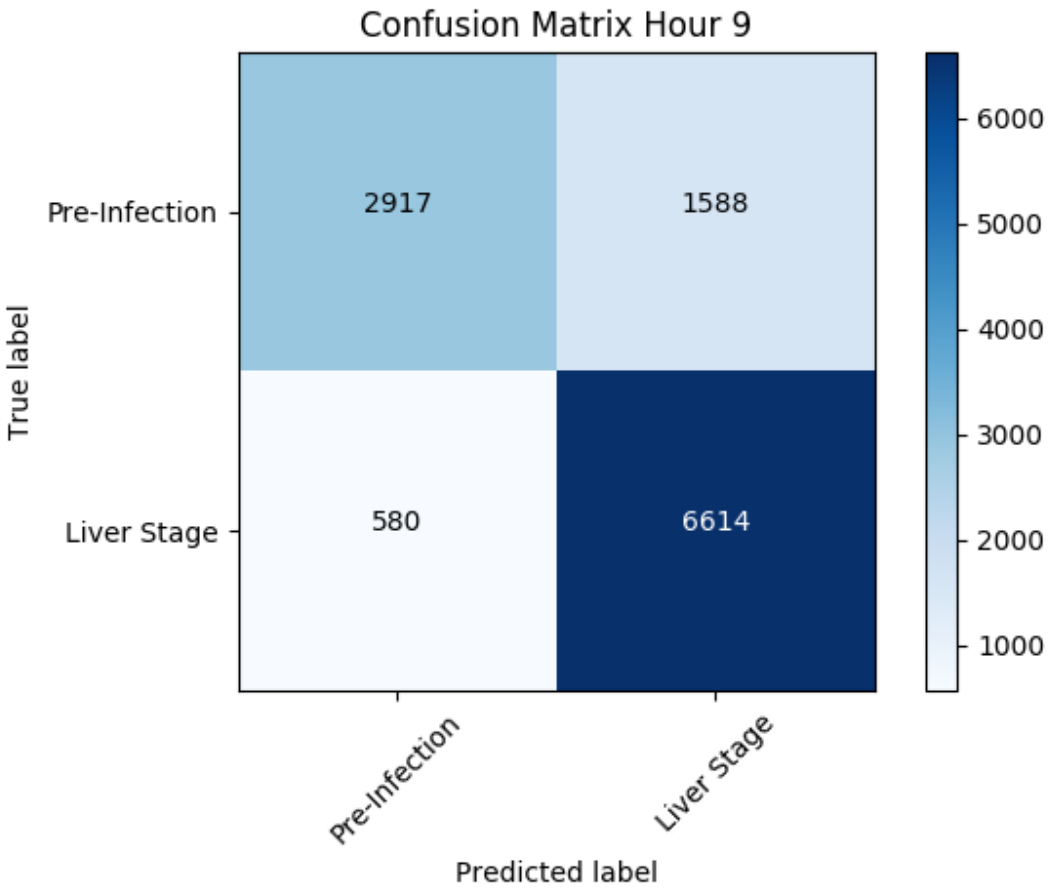
Figure 5.12: Confusion Matrix for Hour 8



	Precision	Recall	F_1 score	Support
Pre-Infection	.587	.385	.465	5878
Liver Stage	.555	.739	.634	6089

Table 5.12: Statistics from Hour 8

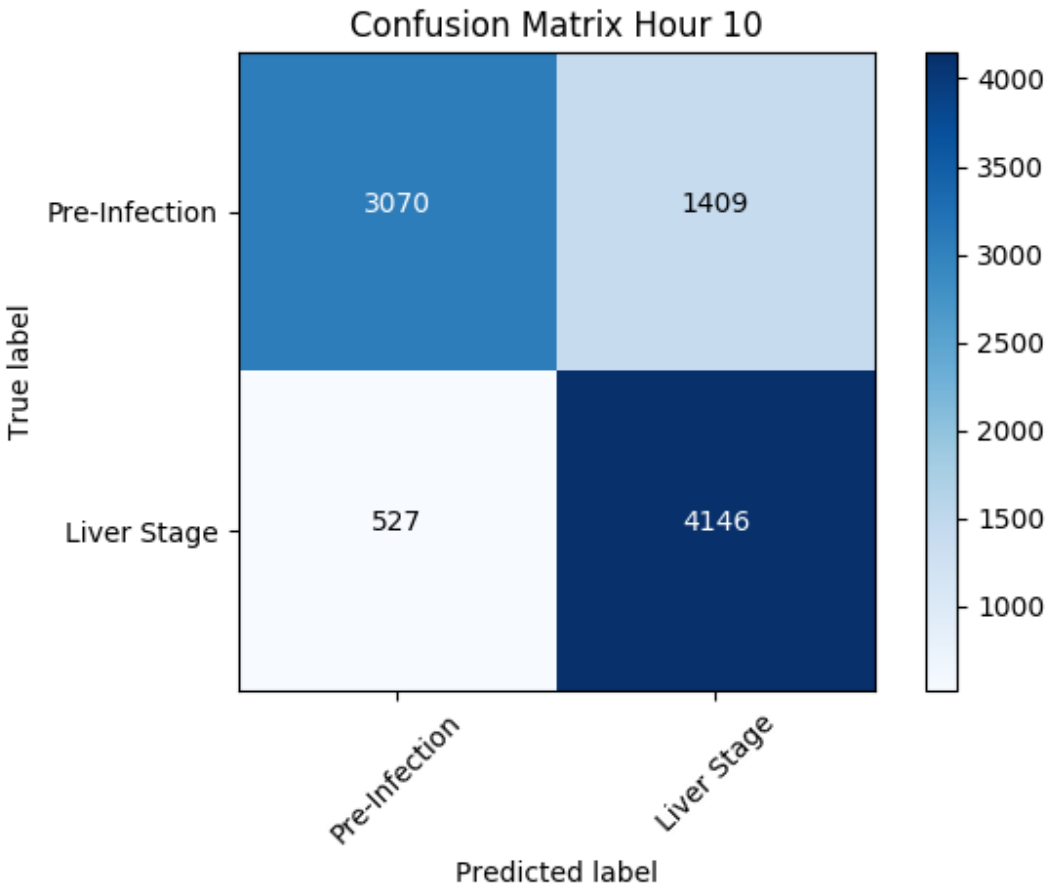
Figure 5.13: Confusion Matrix for Hour 9



	Precision	Recall	F_1 score	Support
Pre-Infection	.834	.648	.729	4505
Liver Stage	.806	.919	.859	7194

Table 5.13: Statistics from Hour 9

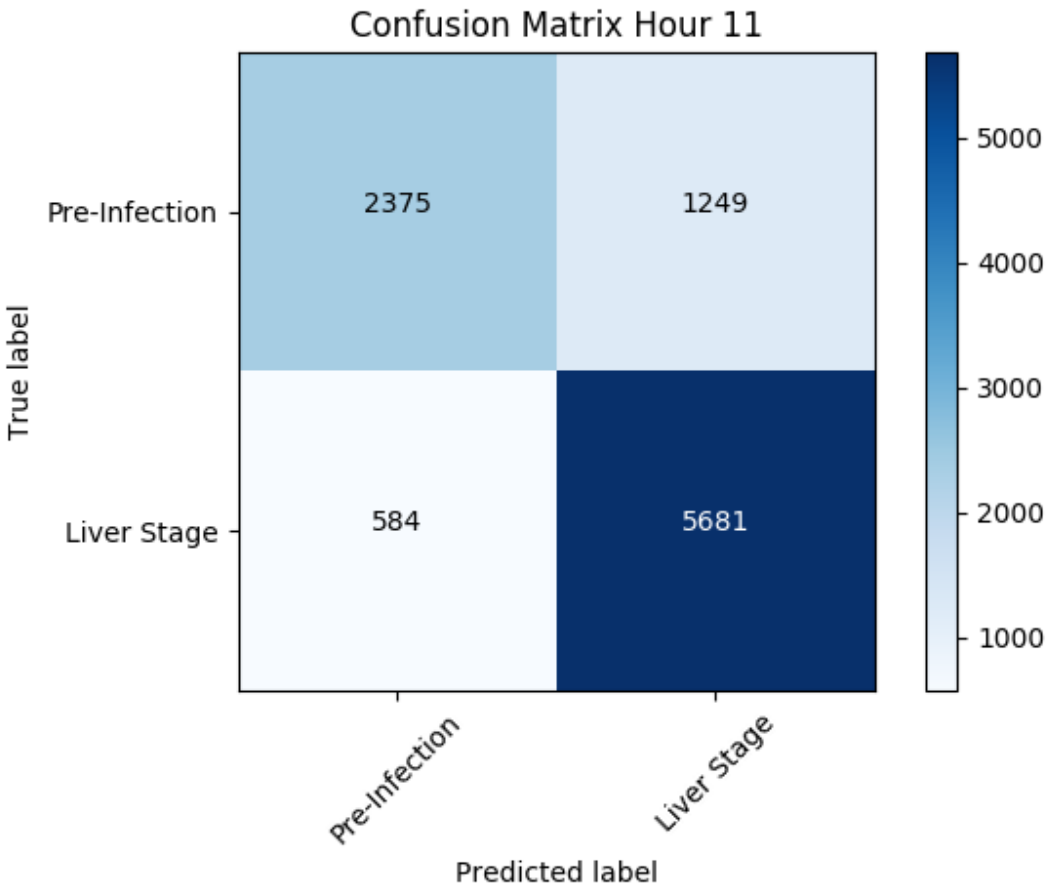
Figure 5.14: Confusion Matrix for Hour 10



	Precision	Recall	F_1 score	Support
Pre-Infection	.853	.685	.760	4479
Liver Stage	.746	.887	.811	4673

Table 5.14: Statistics from Hour 10

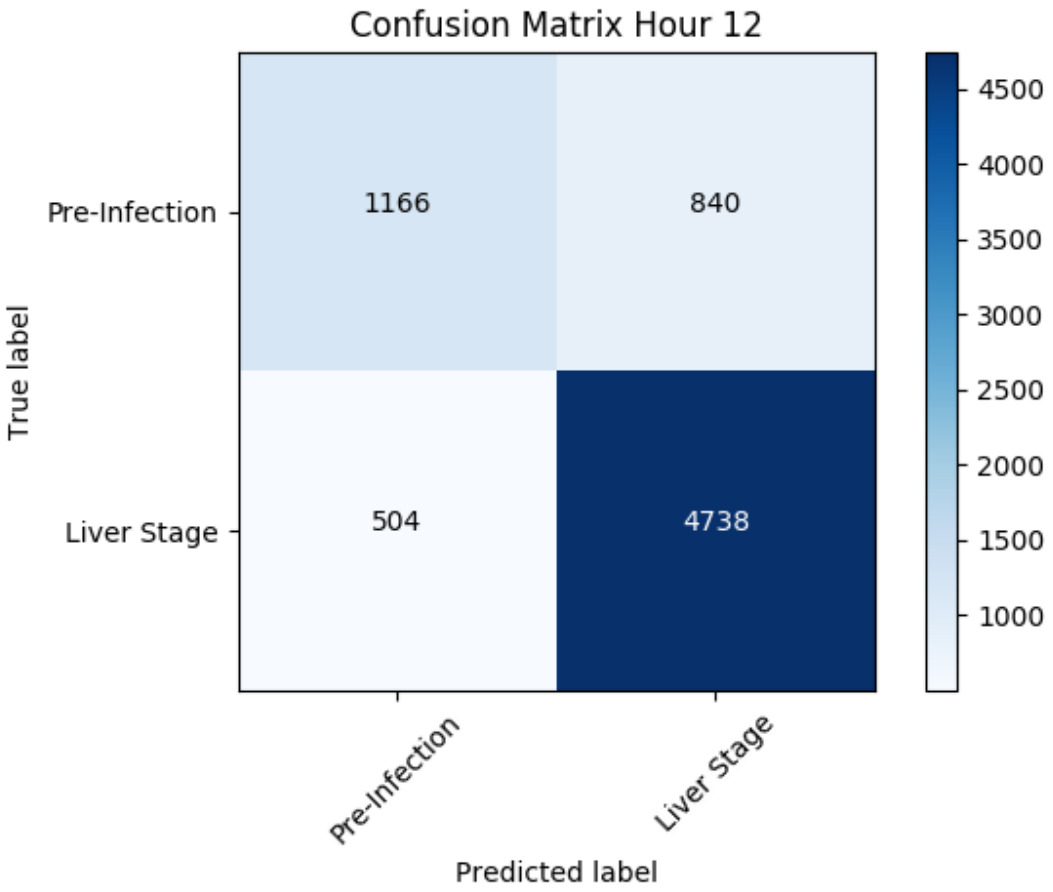
Figure 5.15: Confusion Matrix for Hour 11



	Precision	Recall	F_1 score	Support
Pre-Infection	.803	.655	.722	3624
Liver Stage	.820	.907	.861	6265

Table 5.15: Statistics from Hour 11

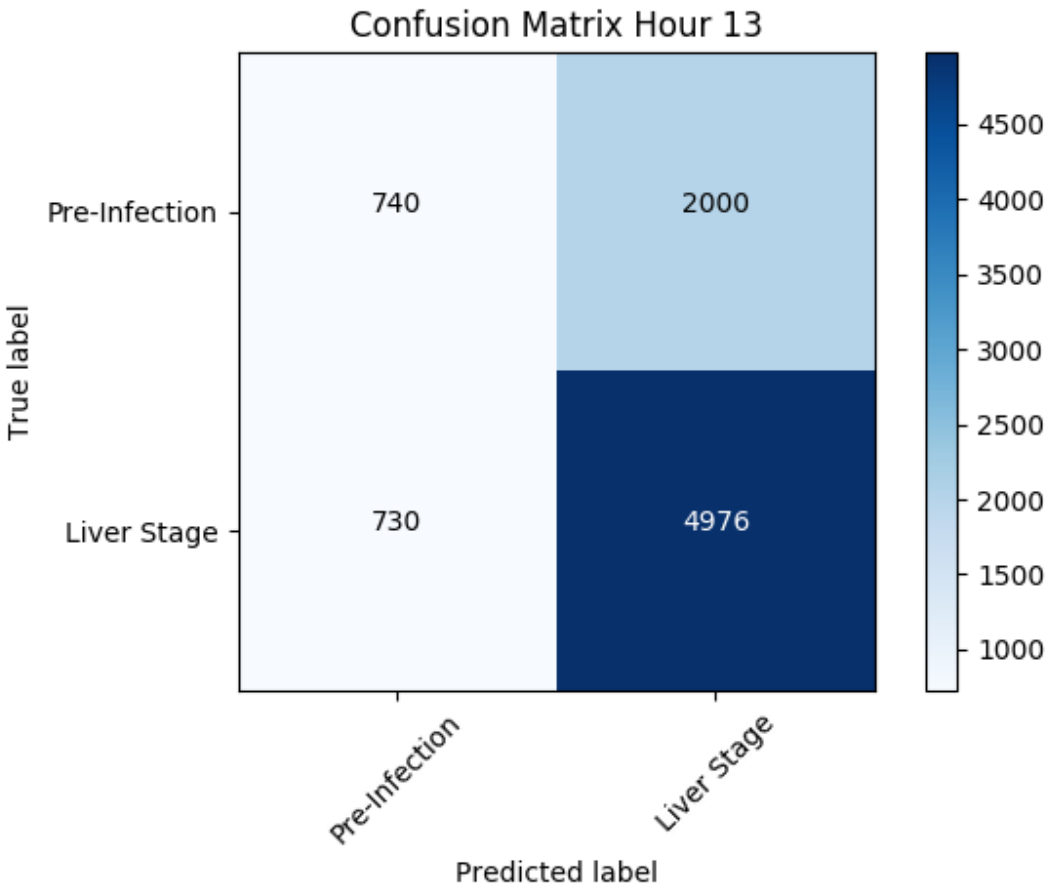
Figure 5.16: Confusion Matrix for Hour 12



	Precision	Recall	F_1 score	Support
Pre-Infection	.698	.581	.634	2006
Liver Stage	.849	.904	.876	5242

Table 5.16: Statistics from Hour 12

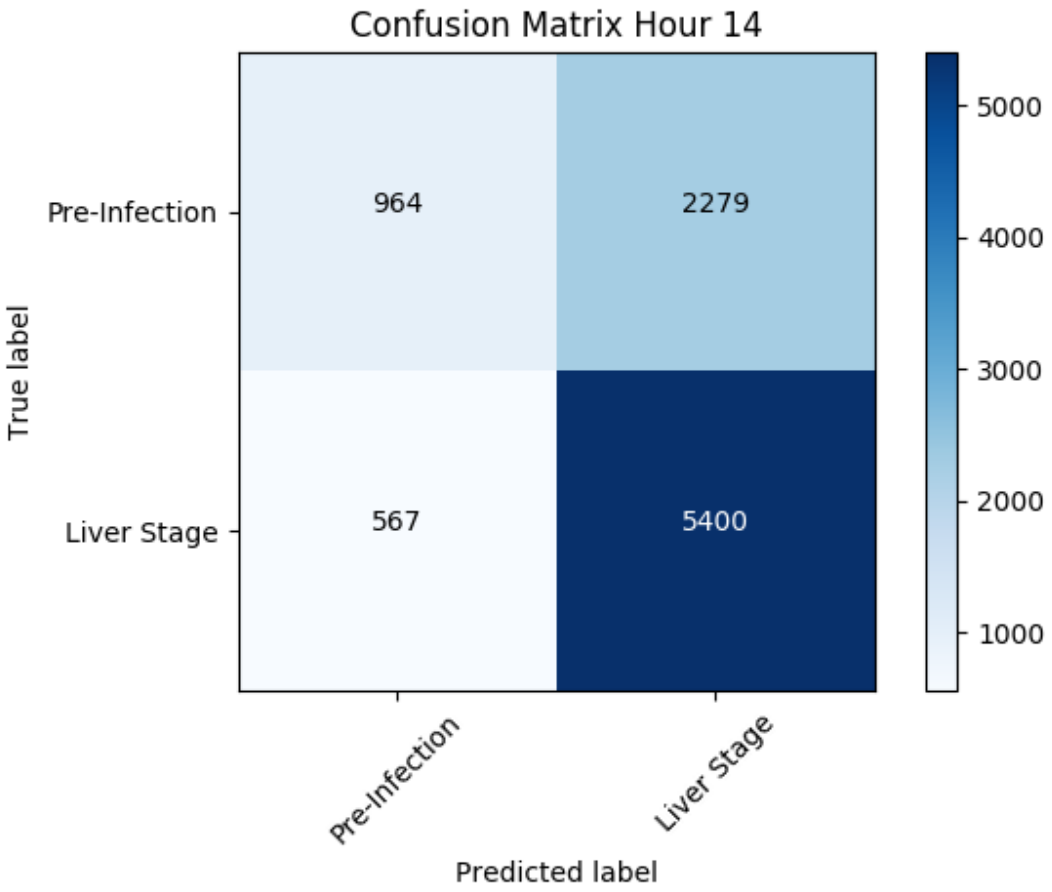
Figure 5.17: Confusion Matrix for Hour 13



	Precision	Recall	F_1 score	Support
Pre-Infection	.503	.270	.352	2740
Liver Stage	.713	.872	.785	5706

Table 5.17: Statistics from Hour 13

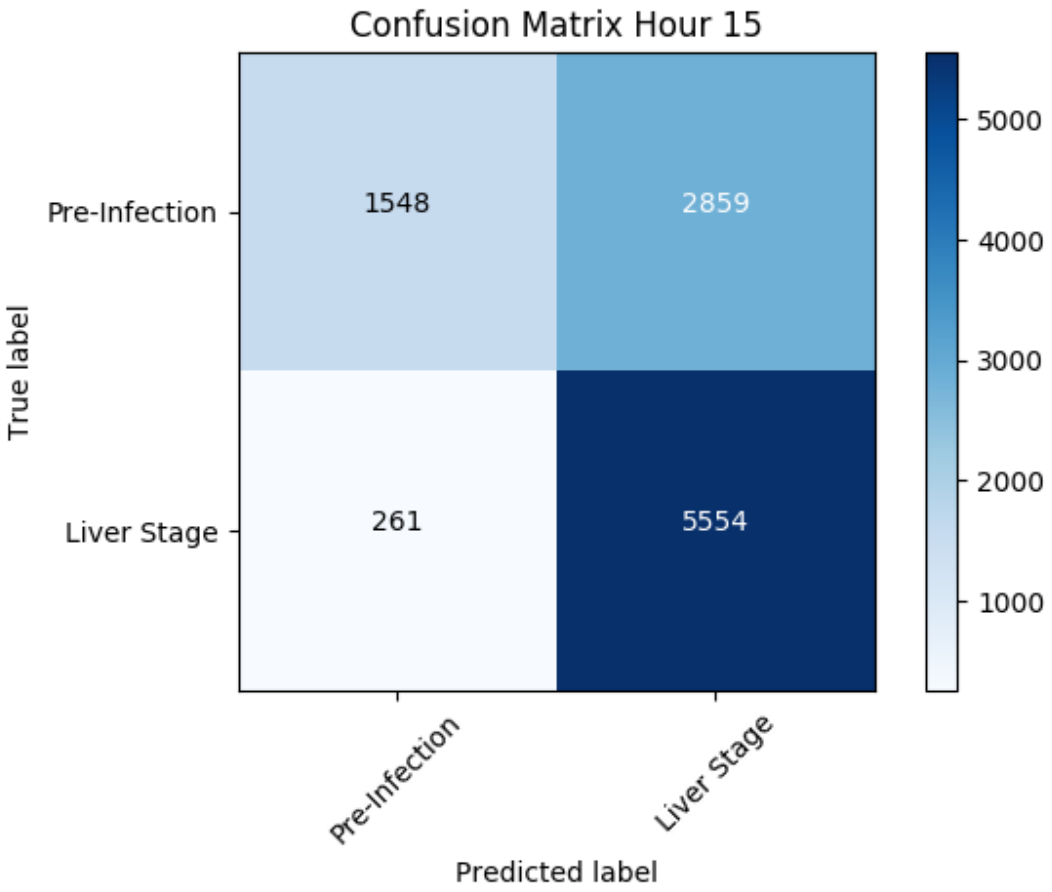
Figure 5.18: Confusion Matrix for Hour 14



	Precision	Recall	F_1 score	Support
Pre-Infection	.630	.297	.404	3243
Liver Stage	.703	.905	.791	5967

Table 5.18: Statistics from Hour 14

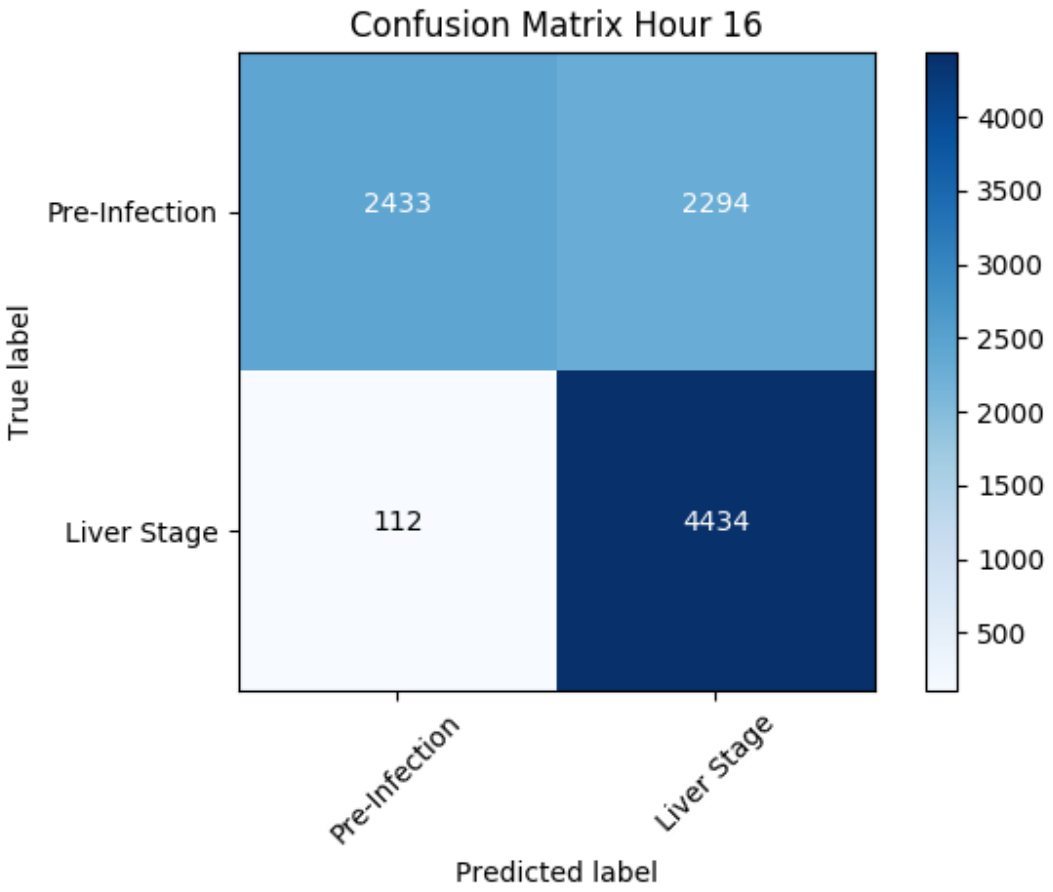
Figure 5.19: Confusion Matrix for Hour 15



	Precision	Recall	F_1 score	Support
Pre-Infection	.856	.351	.498	4407
Liver Stage	.660	.955	.781	5815

Table 5.19: Statistics from Hour 15

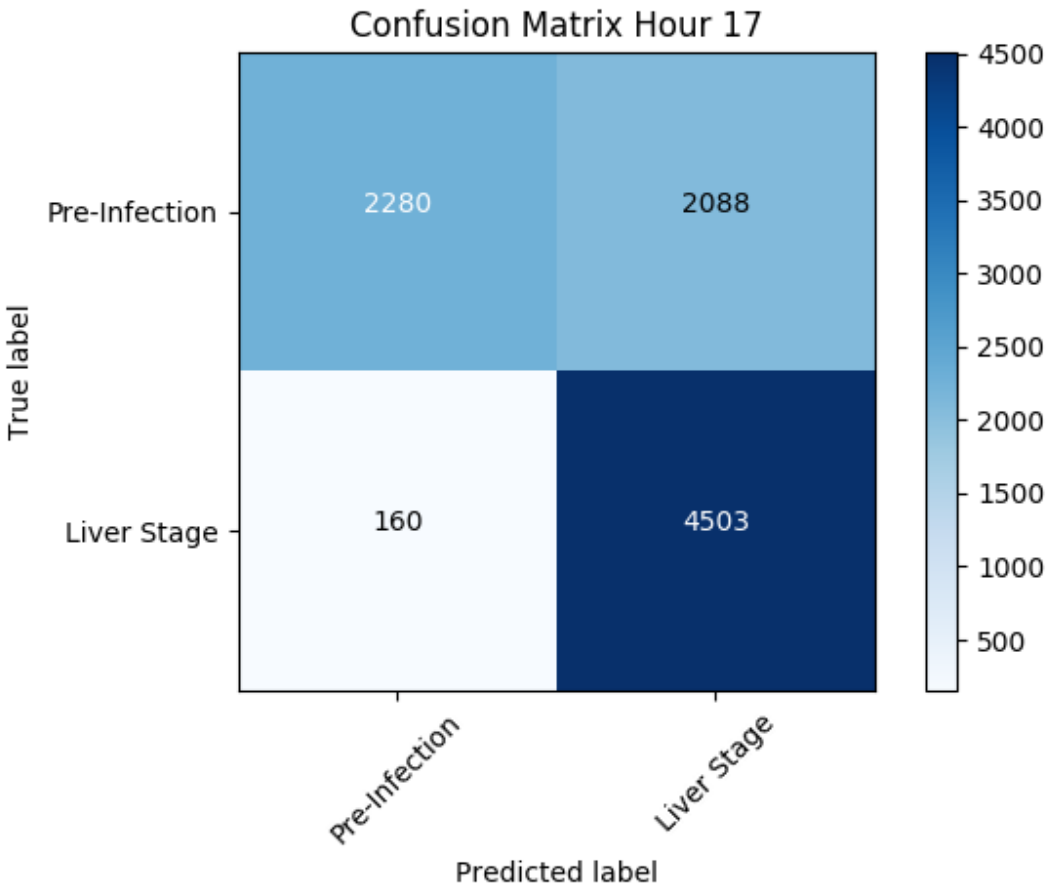
Figure 5.20: Confusion Matrix for Hour 16



	Precision	Recall	F_1 score	Support
Pre-Infection	.956	.515	.669	4727
Liver Stage	.659	.975	.787	4546

Table 5.20: Statistics from Hour 16

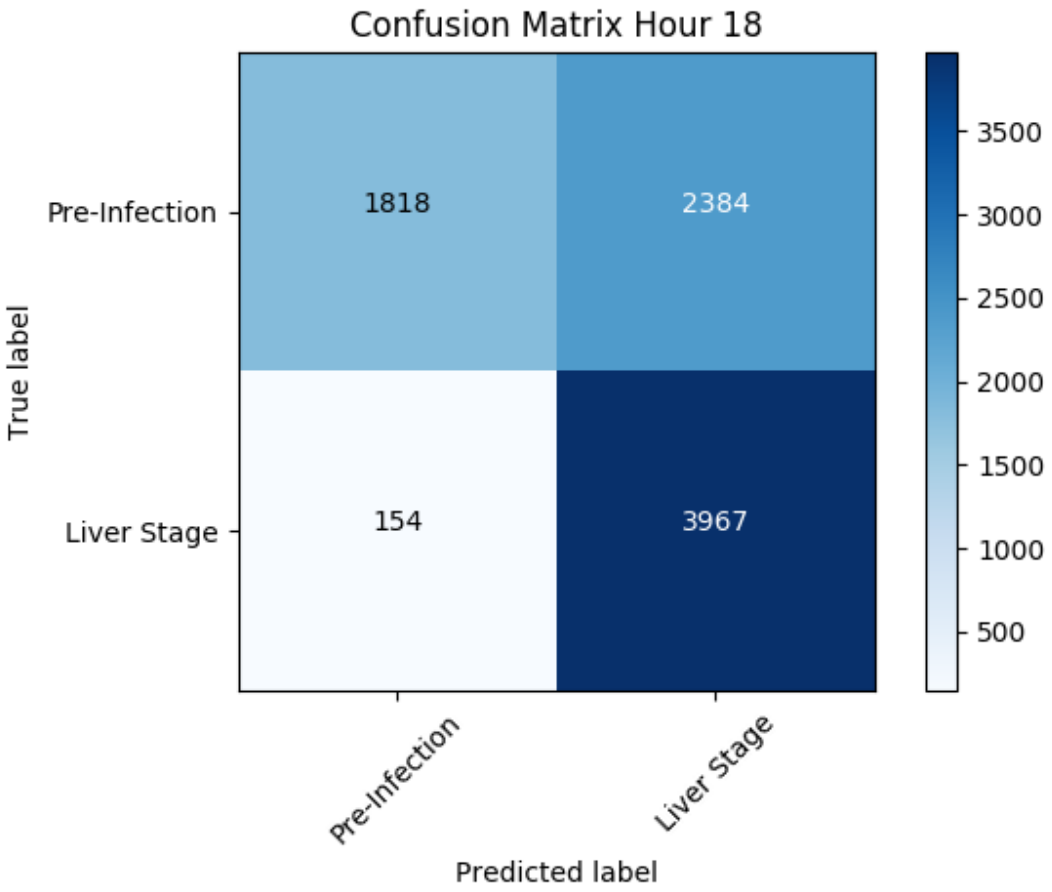
Figure 5.21: Confusion Matrix for Hour 17



	Precision	Recall	F_1 score	Support
Pre-Infection	.934	.522	.670	4368
Liver Stage	.683	.966	.800	4663

Table 5.21: Statistics from Hour 17

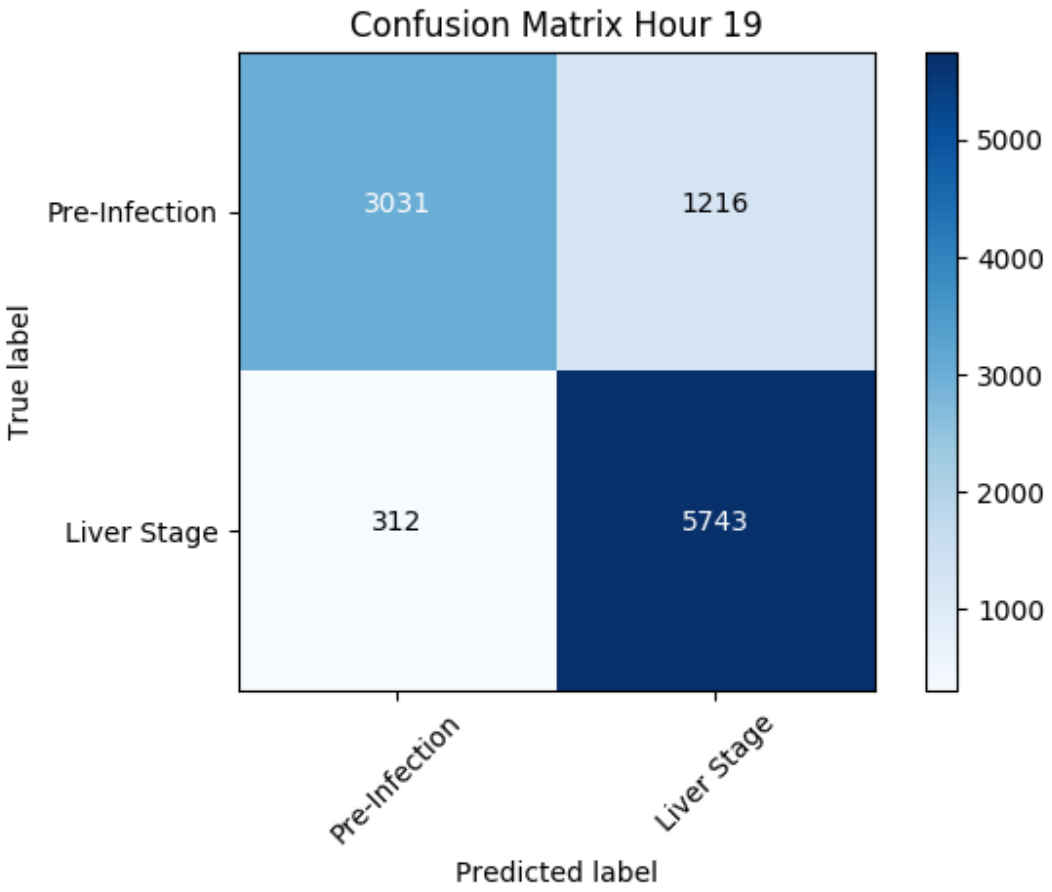
Figure 5.22: Confusion Matrix for Hour 18



	Precision	Recall	F_1 score	Support
Pre-Infection	.922	.433	.589	4202
Liver Stage	.625	.963	.758	4121

Table 5.22: Statistics from Hour 18

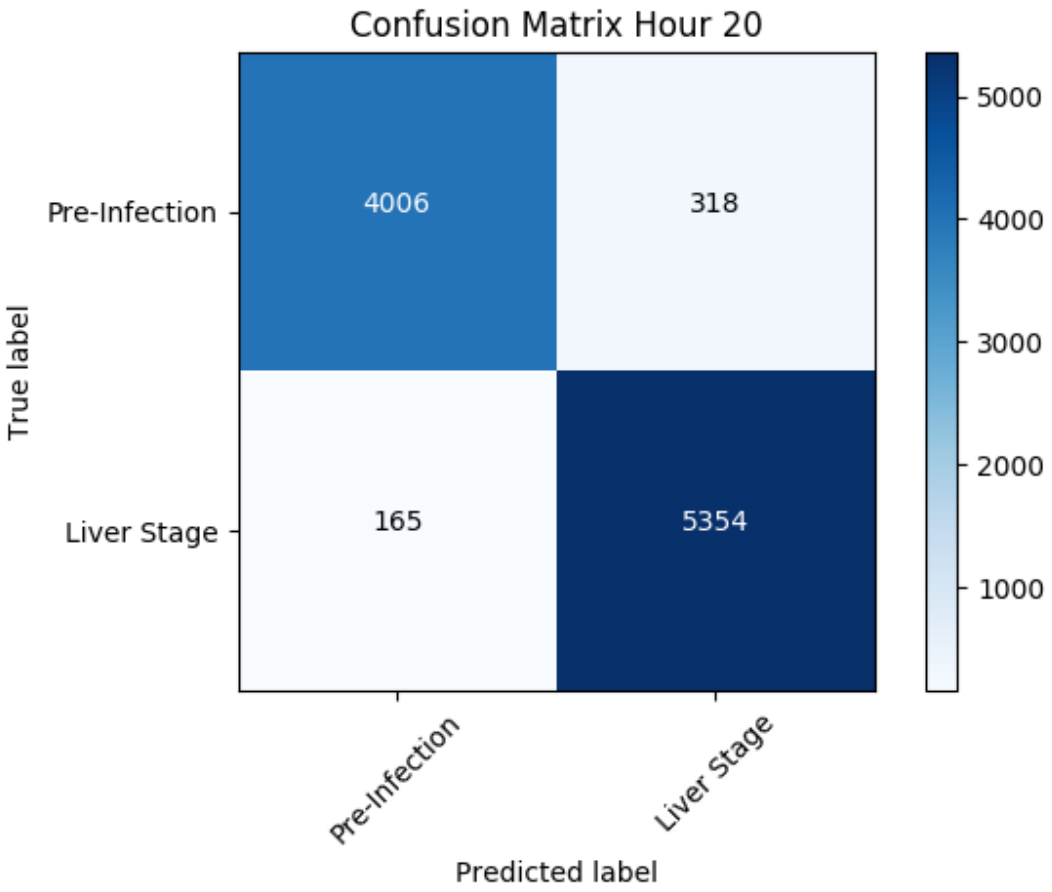
Figure 5.23: Confusion Matrix for Hour 19



	Precision	Recall	F_1 score	Support
Pre-Infection	.907	.714	.799	4247
Liver Stage	.825	.948	.883	6055

Table 5.23: Statistics from Hour 19

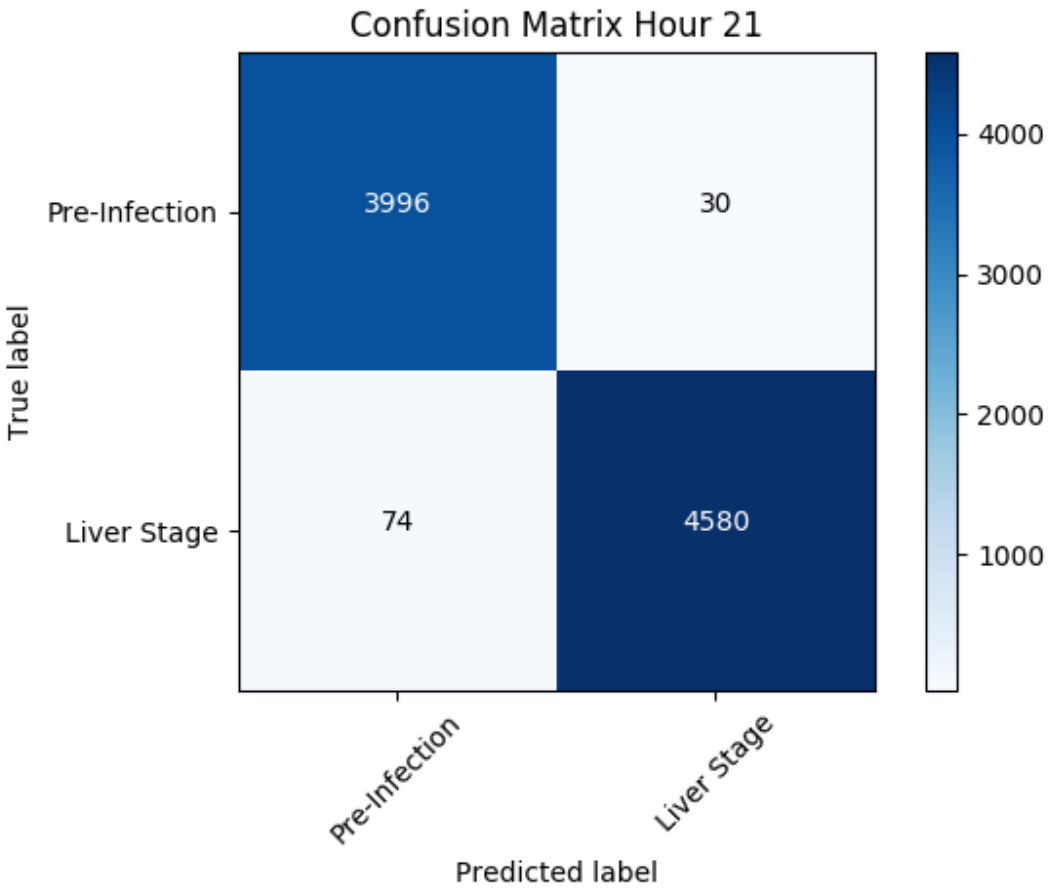
Figure 5.24: Confusion Matrix for Hour 20



	Precision	Recall	F_1 score	Support
Pre-Infection	.960	.926	.943	4324
Liver Stage	.944	.970	.957	5519

Table 5.24: Statistics from Hour 20

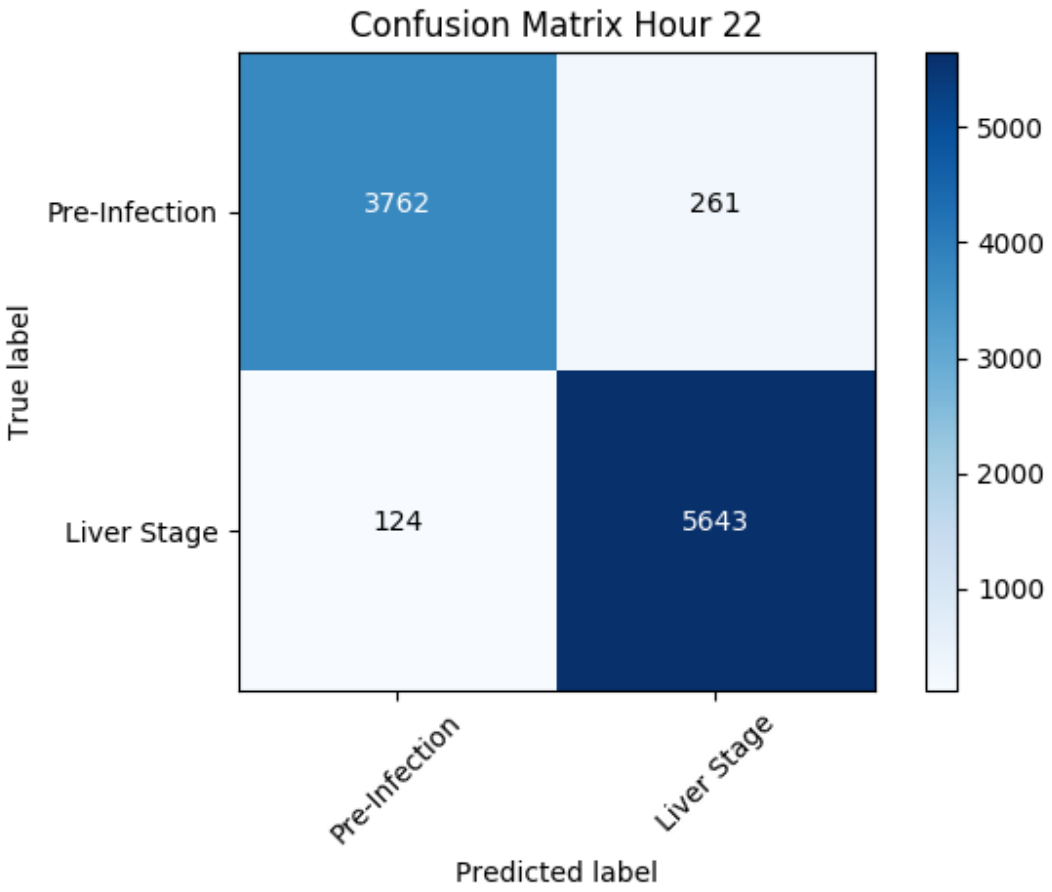
Figure 5.25: Confusion Matrix for Hour 21



	Precision	Recall	F_1 score	Support
Pre-Infection	.982	.993	.987	4026
Liver Stage	.993	.984	.989	4654

Table 5.25: Statistics from Hour 21

Figure 5.26: Confusion Matrix for Hour 22



	Precision	Recall	F_1 score	Support
Pre-Infection	.968	.935	.951	4023
Liver Stage	.956	.978	.967	5767

Table 5.26: Statistics from Hour 22

5.4 Code to Train and Evaluate Model

Create and Train Model

```
import tensorflow as tf
from keras import backend as K
from keras.layers import BatchNormalization
from keras.layers import Add


#New Regularization Term
def l2_condsq(weight_matrix):
    A_1=tf.nn.l2_loss(weight_matrix)
    A_2=tf.linalg.inv(weight_matrix)
    A_5=.1*tf.multiply(tf.nn.l2_loss(A_2),A_1)


    return A_5

#Helper Functions to Build Model
def _bn_relu(layer, dropout=0, **params):
    from keras.layers import BatchNormalization
    from keras.layers import Activation
    layer = BatchNormalization()(layer)
    layer = Activation(params["conv_activation"])(layer)

    if dropout > 0:
        from keras.layers import Dropout
        layer = Dropout(params["conv_dropout"])(layer)

```

```
27      return layer
28
29  def add_conv_weight(
30          layer,
31          filter_length,
32          num_filters,
33          subsample_length=1,
34          **params):
35      from keras.layers import Conv1D
36      layer = Conv1D(
37          filters=num_filters,
38          kernel_size=filter_length,
39          strides=subsample_length,
40          padding='same',
41          kernel_initializer=params["conv_init"])(layer)
42      return layer
43
44
45  def add_conv_layers(layer, **params):
46      for subsample_length in params["conv_subsample_lengths"]:
47          layer = add_conv_weight(
48                      layer,
49                      params["conv_filter_length"],
50                      params["conv_num_filters_start"],
51                      subsample_length=subsample_length,
52                      **params)
53          layer = _bn_relu(layer, **params)
54      return layer
55
```

```
def resnet_block(
        layer,
        num_filters,
        subsample_length,
        block_index,
        **params):
    from keras.layers import Add
    from keras.layers import MaxPooling1D
    from keras.layers.core import Lambda

    def zeropad(x):
        y = K.zeros_like(x)
        return K.concatenate([x, y], axis=2)

    def zeropad_output_shape(input_shape):
        shape = list(input_shape)
        assert len(shape) == 3
        shape[2] *= 2
        return tuple(shape)

    shortcut = MaxPooling1D(pool_size=subsample_length)(layer)
    zero_pad = (block_index % params["conv_increase_channels_at"]) == 0
    \
        and block_index > 0
    if zero_pad is True:
        shortcut = Lambda(zeropad, output_shape=zeropad_output_shape)(
   shortcut)

    for i in range(params["conv_num_skip"]):
```

```
83          if not (block_index == 0 and i == 0):
84              layer = _bn_relu(
85                  layer,
86                  dropout=params["conv_dropout"] if i > 0 else 0,
87                  **params)
88          layer = add_conv_weight(
89              layer,
90              params["conv_filter_length"],
91              num_filters,
92              subsample_length if i == 0 else 1,
93              **params)
94      layer = Add()([shortcut, layer])
95      return layer
96
97  def get_num_filters_at_index(index, num_start_filters, **params):
98      return 2**int(index / params["conv_increase_channels_at"]) \
99          * num_start_filters
100
101 def add_resnet_layers(layer, **params):
102     layer = add_conv_weight(
103         layer,
104         params["conv_filter_length"],
105         params["conv_num_filters_start"],
106         subsample_length=1,
107         **params)
108     layer = _bn_relu(layer, **params)
109     for index, subsample_length in enumerate(params["
    conv_subsample_lengths"]):
110         num_filters = get_num_filters_at_index(
```

```
111             index, params["conv_num_filters_start"], **params)
112         layer = resnet_block(
113             layer,
114             num_filters,
115             subsample_length,
116             index,
117             **params)
118     layer = _bn_relu(layer, **params)
119     return layer
120
121
122
123 def add_output_layer(layer, **params):
124     from keras.layers.core import Dense, Activation
125     from keras.layers.wrappers import TimeDistributed
126     layer = TimeDistributed(Dense(params["num_categories"]))(layer)
127     layer = BatchNormalization()(layer)
128     shortcut=layer
129     layer = Activation(params["conv_activation"])(layer)
130     layer = TimeDistributed(Dense(params["num_categories"],
    kernel_regularizer=l2_condsq))(layer)
131     layer = Add()([shortcut, layer])
132     return Activation('softmax')(layer)
133
134 def add_compile(model, **params):
135     from keras.optimizers import Adam
136     optimizer = Adam(
137         lr=params["learning_rate"],
138         clipnorm=params.get("clipnorm", 1))
```

```
139
140     model.compile(loss='categorical_crossentropy',
141                   optimizer=optimizer,
142                   metrics=['accuracy'])
143
144 def build_network(**params):
145     from keras.models import Model
146     from keras.layers import Input
147     inputs = Input(shape=params['input_shape'],
148                    dtype='float32',
149                    name='inputs')
150
151     if params.get('is_regular_conv', False):
152         layer = add_conv_layers(inputs, **params)
153     else:
154         layer = add_resnet_layers(inputs, **params)
155
156     output = add_output_layer(layer, **params)
157     model = Model(inputs=[inputs], outputs=[output])
158     if params.get("compile", True):
159         add_compile(model, **params)
160     return model
161
162 import json
163 import keras
164 import numpy as np
165 import os
166 import random
167 import scipy.io as sio
```

```
168 def load_ecg_mat(ecg_file):
169     return sio.loadmat(ecg_file)['val'].squeeze()
170
171
172 STEP=512
173 labels=[]
174 ecgs=[]
175 def load_all(data_path):
176     label_file = data_path+ "/REFERENCE-v3.csv"
177     with open(label_file, 'r') as fid:
178         records = [l.strip().split(",") for l in fid]
179
180     dataset = []
181     for record, label in records:
182         ecg_file = os.path.join(data_path, record + ".mat")
183         ecg_file = os.path.abspath(ecg_file)
184         ecg = load_ecg_mat(ecg_file)
185         num_labels = ecg.shape[0] / STEP
186         dataset.append((ecg_file, [label]*int(num_labels)))
187     return dataset
188
189 def split(dataset, dev_frac):
190     dev_cut = int(dev_frac * len(dataset))
191     random.shuffle(dataset)
192     dev = dataset[:dev_cut]
193     train = dataset[dev_cut:]
194     return train, dev
195 def make_json(save_path, dataset):
196     with open(save_path, 'w') as fid:
```

```
197         for d in dataset:
198             datum = {'ecg' : d[0],
199                      'labels' : d[1]}
200             json.dump(datum, fid)
201             fid.write('\n')
202 def load_dataset(data_json):
203     with open(data_json, 'r') as fid:
204         data = [json.loads(l) for l in fid]
205     labels = []; ecgs = []
206     for d in data:
207         labels.append(d['labels'])
208         ecgs.append(load_ecg(d['ecg']))
209     return ecgs, labels
210 
211 
212 data_path=r"/scratch/zw89669/training2017"
213 dataset=load_all(data_path)
214 random.seed(2018)
215 dev_frac=.1
216 train, dev = split(dataset, dev_frac)
217 make_json("train.json", train)
218 make_json("dev.json", dev)
219 
220 
221 
222 def data_generator(batch_size, preproc, x, y):
223     num_examples = len(x)
224     examples = zip(x, y)
225     examples = sorted(examples, key = lambda x: x[0].shape[0])
```

```
226     end = num_examples - batch_size + 1
227     batches = [examples[i:i+batch_size]
228                for i in range(0, end, batch_size)]
229     random.shuffle(batches)
230     while True:
231         for batch in batches:
232             x, y = zip(*batch)
233             yield preproc.process(x, y)
234 
235 class Preproc:
236 
237     def __init__(self, ecg, labels):
238         self.mean, self.std = compute_mean_std(ecg)
239         self.classes = sorted(set(l for label in labels for l in label)
    )
240         self.int_to_class = dict( zip(range(len(self.classes)), self.
    classes))
241         self.class_to_int = {c : i for i, c in self.int_to_class.items
    ()}
242 
243     def process(self, x, y):
244         return self.process_x(x), self.process_y(y)
245 
246     def process_x(self, x):
247         x = pad(x)
248         x = (x - self.mean) / self.std
249         x = x[:, :, None]
250         return x
251 
```

```
252     def process_y(self, y):
253         y = pad([[self.class_to_int[c] for c in s] for s in y], val=3,
    dtype=np.int32)
254         y = keras.utils.np_utils.to_categorical(
255                 y, num_classes=len(self.classes))
256         return y
257 
258 def pad(x, val=0, dtype=np.float32):
259     max_len = max(len(i) for i in x)
260     padded = np.full((len(x), max_len), val, dtype=dtype)
261     for e, i in enumerate(x):
262         padded[e, :len(i)] = i
263     return padded
264 
265 def compute_mean_std(x):
266     x = np.hstack(x)
267     return (np.mean(x).astype(np.float32),
268             np.std(x).astype(np.float32))
269 
270 def load_dataset(data_json):
271     with open(data_json, 'r') as fid:
272         data = [json.loads(l) for l in fid]
273     labels = []; ecgs = []
274     for d in data:
275         labels.append(d['labels'])
276         ecgs.append(load_ecg(d['ecg']))
277     return ecgs, labels
278 
279 def load_ecg(record):
```

```
280     if os.path.splitext(record)[1] == ".npy":
281         ecg = np.load(record)
282     elif os.path.splitext(record)[1] == ".mat":
283         ecg = sio.loadmat(record)['val'].squeeze()
284     else: # Assumes binary 16 bit integers
285         with open(record, 'r') as fid:
286             ecg = np.fromfile(fid, dtype=np.int16)
287 
288     trunc_samp = STEP * int(len(ecg) / STEP)
289     return ecg[:trunc_samp]
290 
291 train = load_dataset("train.json")
292 
293 dev = load_dataset("dev.json")
294 
295 preproc =Preproc(*train)
296 
297 
298 
299 def make_save_dir(dirname, experiment_name):
300     start_time = str(int(time.time())) + '-' + str(random.randrange
    (1000))
301     save_dir = os.path.join(dirname, experiment_name, start_time)
302     if not os.path.exists(save_dir):
303         os.makedirs(save_dir)
304     return save_dir
305 
306 save_dir=r'/scratch/zw89669/ECGMODELCP/'
307 
```

```
def get_filename_for_saving(save_dir):
    return os.path.join(save_dir,
            "{val_loss:.3f}-{val_acc:.3f}-{epoch:03d}-{loss:.3f}-{acc
   :.3f}.justl1reglinf.1ADDweights512sq.hdf5")
checkpointer = keras.callbacks.ModelCheckpoint(
        filepath=get_filename_for_saving(save_dir),
        save_best_only=False,save_weights_only=True)


params={
    "conv_subsample_lengths": [1,2,1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
   1, 2, 1, 2],
    "conv_filter_length": 16,
    "conv_num_filters_start": 32,
    "conv_init": "he_normal",
    "conv_activation": "relu",
    "conv_dropout": 0.2,
    "conv_num_skip": 2,
    "conv_increase_channels_at": 4,

    "learning_rate": 0.001,
    "batch_size": 16,


    "generator": True
}


```

```
335 params.update({
336         "input_shape": [None, 1],
337         "num_categories": len(preproc.classes)})
338
339 model = build_network(**params)
340
341
342 stopping = keras.callbacks.EarlyStopping(patience=8)
343
344
345 reduce_lr = keras.callbacks.ReduceLROnPlateau(
346         factor=0.1,
347         patience=2,
348         min_lr=params["learning_rate"] * 0.001)
349
350
351 batch_size = params.get("batch_size", 32)
352
353
354 MAX_EPOCHS = 100
355
356 if params.get("generator",False):
357         train_gen = data_generator(batch_size, preproc, *train)
358         dev_gen = data_generator(batch_size, preproc, *dev)
359         model.fit_generator(
360             train_gen,
361             steps_per_epoch=int(len(train[0]) / batch_size),
362             epochs=MAX_EPOCHS,
363             validation_data=dev_gen,
```

```
            validation_steps=int(len(dev[0]) / batch_size),
            callbacks=[checkpointer, reduce_lr, stopping],verbose=1)
else:
    train_x, train_y = preproc.process(*train)
    dev_x, dev_y = preproc.process(*dev)
    model.fit(
        train_x, train_y,
        batch_size=batch_size,
        epochs=MAX_EPOCHS,
        validation_data=(dev_x, dev_y),
        callbacks=[checkpointer, reduce_lr, stopping],verbose=1)
```

Evaluate Model

```
import json
import keras
import numpy as np
import scipy.io as sio
import scipy.stats as sst
import os
import sys
import tensorflow as tf
import scipy.stats as sst
import sklearn.metrics as skm
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support,
   confusion_matrix, classification_report
import sklearn.metrics as skm
import itertools

```

```
16
17
18 STEP = 256
19 from keras import backend as K
20 from keras.layers import BatchNormalization
21 from keras.layers import Add
22 def _bn_relu(layer, dropout=0, **params):
23
24     from keras.layers import Activation
25     layer = BatchNormalization()(layer)
26     layer = Activation(params["conv_activation"])(layer)
27
28     if dropout > 0:
29         from keras.layers import Dropout
30         layer = Dropout(params["conv_dropout"])(layer)
31
32     return layer
33
34 def add_conv_weight(
35         layer,
36         filter_length,
37         num_filters,
38         subsample_length=1,
39         **params):
40     from keras.layers import Conv1D
41     layer = Conv1D(
42         filters=num_filters,
43         kernel_size=filter_length,
44         strides=subsample_length,
```

```
45          padding='same',
46          kernel_initializer=params["conv_init"])(layer)
47      return layer
48
49
50  def add_conv_layers(layer, **params):
51      for subsample_length in params["conv_subsample_lengths"]:
52          layer = add_conv_weight(
53                      layer,
54                      params["conv_filter_length"],
55                      params["conv_num_filters_start"],
56                      subsample_length=subsample_length,
57                      **params)
58          layer = _bn_relu(layer, **params)
59      return layer
60
61  def resnet_block(
62          layer,
63          num_filters,
64          subsample_length,
65          block_index,
66          **params):
67
68      from keras.layers import MaxPooling1D
69      from keras.layers.core import Lambda
70
71      def zeropad(x):
72          y = K.zeros_like(x)
73          return K.concatenate([x, y], axis=2)
```

```
74
75      def zeropad_output_shape(input_shape):
76          shape = list(input_shape)
77          assert len(shape) == 3
78          shape[2] *= 2
79          return tuple(shape)
80
81      shortcut = MaxPooling1D(pool_size=subsample_length)(layer)
82      zero_pad = (block_index % params["conv_increase_channels_at"]) == 0
        \
83          and block_index > 0
84      if zero_pad is True:
85          shortcut = Lambda(zeropad, output_shape=zeropad_output_shape)(
    shortcut)
86
87      for i in range(params["conv_num_skip"]):
88          if not (block_index == 0 and i == 0):
89              layer = _bn_relu(
90                  layer,
91                  dropout=params["conv_dropout"] if i > 0 else 0,
92                  **params)
93          layer = add_conv_weight(
94              layer,
95              params["conv_filter_length"],
96              num_filters,
97              subsample_length if i == 0 else 1,
98              **params)
99      layer = Add()([shortcut, layer])
100     return layer
```

```
101
102 def get_num_filters_at_index(index, num_start_filters, **params):
103     return 2**int(index / params["conv_increase_channels_at"]) \
104         * num_start_filters
105
106 def add_resnet_layers(layer, **params):
107     layer = add_conv_weight(
108         layer,
109         params["conv_filter_length"],
110         params["conv_num_filters_start"],
111         subsample_length=1,
112         **params)
113     layer = _bn_relu(layer, **params)
114     for index, subsample_length in enumerate(params["
    conv_subsample_lengths"]):
115         num_filters = get_num_filters_at_index(
116             index, params["conv_num_filters_start"], **params)
117         layer = resnet_block(
118             layer,
119             num_filters,
120             subsample_length,
121             index,
122             **params)
123     layer = _bn_relu(layer, **params)
124     return layer
125
126 def add_output_layer(layer, **params):
127     from keras.layers.core import Dense, Activation
128     from keras.layers.wrappers import TimeDistributed
```

```
129     layer = TimeDistributed(Dense(params["num_categories"]))(layer)
130     layer = BatchNormalization()(layer)
131     shortcut=layer
132     layer = Activation(params["conv_activation"])(layer)
133     layer = TimeDistributed(Dense(params["num_categories"]))(layer)
134     layer = Add()([shortcut, layer])
135     return Activation('softmax')(layer)
136 
137 def add_compile(model, **params):
138     from keras.optimizers import Adam
139     optimizer = Adam(
140         lr=params["learning_rate"],
141         clipnorm=params.get("clipnorm", 1))
142 
143     model.compile(loss='categorical_crossentropy',
144                   optimizer=optimizer,
145                   metrics=['accuracy'])
146 
147 def build_network(**params):
148     from keras.models import Model
149     from keras.layers import Input
150     inputs = Input(shape=params['input_shape'],
151                    dtype='float32',
152                    name='inputs')
153 
154     if params.get('is_regular_conv', False):
155         layer = add_conv_layers(inputs, **params)
156     else:
157         layer = add_resnet_layers(inputs, **params)
```

```
158
159     output = add_output_layer(layer, **params)
160     model = Model(inputs=[inputs], outputs=[output])
161     if params.get("compile", True):
162         add_compile(model, **params)
163     return model
164
165
166 def load_dataset(data_json):
167     with open(data_json, 'r') as fid:
168         data = [json.loads(l) for l in fid]
169     labels = []; ecgs = []
170     for d in data:
171         labels.append(d['labels'])
172         ecgs.append(load_ecg(d['ecg']))
173     return ecgs, labels
174
175
176
177
178 def load_ecg(record):
179     if os.path.splitext(record)[1] == ".npy":
180         ecg = np.load(record)
181     elif os.path.splitext(record)[1] == ".mat":
182         ecg = sio.loadmat(record)['val'].squeeze()
183     else: # Assumes binary 16 bit integers
184         with open(record, 'r') as fid:
185             ecg = np.fromfile(fid, dtype=np.int16)
186
```

```
187
188
189 class Preproc:
190
191     def __init__(self, ecg, labels):
192         self.mean, self.std = compute_mean_std(ecg)
193         self.classes = sorted(set(l for label in labels for l in label)
    )
194         self.int_to_class = dict( zip(range(len(self.classes)), self.
    classes))
195         self.class_to_int = {c : i for i, c in self.int_to_class.items
    ()}
196
197     def process(self, x, y):
198         return self.process_x(x), self.process_y(y)
199
200     def process_x(self, x):
201         x = pad(x)
202         x = (x - self.mean) / self.std
203         x = x[:, :, None]
204         return x
205
206     def process_y(self, y):
207         y = pad([[self.class_to_int[c] for c in s] for s in y], val=3,
    dtype=np.int32)
208         y = keras.utils.np_utils.to_categorical(
209                 y, num_classes=len(self.classes))
210         return y
211
```

```
212 def pad(x, val=0, dtype=np.float32):
213     max_len = max(len(i) for i in x)
214     padded = np.full((len(x), max_len), val, dtype=dtype)
215     for e, i in enumerate(x):
216         padded[e, :len(i)] = i
217     return padded
218 
219 def compute_mean_std(x):
220     x = np.hstack(x)
221     return (np.mean(x).astype(np.float32),
222             np.std(x).astype(np.float32))
223 
224 def load_dataset(data_json):
225     with open(data_json, 'r') as fid:
226         data = [json.loads(l) for l in fid]
227     labels = []; ecgs = []
228     for d in data:
229         labels.append(d['labels'])
230         ecgs.append(load_ecg(d['ecg']))
231     return ecgs, labels
232 
233 def load_ecg(record):
234     if os.path.splitext(record)[1] == ".npy":
235         ecg = np.load(record)
236     elif os.path.splitext(record)[1] == ".mat":
237         ecg = sio.loadmat(record)['val'].squeeze()
238     else: # Assumes binary 16 bit integers
239         with open(record, 'r') as fid:
240             ecg = np.fromfile(fid, dtype=np.int16)
```

```
241
242     trunc_samp = STEP * int(len(ecg) / STEP)
243     return ecg[:trunc_samp]
244
245 model_path=r'/scratch/zw89669/TELCP/0.104-0.999-028-0.100-1.000.reg2.1
    ADDweights256sq.hdf5'
246
247 train = load_dataset("train.json")
248
249 dev = load_dataset("dev.json")
250
251 preproc =Preproc(*train)
252
253 params={
254    "conv_subsample_lengths": [1,2,1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
    1, 2],
255     "conv_filter_length": 16,
256     "conv_num_filters_start": 32,
257     "conv_init": "he_normal",
258     "conv_activation": "relu",
259     "conv_dropout": 0.2,
260     "conv_num_skip": 2,
261     "conv_increase_channels_at": 4,
262
263     "learning_rate": 0.001,
264     "batch_size": 16,
265
266     "generator": True,
267
```

```
}


params.update({
        "input_shape": [None, 1],
        "num_categories": len(preproc.classes)})



def make_json(save_path, dataset):
    with open(save_path, 'w') as fid:
        for d in dataset:
            datum = {'ecg' : d[0],
                     'labels' : d[1]}
            json.dump(datum, fid)
            fid.write('\n')



model = build_network(**params)

model.load_weights(model_path)
def load_all(data_path):
    label_file = data_path+ "/Refrence.csv"
    with open(label_file, 'r') as fid:
        records = [l.strip().split(",") for l in fid]
    dataset = []
    for record, label in records:
```

```
297         ecg_file = os.path.join(data_path, record + ".npy")
298         ecg_file = os.path.abspath(ecg_file)
299         ecg = load_ecg_mat(ecg_file)
300         num_labels = ecg.shape[0] / STEP
301         dataset.append((ecg_file, [label]*int(num_labels)))
302     return dataset
303
304
305
306 def load_ecg_mat(ecg_file):
307     if os.path.splitext(ecg_file)[1] == ".npy":
308         ecg = np.load(ecg_file)
309     elif os.path.splitext(ecg_file)[1] == ".mat":
310         ecg = sio.loadmat(ecg_file)['val'].squeeze()
311     else: # Assumes binary 16 bit integers
312         with open(ecg_file, 'r') as fid:
313             ecg = np.fromfile(fid, dtype=np.int16)
314     return ecg
315
316
317 data_path = "dev.json"
318
319
320 #For evaluating on test set
321 #hour=21
322 #data_path=r"/scratch/zw89669/TelemetryData"+str(hour)
323 #dataset=load_all(data_path)
324 #make_json("test.json", dataset)
325 #data_path="test.json"
```

```
326 data = load_dataset(data_path)
327
328
329 probs = []
330 labels = []
331 for x, y  in zip(*data):
332     x, y = preproc.process([x], [y])
333     probs.append(model.predict(x))
334     labels.append(y)
335 preds = []
336 ground_truth = []
337 for p, g in zip(probs, labels):
338     preds.append(sst.mode(np.argmax(p, axis=2).squeeze())[0][0])
339     ground_truth.append(sst.mode(np.argmax(g, axis=2).squeeze())[0][0])
340 report = skm.classification_report(
341         ground_truth, preds,
342         target_names=preproc.classes,
343         digits=3)
344 scores = skm.precision_recall_fscore_support(
345                 ground_truth,
346                 preds,
347                 average=None)
348 CM=confusion_matrix(ground_truth, preds)
349 print(CM)
350
351 print(report)
352
353 def plot_confusion_matrix(cm, classes,
354                           normalize=False,
```

```
                          title='Confusion Matrix On Development Set',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)
    fig = plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

```

```
383     plt.ylabel('True label')
384     plt.xlabel('Predicted label')
385     plt.tight_layout()
386     fig.savefig('COnfusionMatrixregDEV')
387 
388 
389 plot_confusion_matrix(CM,['Pre-Infection', 'Liver Stage'])
```