ECG-Based Abnormal Heartbeat Identification:

A Deep Learning Approach for Arrhythmia Detection

Objective

Problem:

Electrocardiograms (ECG) have created a profound impact in the field of cardiology, specifically in recognizing **heart arrhythmias**, a problem with the rhythm of one's heartbeat. Non-invasive arrhythmia analysis is based on multiple electrodes that reflect the electrical activity on ECGs. An estimated **three million cases** of arrhythmia occur in the United States yearly (Mayo Clinic). Diagnosing this disease early is the key to one's wellness, yet **18%** of ECGs containing Atrial Fibrillation are misinterpreted by cardiologists (Anh et al, 2006).

Purpose:

With the recent advancements in technology, Machine Learning algorithms such as **Deep Neural Networks** (DNNs) and **Convolutional Neural Networks** (CNNs), allow a mathematical model to learn features and identify patterns within a given dataset. Hence, making it possible to **autonomously** recognize diseases in ECGs, capable of identifying arrhythmias to the **accuracy** of Cardiologists.

Question:

Is it possible to create a model capable of **surpassing** the **accuracy** of **Cardiologists** in identifying heart **arrhythmias** in Electrocardiograms?

Hypothesis:

It is **possible** to **exceed** the **accuracy** of **Cardiologists** when compared to that of a Convolutional Neural Network's, to identify heart arrhythmias in Electrocardiograms (ECGs).

Variables

Constants	• Raw **training data**
Manipulated Variables	• **Hyper Parameters** in each layer • **Layers** in the model • Level of **data augmentation**
Responding Variables	• **Loss** of the model • **Accuracy** of the model

What is Deep Learning?

- Deep Learning is a subclass of Machine Learning, which is inspired by a **neuron's structure**, and **function** in the brain, groups of neurons are called **Neural Networks**.
- The first layer of a Neural Network is called the **input layer and** is composed of neurons that represent the input data.
- A **neuron** holds a number (often between 0-1), the number corresponds to the activation of each neuron.
- The layers in the middle are **hidden layers**. These layers contain neurons that are responsible for identifying **features** within the dataset.
- The activations in neurons of each layer change to correctly predict the right class.
- The last layer of a Neural Network is called the **output layer**, which contains a neuron for each **class** in the dataset. Each neuron's activation signifies the model's certainty for that class. Hence, the largest activation in the output layer resembles the model's most confident output.

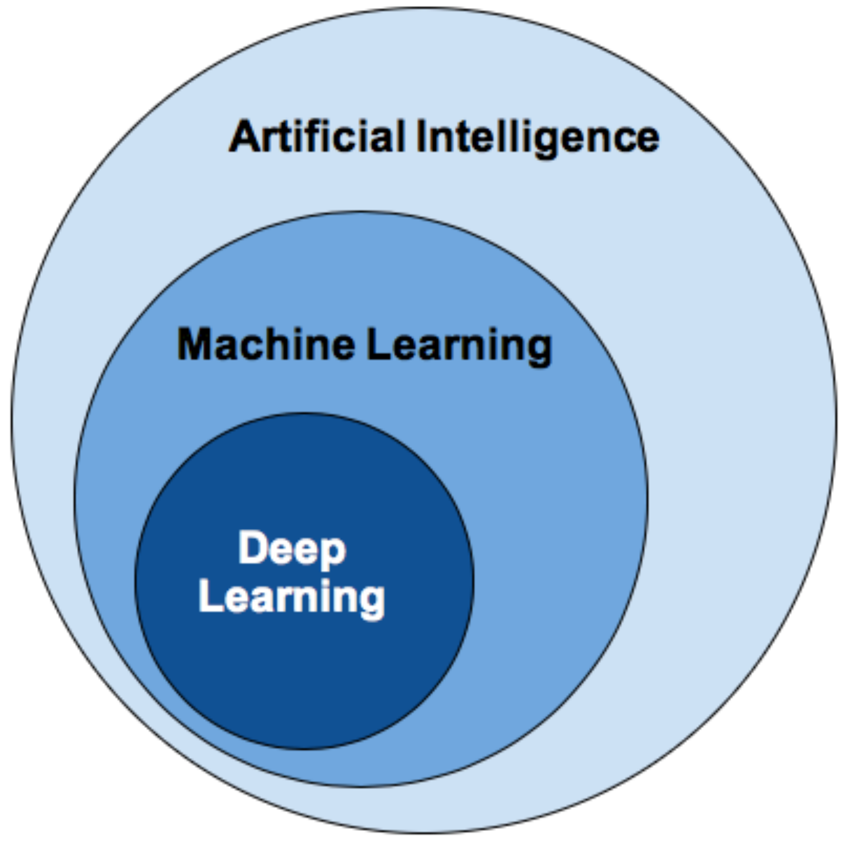


Figure 1.1: Diagram of classes in Artificial Intelligence

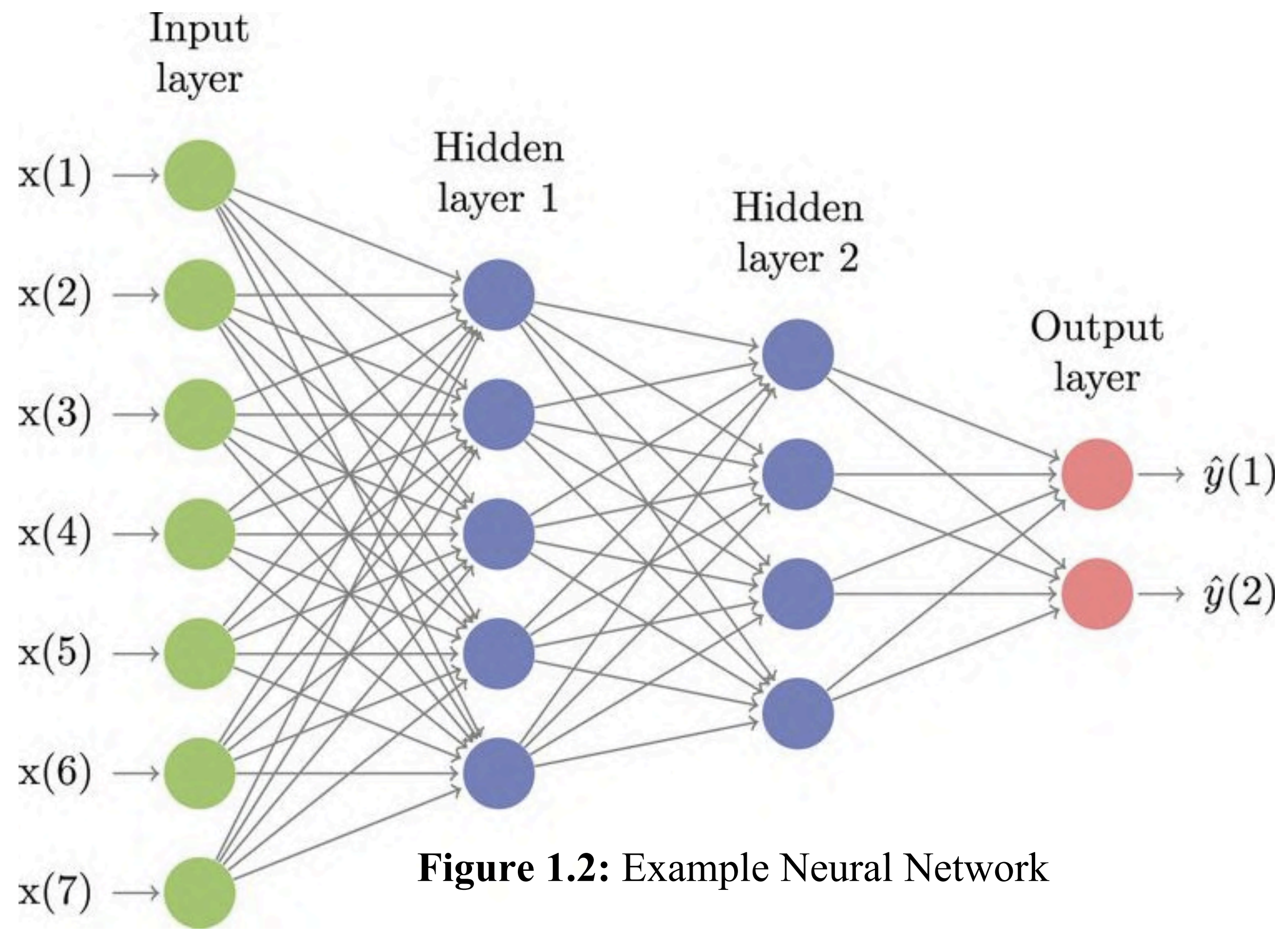


Figure 1.2: Example Neural Network

Methods

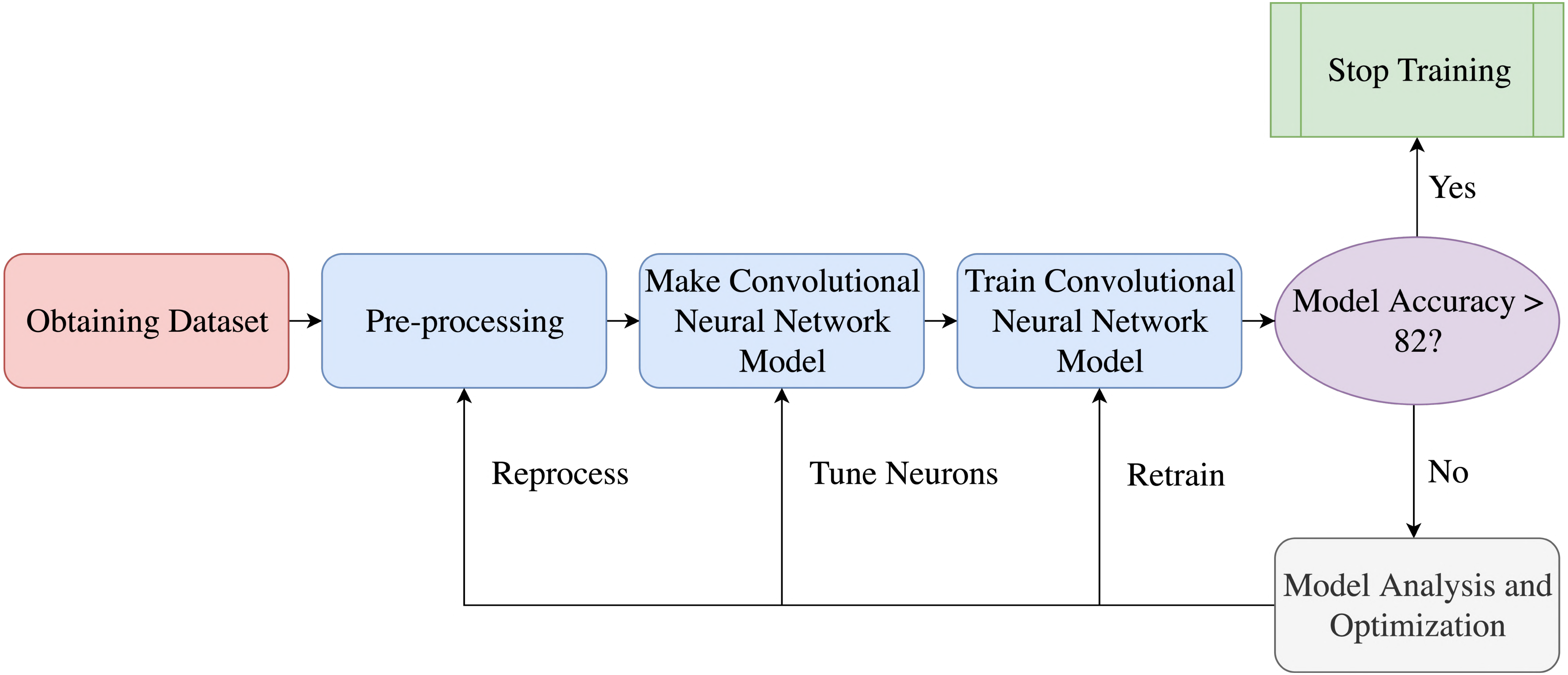


Figure 2.1: Flowchart of methods

Database

The **PhysioNet** database contains 8,522 ECG recordings, divided into 4 classes: **Normal**, **Atrial Fibrillation**, **Other**, and **Noisy**. The raw data is provided in EFDB-compliant MATLAB V4 files, which including a *.mat* file containing the ECG recording and a *.hea* file containing the metadata for the recording (Clifford, et al, 2017).

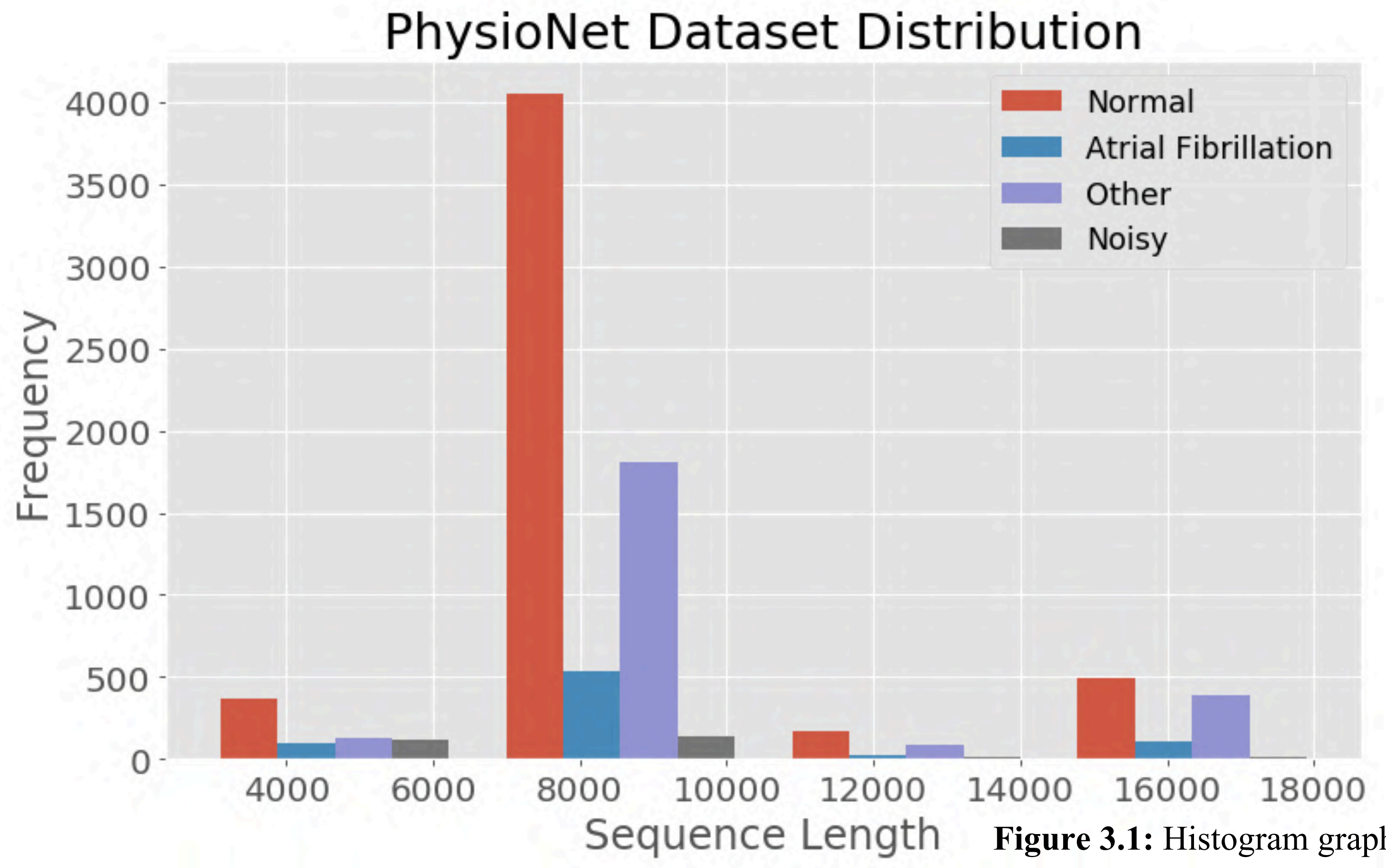


Figure 3.1: Histogram graph of frequency and sequence length of the PhysioNet dataset

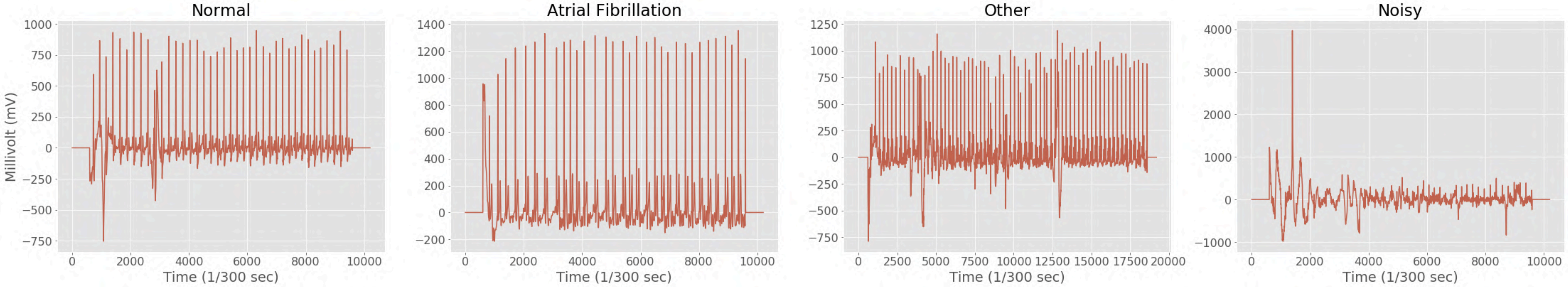


Figure 3.2: Example data from dataset for each class

Pre-processing Data

- Neural Networks require a **constant input vector length**. Hence, the raw ECG data was split into sequences, each with a length of **600 indices**
- These slices were based on each peak in an ECG. The peak is considered the middle of the sequence, and a margin of 300 indices on each side of the peak creates a full sequence
- Each ECG sequence was **normalized** to values between 0 and 1 to create **uniformity** in the dataset
- To create an unbiased model, all classes (e.g. Noisy) in the training data should contain an equal amount of sequences. Thus, the dataset was dramatically reduced to create a fully balanced distribution

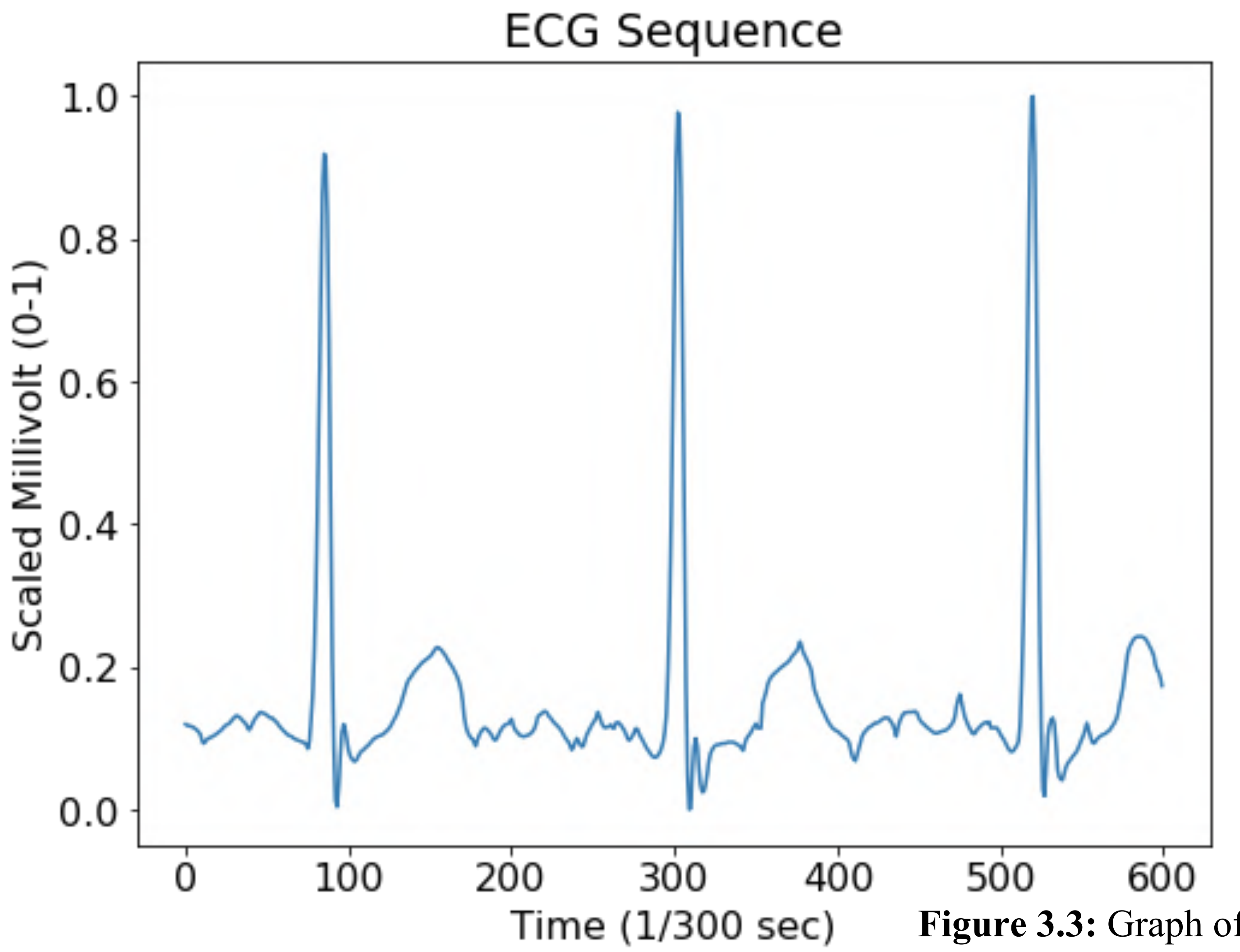


Figure 3.3: Graph of ECG sequence with length of 600

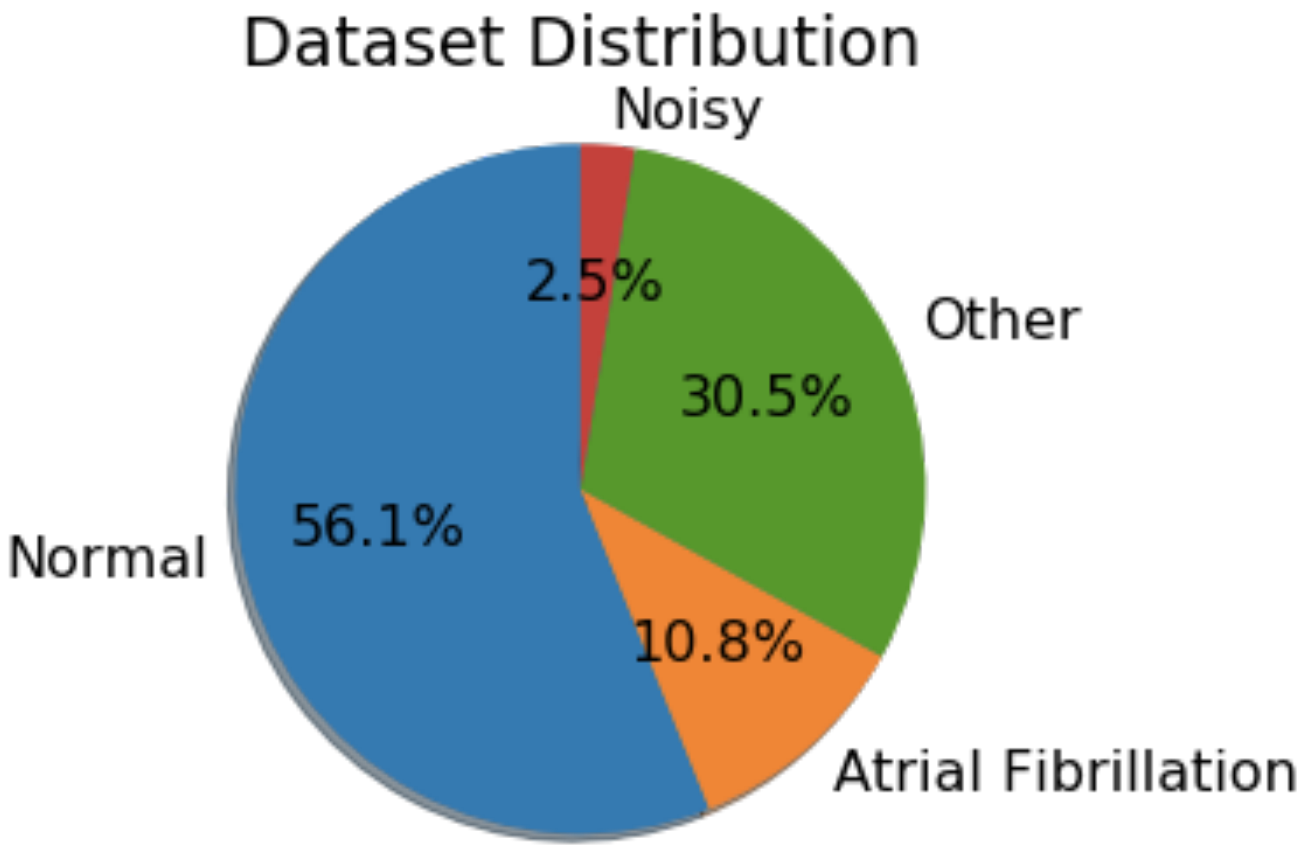


Figure 3.4: Pie chart illustrating the dataset distribution

Python Implementation:

```
def  pre_processing_data(self, AUGMEN_NUN):
    for records in self.LABELS:
      with open(records) as record:
        for ecg_file in tqdm(record):
          path = self.DATA+ecg_file[:-1]
          metadata = open(path+".hea", "r").read().split(" ")
          ECGs = list(loadmat(path)['val'][0])
          for i in range(int(self.ECG_LENGTH+1)):
            ECGs.insert(i, 0)
            ECGs.append(0)
          peaks = detect_beats(ECGs, float(metadata[2]))

          for peak in range(0, len(peaks),
            self.ECG_PER_SAMPLE):
            try:
              ECG = ECGs[peaks[peak]-int(self.ECG_LENGTH/2):

      peaks[peak+self.ECG_PER_SAMPLE]+int(self.ECG_LENGTH/2)]
              ECG = self.zero_padding(self.rnd_zero(ECG))
              ECG = (ECG + abs(np.amin(ECG)))
              ECG = ECG / np.amax(ECG)
              self.data.append([np.array(ECG),
            np.eye(len(self.LABELS))[self.LABELS[records]]])

              for _ in range(AUGMEN_NUN):
                aug_ECG = self.zero_padding(
                  self.rnd_zero(self.resampling(ECG)))
                aug_ECG = (aug_ECG + abs(np.amin(aug_ECG)))
                aug_ECG = aug_ECG / np.amax(aug_ECG)
                self.data.append([np.array(aug_ECG),
            np.eye(len(self.LABELS))[self.LABELS[records]]])

            except Exception as e:
              pass
```

Data Augmentation

- A strategy that enables a significant increase in the diversity of data available for training models, without actually collecting new data.
- **Zero Padding**: Appends zeros to the end of an ECG sequence that is not 600 indices in length.
- **Random Zero Bursts**: Implements random zeros in ECG sequences to replicate and constitute noisy data that occurs while collecting a sample
- **Random Resampling**: Changes the sampling rate of an ECG sequence, which *stretches* or *compresses* the sequence

Python Implementation:

```
def zero_padding(self, ECG):
    if len(ECG) > self.ECG_LENGTH:
      return ECG[:self.ECG_LENGTH]
    for _ in range(self.ECG_LENGTH-len(ECG)):
      ECG.append(0)
    return ECG

  def rnd_bursts(self, ECG):
    for _ in range(np.random.randint(7)):
      pos = abs(np.random.randint(abs(len(ECG)-11)))
      dist = abs(np.random.randint(7))
      ECG[pos:pos+dist]=[0]*dist
    return ECG

  def resampling(self, ECG):
    MARGIN = 60
    return signal.resample(ecg,
    abs(np.random.randint(MARGIN)+(self.ECG_LENGTH-MARGIN)))
```

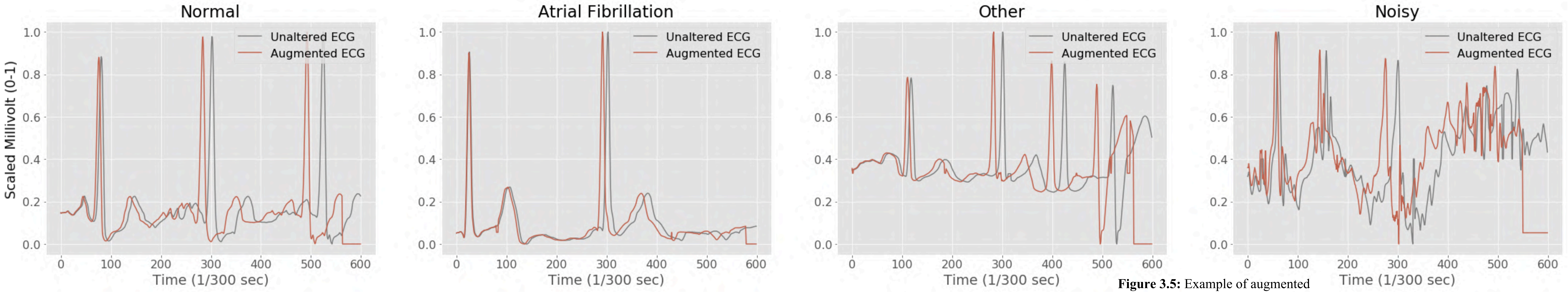


Figure 3.5: Example of augmented ECG for each class

Convolutional Neural Network Model

Convoluting also decreases the size of the input vector. A vector (with a size of 1x5) filters across the data by producing all values in the vector by a filter vector. This method also assists the model in finding features.

The **Rectified Linear** activation function alters the range of the incoming data by setting all numbers below 0 to 0 and leaving all positive numbers intact.

The **Linear Output** layer transforms the Linear layer output into a 1x4. Each column in the tensor represents a class's likelihood of being the correct class in the dataset. Thus, the column with the largest values is the model prediction for the input.

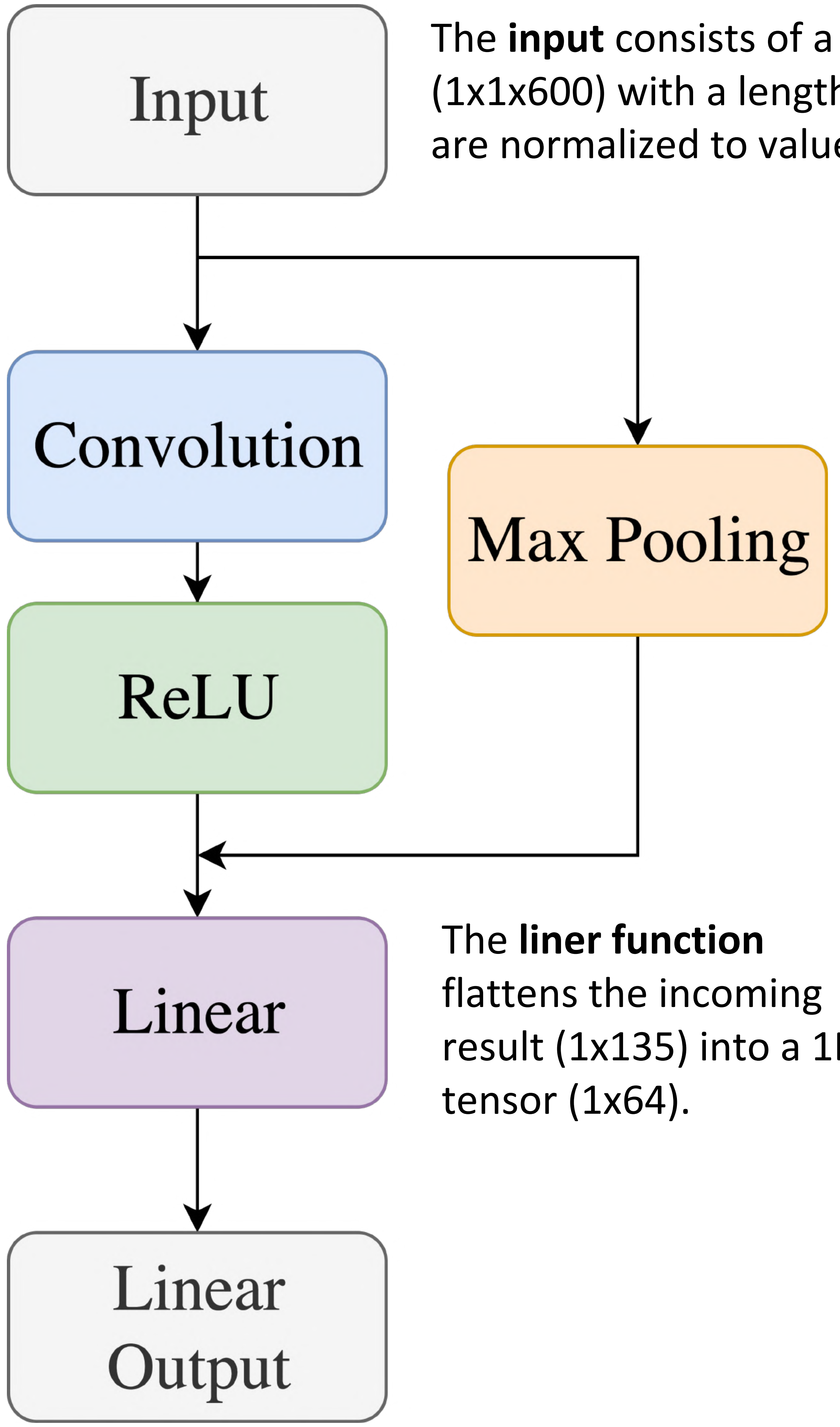


Figure 3.6: Flowchart mapping out the Convolutional Neural Network

The **input** consists of a 1-dimensional tensor (1x1x600) with a length of 600. The elements are normalized to values between 0 and 1.

Max Pooling reduces the size of the input, in the case of a 1D tensor. A kernel block (with a size of 1x3) filters over the 1D tensor. Taking the maximum value in the kernel block. This operation aids the model in finding features in the data.

The **liner function** flattens the incoming result (1x135) into a 1D tensor (1x64).

Python Implementation:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1,180, 5, padding=2)
        self.conv2 = nn.Conv1d(180, 150, 5, padding=2)
        self.conv3 = nn.Conv1d(150, 120, 5, padding=2)
        self.conv4 = nn.Conv1d(120, 90, 5, padding=2)
        self.conv5 = nn.Conv1d(90, 45, 5, padding=2)

        x = torch.randn(1,1,600).view(-1,1,600)
        self._to_linear = None
        self.convs(x)

        self.fc1 = nn.Linear(self._to_linear, 64)
        self.fc2 = nn.Linear(64, 4)

    def convs(self, x):
        x = F.max_pool1d(F.relu(self.conv1(x)), 3)
        x = F.max_pool1d(F.relu(self.conv2(x)), 3)
        x = F.max_pool1d(F.relu(self.conv3(x)), 3)
        x = F.max_pool1d(F.relu(self.conv4(x)), 3)
        x = F.max_pool1d(F.relu(self.conv5(x)), 3)

        if self._to_linear is None:
            self._to_linear = x[0].shape[0]*x[0].shape[1]
        return x

    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Evaluation Metrics

Cross-Entropy Loss:

- Measures how good a prediction from the CNN does in terms of being able to predict the expected outcome.
- Aids the CNN in adjusting the weights and bias of a model

Equation:

$$\mathcal{L}(x, c) = -\ln \frac{e^{x[c]}}{\sum_{i=1}^{N} e^{x[i]}}$$

where…

$\mathrm{N} =$ Number of classes
$\mathrm{c} =$ Index of the correct class
$x =$ Vector of predicted probability of classes

Accuracy:

- Measures the correctness of the CNN's prediction with the ground truth (provided annotation)

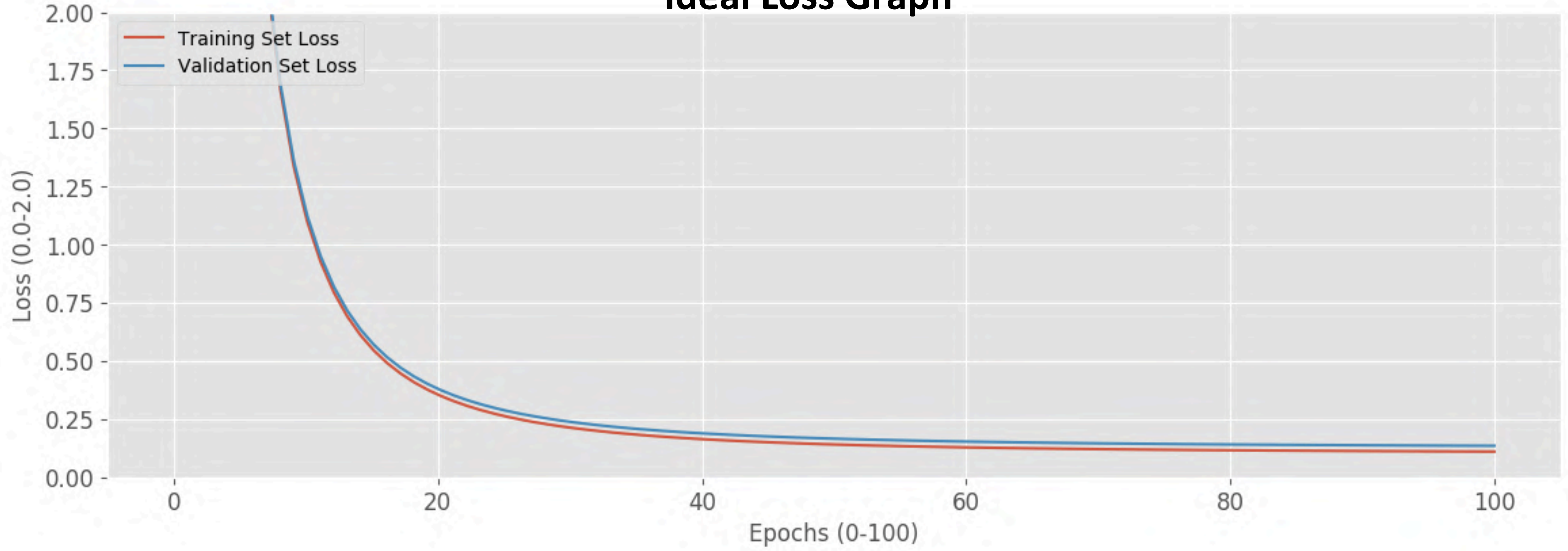
Equation:

$$\overline{A} = \frac{2X_x}{\sum_N + \sum_n}$$

where…

		Predicted Classification				
		Normal	AF	Other	Noisy	*Total*
Reference Classification	Normal	Nn	Na	No	Np	$\sum N$
	AF	An	Aa	Ao	Ap	$\sum A$
	Other	On	Oa	Oo	Op	$\sum O$
	Noisy	Pn	Pa	Po	Pp	$\sum P$
	Total	$\sum n$	$\sum a$	$\sum o$	$\sum p$	

Ideal Loss Graph



Ideal Accuracy Graph

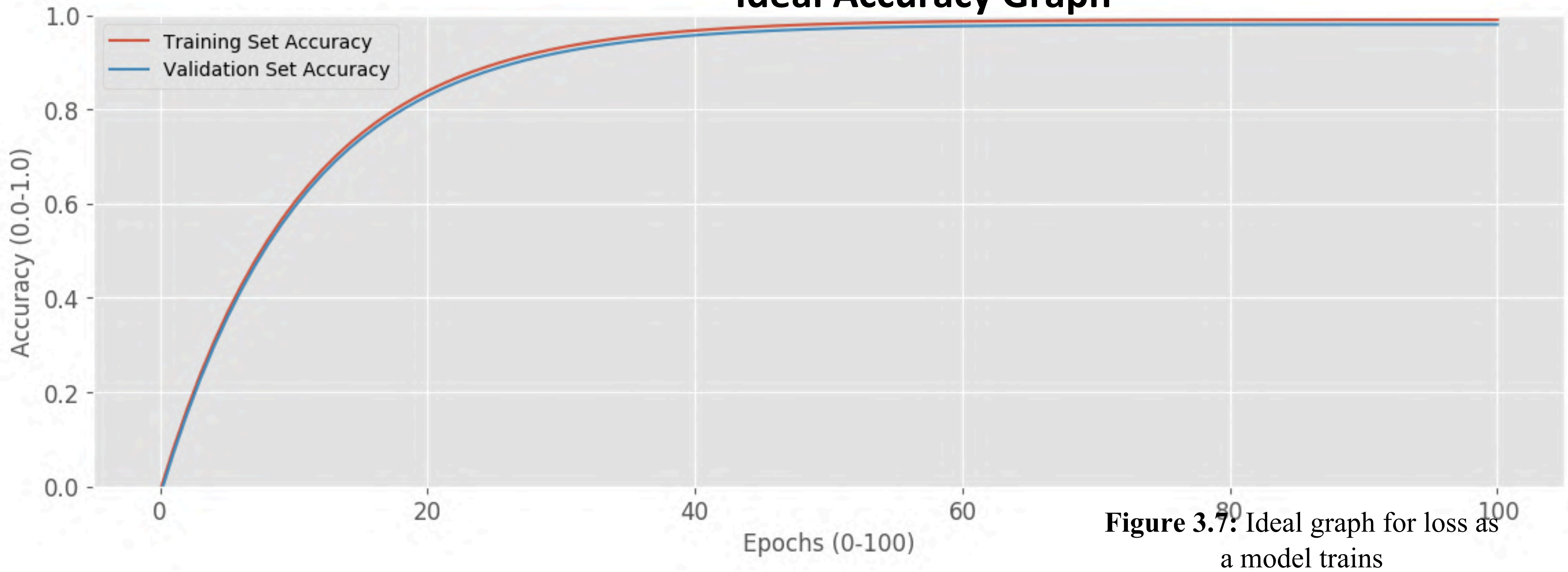


Figure 3.7: Ideal graph for loss as a model trains

Loss and Accuracy

Table 4.1: Raw data table of the CNN metrics **without** data augmentation

Table 1 shows both the training and validation set accuracy and loss of the trained Convolution Neural Network without data augmentation.

Layer	Hyperparameter
Layer 1	45
Layer 2	90
Layer 3	180
Layer 4	4
Total Hyperparameters	**319**

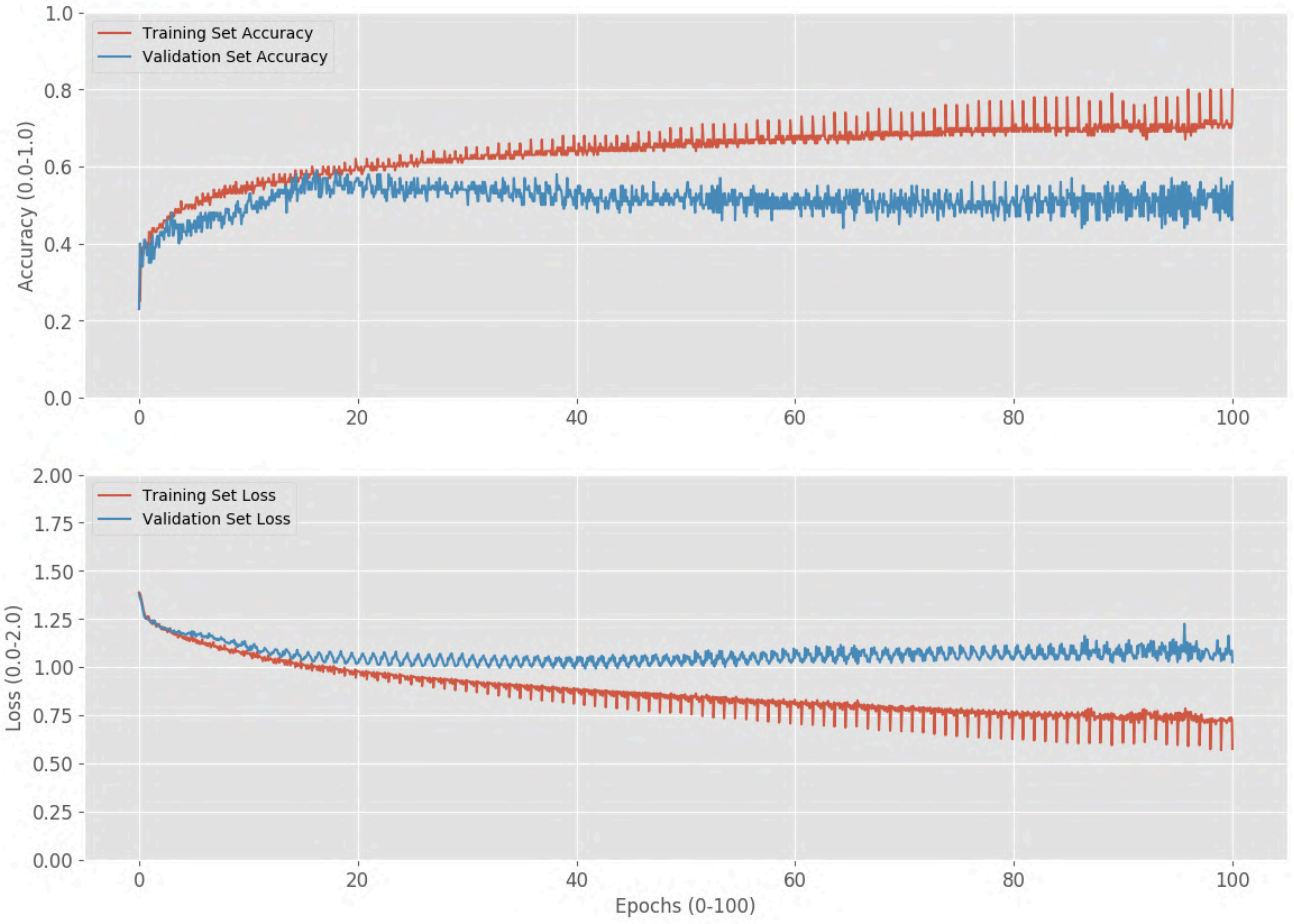
Table 4.2: Data table conveying the amount of layers and hyperparameters in the CNN

Epochs	Training Set Accuracy (%)	Validation Set Accuracy (%)	Training Set Loss	Validation Set Loss
1	0.25	0.23	1.3876	1.3773
5	0.5	0.44	1.1488	1.1835
10	0.54	0.51	1.0705	1.086
15	0.58	0.57	1.0162	1.0405
20	0.6	0.57	0.9802	1.0227
25	0.61	0.54	0.9513	1.0055
30	0.62	0.53	0.9309	1.0013
35	0.63	0.5	0.9047	1.0159
40	0.63	0.49	0.8849	1.0114
45	0.65	0.5	0.8546	1.0546
50	0.67	0.52	0.8288	1.0533
55	0.67	0.52	0.8121	1.0662
60	0.66	0.53	0.8307	1.0484
65	0.69	0.5	0.7906	1.0668
70	0.68	0.49	0.784	1.0693
75	0.69	0.5	0.778	1.0442
80	0.7	0.52	0.7595	1.0377
85	0.7	0.52	0.752	1.0387
90	0.7	0.56	0.7471	1.0332
95	0.69	0.52	0.7615	1.0433
100	0.71	0.46	0.7288	1.0854
ΔTraining & Validation Set	0.13099		0.19536	
Standard Deviation	0.07001	0.03646	0.13428	0.0498
Max Value	0.8	0.59	0.7138	1.0414

Loss and Accuracy

Figure 4.3: Graph of the CNN metrics **without** data augmentation

Figure 3.1 shows that the CNN's accuracy resembles a logarithmic curve. As the validation set's accuracy approaches a horizontal asymptote at 60% and a point of inflection, the accuracy decreases. Likewise, the loss curve mimics an exponential curve, approaching a minimum loss of 1.0.



Loss and Accuracy

Table 4.4: Raw data table of the CNN metrics **with** data augmentation

Table 3 shows both the training and validation set accuracy and loss of the trained Convolution Neural Network with data augmentation.

Epochs	Training Set Accuracy (%)	Validation Set Accuracy (%)	Training Set Loss	Validation Set Loss
1	0.26	0.35	1.3898	1.3703
5	0.72	0.75	0.7206	0.709
10	0.78	0.83	0.5783	0.5382
15	0.83	0.85	0.4604	0.4303
20	0.89	0.88	0.3313	0.3619
25	0.88	0.9	0.3084	0.2827
30	0.89	0.9	0.2741	0.2673
35	0.89	0.87	0.2632	0.3667
40	0.93	0.9	0.1946	0.2877
45	0.94	0.91	0.1614	0.2158
50	0.95	0.93	0.1202	0.2112
55	0.96	0.93	0.0966	0.2344
60	0.97	0.91	0.0877	0.312
65	0.95	0.85	0.1289	0.4028
70	0.97	0.96	0.0781	0.2223
75	0.97	0.91	0.0879	0.2079
80	0.97	0.95	0.0788	0.1301
85	0.97	0.91	0.0793	0.2103
90	0.97	0.89	0.0895	0.2366
95	0.98	0.97	0.0554	0.0833
100	0.97	0.94	0.085	0.1738
ΔTraining & Validation Set	0.03051		0.08152	
Standard Deviation	0.09851	0.08767	0.22844	0.2023
Best Value	1	0.99	0.0554	0.0833

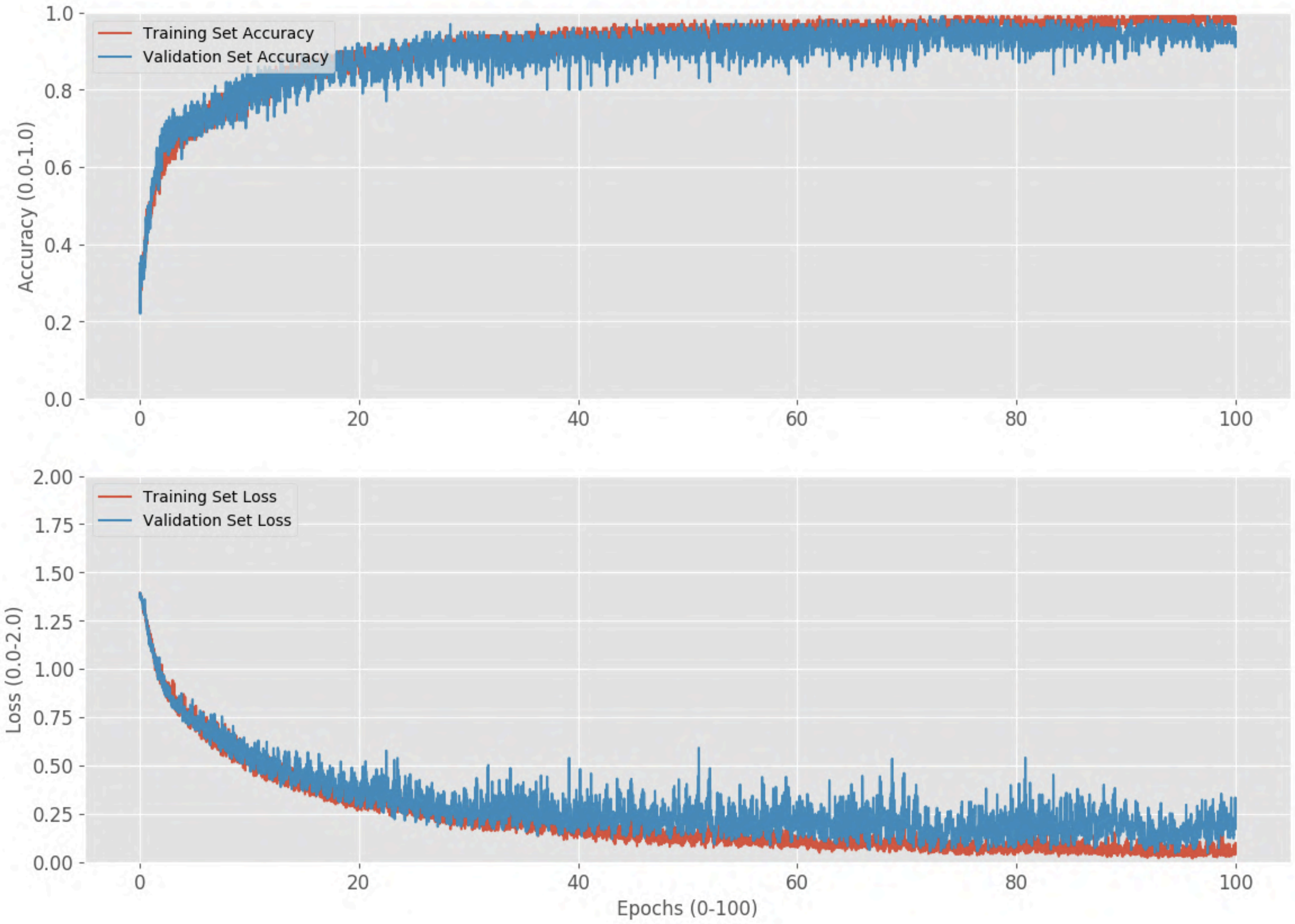
Layer	Hyperparameter
Layer 1	180
Layer 2	150
Layer 3	120
Layer 4	90
Layer 5	45
Layer 6	64
Layer 7	4
Total Hyperparameters	**653**

Table 4.5: Data table conveying the amount of layers and hyperparameters in the CNN

Loss and Accuracy

Figure 4.6: Graph of the CNN metrics **with** data augmentation

Figure 3.2 illustrates that the CNN's training and validation set accuracies correlation to each other, implying the CNN is learning, rather than memorizing the training data. Furthermore, both curves converge at 100%. Although, the validation loss curve is more sporadic, both loss curves tread similarly, and approach a loss of 0.1.



Discussion

- Heart arrhythmias are irregular rhythms in heartbeats that affect 3 million people worldwide every year. Due to the increasing rate of ECGs recording for diagnosis, it is now possible to devolve a Convolutional Neural Network to identify arrhythmias in ECGs. A CNN was developed and trained, to achieve high accuracy in identifying arrhythmias in ECGs. The 1D Convolution Neural Network not only surpassed the accuracy of cardiologists in identifying Atrial Fibrillation, but also achieved an overall top accuracy of 99%, and a constant accuracy of 96%. Furthermore, the CNN was cross-validated against a new dataset that the model had never seen before to ensure no overfitting occurred during the training process. On this test, the CNN model achieved an accuracy of 96%. The key to achieving such success is due to the large annotated dataset (PhysioNet), and data augmentation techniques. Originally, training a shallow CNN with few parameters were thought to create less complexly in learning, and make the CNN faster in training. Doing that merely did the opposite, the model did not learn fast, as the CNN started to overfit to the training data. Adding data augmentation not only fixed the issue of overfitting, but also increased the dataset size; conversely, this increased the time the CNN took to train.

Further Exploration and Application

- Implementing larger datasets with multiple nodes that record the heart’s electrical activity simultaneous
  - Apnea-ECG Database
  - CTU-UHB Intrapartum Cardiotocography Database
  - Fantasia Database
  - MIT-BIH Polysomnographic Database
  - OB-1 Database
- Optimizing the time taken to identify an arrhythmia ECG
  - Allows for faster training and response times
- Apply the CNN to an Electroencephalogram (EEG), which measures neural electrical activity to predict body movement, and thought.
- Creating a portable handheld device that can read and identify if arrhythmias are present in an ECG
- Aid experts in diagnosing cardiovascular diseases which can be seen from ECG signals.
- Discovering new methods in identifying arrhythmias in ECGs
- Implement model in ECG reader to autonomously identify arrhythmias in emergency situations
- Decrease the number of misdiagnosis in arrhythmias

Figures

References

Awerdich. (2020, January 5). awerdich/physionet. Retrieved from https://github.com/awerdich/physionet

Awni. (2019, January 15). awni/ecg. Retrieved from https://github.com/awni/ecg

Brownlee, J. (2019, December 19). What is Deep Learning? Retrieved from https://machinelearningmastery.com/what-is-deep-learning/

Brownlee, J. (2019, August 6). A Gentle Introduction to the Rectified Linear Unit (ReLU). Retrieved from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Bushaev, V. (2018, October 24). Adam - latest trends in deep learning optimization. Retrieved from https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

C-Labpl. (2018, April 30). c-labpl/qrs_detector. Retrieved from https://github.com/c-labpl/qrs_detector

Cao, P., Li, X., Mao, K., Lu, F., Ning, G., Fang, L., & Pan, Q. (2020). A novel data augmentation method to enhance deep neural networks for detection of atrial fibrillation. *Biomedical Signal Processing and Control*, *56*, 101675. doi: 10.1016/j.bspc.2019.101675

Convolutional Neural Networks (LeNet). (n.d.). Retrieved from http://deeplearning.net/tutorial/lenet.html

CVxTz. (2019, May 17). CVxTz/ECG_Heartbeat_Classification. Retrieved from https://github.com/CVxTz/ECG_Heartbeat_Classification

Gao, X. (2019, April 3). Diagnosing Abnormal Electrocardiogram (ECG) via Deep Learning. Retrieved from https://www.intechopen.com/online-first/diagnosing-abnormal-electrocardiogram-ecg-via-deep-learning

References

Gao, X. (2019). Diagnosing Abnormal Electrocardiogram (ECG) via Deep Learning. *Electrocardiography [Working Title]*. doi: 10.5772/intechopen.85509

Huang, L., Pan, W., Zhang, Y., Qian, L., Gao, N., & Wu, Y. (2020). Data Augmentation for Deep Learning-Based Radio Modulation Classification. *IEEE Access*, *8*, 1498–1506. doi: 1912.03026

Krylatov-Pavel. (2019, August 19). krylatov-pavel/aibolit-ECG. Retrieved from https://github.com/krylatov-pavel/aibolit-ECG

Pyakillya, B., Kazachenko, N., & Mikhailovsky, N. (2017). Deep Learning for ECG Classification. *Journal of Physics: Conference Series*, *913*, 012004. doi: 10.1088/1742-6596/913/1/012004

Sakai, A., Minoda, Y., & Morikawa, K. (2017). Data augmentation methods for machine-learning-based classification of bio-signals. *2017 10th Biomedical Engineering International Conference (BMEiCON)*. doi: 10.1109/bmeicon.2017.8229109

Skalski, P. (2019, January 4). Preventing Deep Neural Network from Overfitting. Retrieved from https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a

Wathen, J. E., Rewers, A. B., Yetman, A. T., & Schaffer, M. S. (2005). Accuracy of ECG Interpretation in the Pediatric Emergency Department. *Annals of Emergency Medicine*, *46*(6), 507–511. doi: 10.1016/j.annemergmed.2005.03.013

References

Zhang, X.-R., Lei, M.-Y., & Li, Y. (2018). An Amplitudes-Perturbation Data Augmentation Method in Convolutional Neural Networks for EEG Decoding. *2018 5th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*. doi: 10.1109/iccss.2018.8572304

Zhang, Z., Duan, F., Sole-Casals, J., Dinares-Ferran, J., Cichocki, A., Yang, Z., & Sun, Z. (2019). A Novel Deep Learning Approach With Data Augmentation to Classify Motor Imagery Signals. *IEEE Access*, *7*, 15945–15954. doi: 10.1109/access.2019.2895133

Zihlmann, M., Perekrestenko, D., & Tschannen, M. (2017). Convolutional Recurrent Neural Networks for Electrocardiogram Classification. *2017 Computing in Cardiology Conference (CinC)*. doi: 10.22489/cinc.2017.070-060

Alfaras, Miquel, Soriano, & Silvia. (2019, July 3). A Fast Machine Learning Model for ECG-Based Heartbeat Classification and Arrhythmia Detection., from https://www.frontiersin.org/articles/10.3389/fphy.2019.00103/full.

Mayo Clinic. (2019, April 2). Heart arrhythmia. Retrieved October 30, 2019, from https://www.mayoclinic.org/diseases-conditions/heart-arrhythmia/symptoms-causes/syc-20350668?utm_source=Google&utm_medium=abstract&utm_content=Cardiac- arrhythmia&utm_campaign=Knowledge-panel.

Srinivasan, N. T., & Schilling, R. J. (2018, June). Sudden Cardiac Death and Arrhythmias. Retrieved October 30, 2019, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6020177/.

Thank You!
Any Questions?