Minia University

Faculty of Engineering

CSE Dept.

Detecting Freeze of Gait in Parkinson Patients

A project submitted to the department of Computers and Systems Engineering as a partial fulfillment for a B.Sc. degree

Submitted By

Ahmad Mostafa

Supervisors

Dr. Abdullah Hassan

Dr. Mohammed Elhenawy

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Mohammed Elhenawy, and Dr. Abdullah Hassan, for their support, outstanding guidance and encouragement throughout my senior project.

Index

Abstract:

Freezing of Gait (FoG) is a common symptom experienced by individuals with Parkinson's disease, leading to significant mobility impairments and an increased risk of falls. In recent years, deep learning techniques have shown promise in various medical applications. This study aims to explore the use of deep learning algorithms to detect FoG episodes using inertial measurement unit (IMU) sensors.

Multiple LSTM and shallow models were developed to reach an optimal solution for the problem.

The models were trained on the dataset, using raw sensor readings. To improve generalization, data augmentation techniques were employed, such as scaling.

The trained model achieved promising results in detecting FoG episodes. It demonstrated high accuracy, sensitivity, and specificity in discriminating between FoG and non-FoG instances.

Introduction

Deep-learning in medical applications

Machine learning has emerged as a transformative technology in various industries, and the medical field is no exception. With its ability to analyze vast amounts of data and identify complex patterns, machine learning is revolutionizing healthcare, leading to improved diagnostics, personalized treatment plans, and enhanced patient care.

One of the primary applications of machine learning in medicine is in diagnostics. Machine learning algorithms can analyze medical images, such as X-rays, CT scans, and MRIs, with remarkable accuracy, aiding in the early detection and diagnosis of diseases like cancer, cardiovascular conditions, and neurological disorders. These algorithms learn from large datasets, allowing them to recognize subtle patterns and abnormalities that might be missed by human eyes alone.

Another crucial role of machine learning in medicine is predicting patient outcomes. By leveraging patient data, including medical records, genetic information, and lifestyle factors, machine learning algorithms can generate predictive models. These models help physicians make informed decisions about treatment options, identify high-risk patients who may require additional care, and develop personalized treatment plans tailored to individual patients.

Furthermore, machine learning enables real-time monitoring and early detection of adverse events. By continuously analyzing patient data from wearable devices, sensors, and electronic health records, machine learning algorithms can identify warning signs of deterioration or potential complications. This early warning system allows healthcare professionals to intervene promptly, preventing adverse events and improving patient outcomes.

Machine learning also plays a significant role in drug discovery and development. By analyzing massive datasets of molecular structures, genetic information, and clinical trial results, machine learning algorithms can identify potential drug targets, design novel compounds, and predict drug efficacy and safety profiles. This accelerates the drug development process, reduces costs, and holds the promise of personalized medicine based on an individual's genetic profile.

Inertial Measurement Unit sensors, are powerful devices that play a significant role in various fields. They combine multiple sensors, including accelerometers, gyroscopes, and magnetometers, to measure an object's orientation, velocity, and gravitational forces. IMUs are widely used in robotics, virtual reality, motion tracking, and navigation systems. These sensors provide vital data for stabilization, gesture recognition, and precise motion control. With their small size, low power consumption, and high accuracy, IMU sensors have become integral components in modern devices such as smartphones, drones, and wearable fitness trackers. They enable advanced applications and enhance user experiences by capturing and analyzing dynamic movements in real-time.

Parkinson Disease:

Parkinson's disease is a neurological disorder that affects millions of people worldwide. Among its various motor symptoms, one particularly intriguing and challenging manifestation is known as freezing of gait (FOG) syndrome. FOG refers to a sudden and temporary inability to initiate or continue walking, resulting in an individual feeling as though their feet are glued to the ground. This symptom can significantly impact a person's mobility, independence, and overall quality of life. In this essay, we will delve into the characteristics, causes, and management strategies for FOG syndrome in Parkinson's disease.

Freezing of gait is a complex phenomenon that goes beyond a simple motor impairment. It often manifests as brief episodes of hesitations or "blocks" during walking, where individuals experience a sensation of being stuck, as if their feet are rooted to the floor. These episodes can occur spontaneously or in response to specific triggers, such as turning, narrow spaces, or stress. FOG episodes are typically brief but can be highly debilitating, leading to falls, injuries, and an increased risk of social isolation.

The underlying mechanisms of FOG in Parkinson's disease are not yet fully understood. However, it is believed to result from a combination of motor, cognitive, and sensory dysfunctions. Dopamine deficiency, which is a hallmark feature of Parkinson's, is thought to play a significant role. It disrupts the communication between different brain regions involved in motor control and coordination. Furthermore, impaired attention, executive functions, and sensory processing contribute to the development of FOG episodes.

Managing FOG syndrome requires a multidimensional approach that addresses both the motor and non-motor aspects of the symptom. Medications, such as levodopa and dopamine agonists, are the primary pharmacological interventions for Parkinson's disease, and they can alleviate FOG to some extent. However, medication effectiveness can vary among individuals, and FOG may persist despite optimal drug therapy.

Physical therapy and rehabilitation programs play a crucial role in managing FOG. Gait training exercises, balance training, and strength training can improve mobility and reduce FOG episodes. Visual and auditory cues, such as lines on the floor or rhythmic sounds, can act as external triggers to help overcome freezing episodes. Assistive devices like canes or walkers can provide support and stability during FOG episodes and reduce the risk of falls.

Non-pharmacological interventions, such as deep brain stimulation (DBS), have shown promise in reducing FOG symptoms. DBS involves implanting electrodes into specific areas of the brain and delivering electrical impulses to modulate abnormal neural activity. While DBS is generally effective in managing motor symptoms, its impact on FOG varies among individuals.

Furthermore, cognitive and psychological interventions, such as cognitive-behavioral therapy and mindfulness-based approaches, can help individuals cope with FOG-related anxiety, depression, and fear of falling. These interventions aim to enhance self-awareness, improve attention, and develop strategies to overcome freezing episodes.

In conclusion, freezing of gait syndrome in Parkinson's disease is a complex and debilitating symptom that significantly impacts the mobility and quality of life of affected individuals. Although its exact mechanisms remain unclear, a combination of motor, cognitive, and sensory dysfunctions contributes to its development. Managing FOG requires a comprehensive approach that includes medication optimization, physical therapy, assistive devices, and non-pharmacological interventions. Further research is needed to gain a deeper understanding of FOG syndrome and develop more effective management strategies to alleviate this challenging symptom.

Using IMU sensors readings for building a deep learning model in detecting FOG

IMU sensors are small devices that integrate accelerometers, gyroscopes, and sometimes magnetometers to measure movement and orientation in three-dimensional space. They can be easily attached to different parts of the body, such as the shins or shoes, to capture detailed movement patterns during walking. IMU sensors provide high-frequency data, enabling the detection of subtle changes in movement characteristics associated with FOG episodes.

Deep learning algorithms, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have shown remarkable success in various pattern recognition tasks, including activity recognition and gait analysis. By training these algorithms on labeled IMU data from individuals with and without FOG, they can learn to distinguish between normal gait and freezing episodes.

The first step in utilizing deep learning for FOG detection is data collection. Participants wear IMU sensors while performing various walking tasks, including normal walking, turning, and activities that trigger FOG episodes. During these tasks, IMU sensors record acceleration, angular velocity, and sometimes magnetic field data. This data is synchronized with labels indicating FOG or non-FOG periods, which can be obtained through manual annotation or automated FOG detection systems.

Once the data is collected, it is preprocessed to remove noise, normalize sensor signals, and extract relevant features. Time-domain and frequency-domain features, such as mean, standard deviation, Fourier transform coefficients, and wavelet coefficients, can capture different aspects of movement patterns during FOG episodes. These features are then fed into the deep learning model as input.

Convolutional neural networks (CNNs) are well-suited for extracting spatial and temporal features from IMU data. They consist of multiple convolutional layers that learn hierarchical representations of the input data. The CNN architecture can be customized to capture specific movement patterns associated with FOG, such as changes in gait velocity, step length, or variability in movement amplitude.

Recurrent neural networks (RNNs) are particularly useful for capturing temporal dependencies in sequential data. They can model the temporal dynamics of FOG episodes by processing IMU data in a sequential manner. Long short-term memory (LSTM) networks, a type of RNN, are commonly used for time-series analysis tasks, including FOG detection. LSTMs can effectively capture long-term dependencies and detect subtle changes in movement patterns over time.

The deep learning model is trained using labeled data, where the network learns to associate specific patterns in the IMU data with FOG or non-FOG periods. The model's performance is evaluated on separate validation and test datasets to assess its accuracy, sensitivity, and specificity in detecting FOG episodes.

One advantage of deep learning approaches is their ability to generalize well to unseen data. Once trained, the model can be deployed on wearable devices, such as smartwatches or smartphones, to continuously monitor and detect FOG episodes in real-time. This real-time detection can provide timely feedback to individuals with FOG, allowing them to take precautionary measures or engage in strategies to overcome freezing, such as using visual or auditory cues.

In conclusion, deep learning techniques combined with IMU sensors offer a promising approach for detecting freezing of gait episodes in individuals with Parkinson's disease.

Kaggle

Kaggle competitions are online data science challenges hosted by the Kaggle platform. They attract data enthusiasts, researchers, and professionals from around the world. Participants are given datasets and problem statements, and they compete to develop the most accurate predictive models or solutions. Kaggle competitions provide an opportunity to apply machine learning, statistics, and data analysis techniques to real-world problems, fostering learning, collaboration, and innovation. They offer a platform for participants to showcase their skills, learn from others, and potentially win prizes or gain recognition within the data science community. Kaggle competitions are a valuable avenue for honing data science expertise and tackling complex problems.

Dataset:

I used the dataset in

kaggle/competitions/tlvmc-parkinsons-freezing-gait-prediction

Kaggle dataset description:

This competition dataset includes lower-back 3D accelerometer data from participants who experienced episodes of freezing of gait, a disabling symptom that is typical in Parkinson's disease patients. Walking abilities are significantly impacted by freezing of gait (FOG), which also affects independence and motility.

Goal is to identify the beginning and end of each episode of freezing as well as the occurrence of the three different types of freezing gait events: start hesitating, turning, and walking.

The data series include three datasets, collected under distinct circumstances:

- *The tDCS FOG (tdcsfog) dataset, comprising data series collected in the lab, as subjects completed a FOG-provoking protocol.*
- *The DeFOG (defog) dataset, comprising data series collected in the subject's home, as subjects completed a FOG-provoking protocol*
- *The Daily Living (daily) dataset, comprising one week of continuous 24/7 recordings from sixty-five subjects. Forty-five subjects exhibit FOG symptoms and also have*

series in the defog dataset, while the other twenty subjects do not exhibit FOG symptoms and do not have series elsewhere in the data.

Because the The tDCS FOG was made in a lab with high frequency and accurate annotation, I decided to use this data

File and Field Descriptions

- *train/ Folder containing the data series in the training set within three subfolders: tdcsfog/, defog/, and notype/. Series in the notype folder are from the defog dataset but lack event-type annotations. The fields present in these series vary by folder.*
  - *Time An integer timestep. Series from the tdcsfog dataset are recorded at 128Hz (128 timesteps per second), while series from the defog and daily series are recorded at 100Hz (100 timesteps per second).*
  - *AccV, AccML, and AccAP Acceleration from a lower-back sensor on three axes: V - vertical, ML - mediolateral, AP - anteroposterior. Data is in units of m/s^2 for tdcsfog/ and g for defog/ and notype/.*
  - *StartHesitation, Turn, Walking Indicator variables for the occurrence of each of the event types.*
  - *Class indicator is to differentiate between normal and FoG occurrences*

Time series analysis is a statistical technique used to analyze and forecast patterns in sequential data points collected over time. One popular approach for time series analysis is using Long Short-Term Memory (LSTM) networks, which are a type of recurrent neural network (RNN) capable of learning long-term dependencies in sequential data.

LSTM networks are particularly effective for time series analysis because they can capture complex temporal patterns and handle data with irregular time intervals. Unlike traditional statistical models, LSTM models can automatically learn and extract relevant features from the data, eliminating the need for manual feature engineering.

Here's a general overview of the steps involved in performing time series analysis using LSTM:

1. **Data Preparation:** The first step is to prepare the time series data for analysis. This involves cleaning the data, handling missing values or outliers, and transforming the data into a suitable format for the LSTM model. Typically, the data is divided into sequences of fixed length, where each sequence consists of input features and a corresponding target value.
2. **Train-Test Split:** The data is then divided into training and testing sets. The training set is used to train the LSTM model, while the testing set is used to evaluate its performance and assess its ability to generalize to unseen data. It is important to maintain the temporal order of the data during the splitting process to ensure realistic evaluation.
3. **Feature Scaling:** LSTM models are sensitive to the scale of the input features. Therefore, it is essential to scale the data before feeding it into the model. Common scaling techniques include normalization (e.g., min-max scaling) or standardization (e.g., z-score normalization).
4. **Model Architecture:** The LSTM model architecture is designed based on the specific time series problem. It typically consists of one or more LSTM layers followed by one or more fully connected (dense) layers. The LSTM layers learn the temporal dependencies in the data, while the dense layers transform the LSTM outputs into the desired output format (e.g., a single value or a sequence of predictions).
5. **Model Training:** The LSTM model is trained on the training data using an optimization algorithm such as stochastic gradient descent (SGD) or Adam. During training, the model learns to minimize a chosen loss function, such as mean squared error (MSE), by adjusting its weights and biases. The training process involves forward propagation, where input sequences are fed into the model, and backward propagation, where the gradients are calculated and used to update the model parameters.

6. **Model Evaluation:** Once the model is trained, it is evaluated on the testing data to assess its performance. Various evaluation metrics can be used, depending on the specific problem, such as mean absolute error (MAE), root mean squared error (RMSE), or accuracy. Visualizing the predicted and actual values can also provide insights into the model's performance.
7. **Model Tuning:** If the model's performance is not satisfactory, it may require further tuning. This can involve adjusting hyperparameters such as the number of LSTM layers, the number of hidden units, the learning rate, or the batch size. It may also involve experimenting with different network architectures or regularization techniques to improve the model's generalization capabilities.
8. **Forecasting:** After the model is trained and validated, it can be used for forecasting future values in the time series. By providing the model with the most recent historical data, it can generate predictions for subsequent time steps. The forecasted values can then be compared to the actual values to assess the accuracy of the predictions.

Time series analysis using LSTM is a powerful technique that has been successfully applied in various domains, such as finance, economics, weather forecasting, and industrial process control. It allows for capturing complex temporal patterns and making accurate predictions, making it a valuable tool for understanding and leveraging sequential data.

Dataset EDA:

Time	**AccV**	**AccML**	**AccAP**	**StartHesitation**	**Turn**	**Walking**	**idx**	**ID**	**len_df**	**class**
0	-9.665890	0.042550	0.184744	0	0	0	0	a171e61840	7400	0
1	-9.672969	0.049217	0.184644	0	0	0	0	a171e61840	7400	0
2	-9.670260	0.033620	0.193790	0	0	0	0	a171e61840	7400	0

Time	**AccV**	**AccML**	**AccAP**	**StartHesitation**	**Turn**	**Walking**	**idx**	**ID**	**len_df**	**class**
3	-9.673356	0.035159	0.184369	0	0	0	0	a171e61840	7400	0
4	-9.671458	0.043913	0.197814	0	0	0	0	a171e61840	7400	0

Table 1 a sample of the dataset

	Time	AccV	AccML	AccAP	StartHesitation	Turn	Walking	class
mean	9.289467e+03	-9.306317e+00	-2.012513e-01	1.808524e+00	4.315506e-02	2.376979e-01	2.942767e-02	3.102806e-01
std	1.399893e+04	1.080174e+00	1.269525e+00	2.285849e+00	2.032061e-01	4.256731e-01	1.690020e-01	4.626084e-01
min	0.000000e+00	-3.552112e+01	-2.616440e+01	-4.782964e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.119000e+03	-9.762402e+00	-9.295446e-01	5.672254e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	4.310000e+03	-9.363524e+00	-1.722245e-01	1.987101e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	8.433000e+03	-8.776814e+00	5.752114e-01	3.449026e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
max	9.707600e+04	2.090695e+01	2.748472e+01	3.033769e+01	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

Table 2: Data numeric description

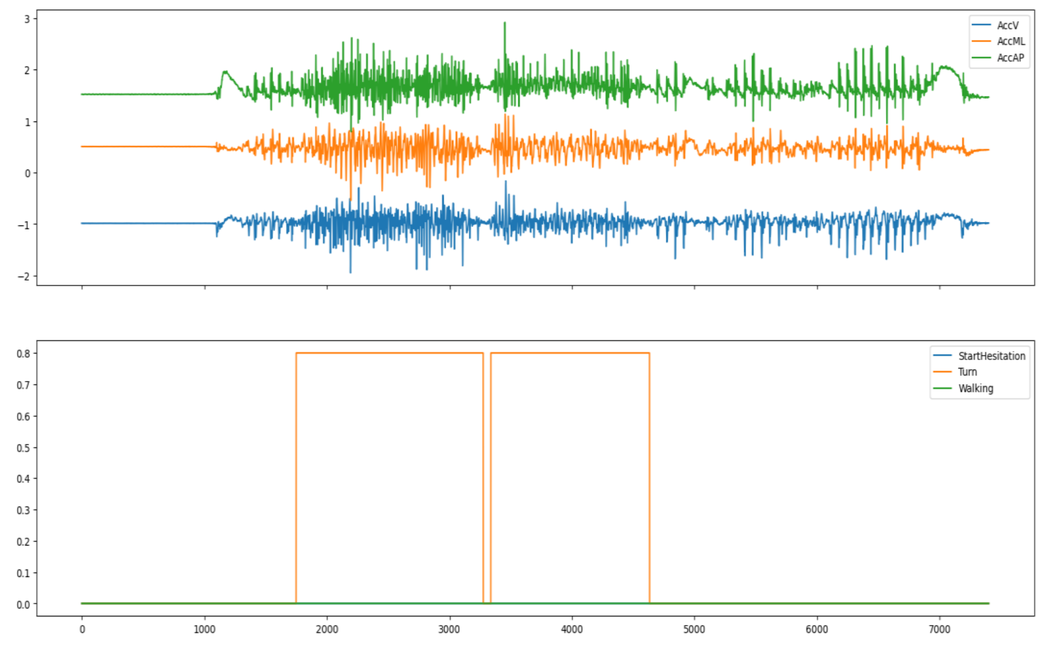


Figure 1 a participant data graph

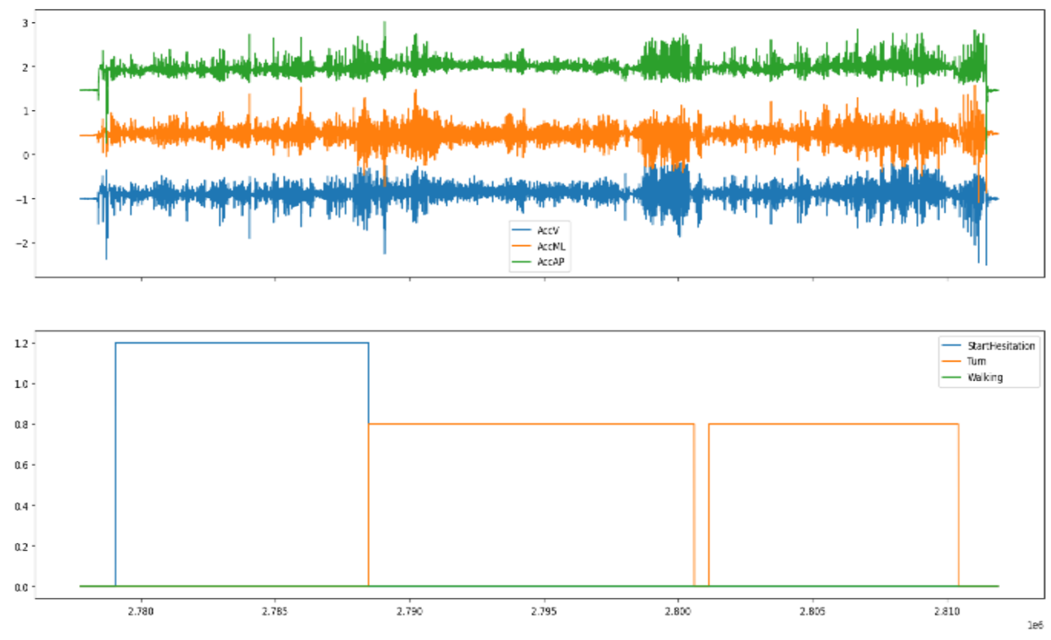


Figure 2 a participant data graph

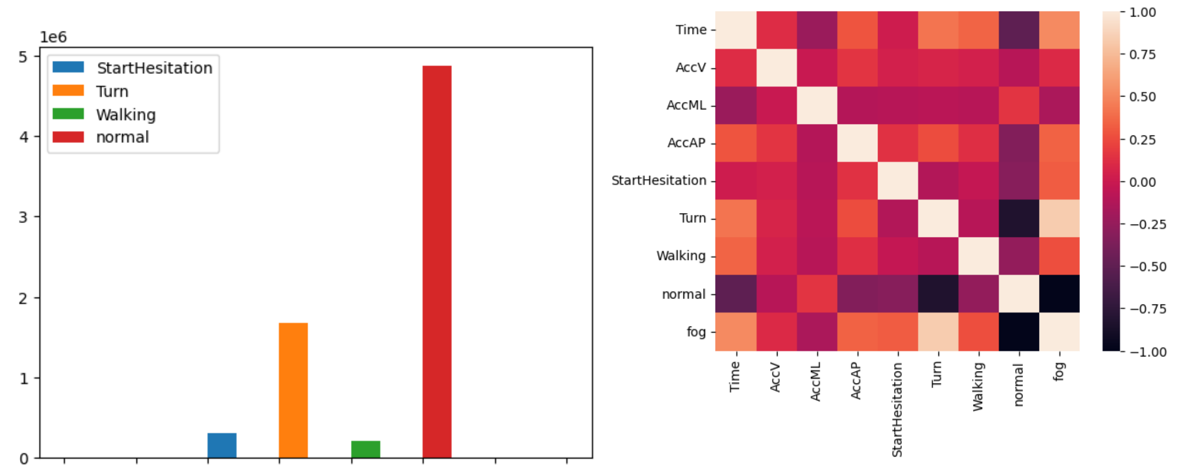


Figure 3 A histogram of each label and a heat map of the feature correlation

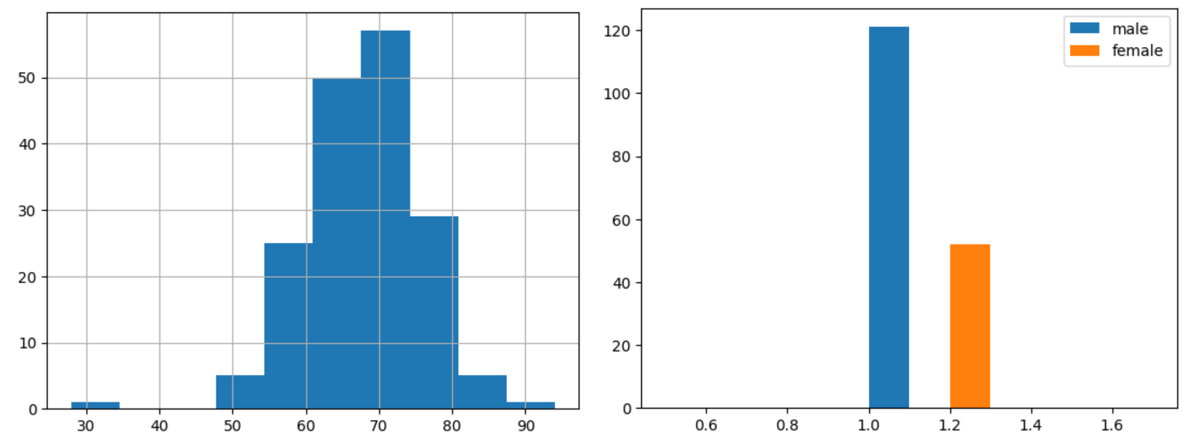


Figure 4 Histograms showing the Age and the Sex of the subjects

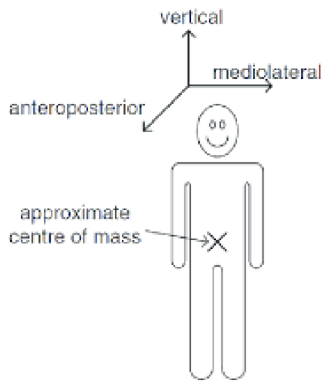


Figure 5 sensor channel names source: https://www.researchgate.net/figure/6-The-directions-of-the-vertical-medio-lateral-and-antero-posterior-axes-with-respect_fig6_305011740

Data Augmentation:

Data had a problem in which the normal sequence is approximately 70% of the data, this kept pushing the models to favor having false negatives.

Scaling had to be done by adding sequences with mode of FoG episode present twice, that balanced the data in most models.

Research questions:

- What is the best model for sequence to label case?
- What is the best model for sequence to sequence case?
- What is the best shallow model for engineered features?
- What is the best time window?
- How small the time window can get?
- What is the best combination of channels?

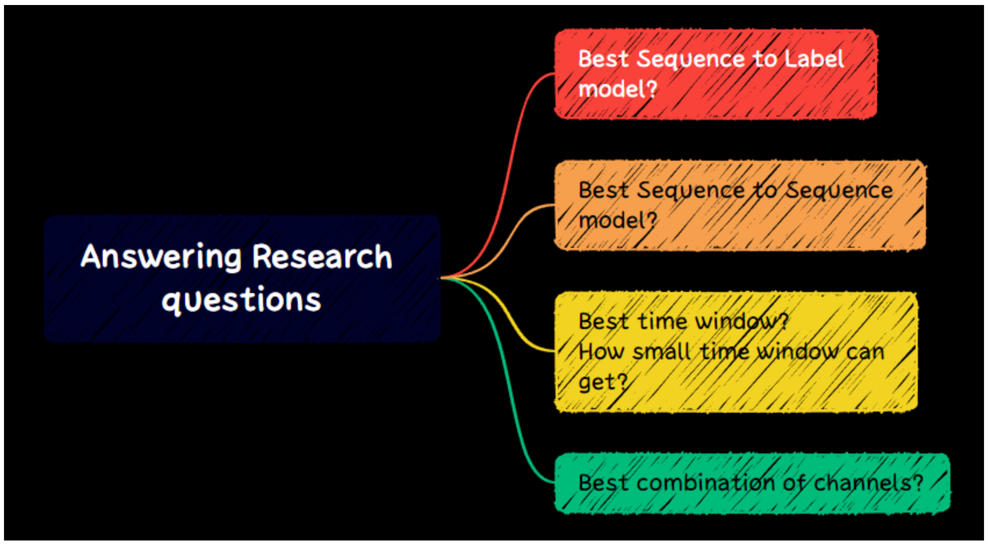

Answering Research questions
Best Sequence to Label model?
Best Sequence to Sequence model?
Best time window? How small time window can get?
Best combination of channels?

Figure 6 Diagram showing Research questions

Methods:

I tried different combinations both with the architecture and the data.

Architecture:

Keras framework was used during training on nvidia-1650GTX 4GB VRam with 1620 MHz clock

The learning rate was modified by trial and error to prevent loss explosion as it kept occurring on too big or too small learning rates, learning rate decay was set to be reduced when accuracy is not improving with patience 5 epochs

Early stopping was set to 10 epochs of not improving accuracy

The loss function was set to categorical cross entropy

$$Loss = -\sum_{i=1}^{output\ size} y_i . log(\hat{y}_i)$$

Standard 20% dropout was used after each layer

Kernel regularization was set to L1 and L2 and Bias regularization was set to L2 to prevent weight explosion or vanishing as it kept occurring

L1 regularization:

$$Cost = \sum_{i=0}^{N}\left(y_i - \sum_{j=0}^{M} x_{ij}W_j\right)^2 + \lambda\sum_{j=0}^{M}|W_j|$$

L2 regularization:

$$Cost = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda\sum_{j=0}^{M} W_j^2$$

The output layer was set to two unit dense layer with softmax activation and used categorical classification instead of binary classification as it acts quite the same with the difference that softmax is preferred to categorical classification while Sigmoid is preferred to Binary classification.

LSTM and CNN were used

LSTM stands for Long Short-Term Memory, and it is a type of recurrent neural network (RNN) architecture that is designed to capture and model long-term dependencies in sequential data. It is particularly effective in tasks such as natural language processing, speech recognition, and time series analysis.

The key idea behind LSTM is to address the vanishing gradient problem that occurs in traditional RNNs. The vanishing gradient problem refers to the issue where the gradients that are backpropagated through the network during training diminish exponentially, making it difficult for the network to learn long-term dependencies.

LSTM solves this problem by introducing a memory cell, which is a more complex unit compared to the simple recurrent unit used in traditional RNNs. The memory cell allows LSTM to selectively remember or forget information over time, enabling it to capture long-term dependencies more effectively.

The LSTM cell consists of several components:

1. **Cell State (Ct):** This represents the long-term memory of the cell. It runs straight down the entire chain of LSTM cells, and information can be added or removed from it through gates.
2. **Input Gate (i):** This gate determines how much of the new input should be stored in the cell state. It takes the current input and the previous hidden state as input and produces a value between 0 and 1 for each element in the cell state.
3. **Forget Gate (f):** This gate determines how much of the previous cell state should be forgotten or erased. It takes the current input and the previous hidden state as input and outputs a value between 0 and 1 for each element in the cell state.

4. **Output Gate (o):** This gate determines how much of the cell state should be outputted as the hidden state. It takes the current input and the previous hidden state as input and produces a value between 0 and 1 for each element in the cell state.
5. **Hidden State (h):** This represents the output of the LSTM cell, which is based on the cell state and the output gate. It is the LSTM's way of selectively revealing information from the cell state to the rest of the network.

The computations in an LSTM cell happen in three main steps:

Step 1: The input gate (i) and the forget gate (f) are computed using the current input and the previous hidden state. These gates determine how much information to store and forget in the cell state.

Step 2: The new cell state (Ct) is computed by combining the information from the input gate (i) and the previous cell state (Ct-1), as well as the new input. The forget gate (f) determines which parts of the previous cell state to erase.

Step 3: The output gate (o) is computed based on the current input and the previous hidden state. The output gate determines how much of the cell state should be revealed as the hidden state (h). The hidden state is then passed to the next LSTM cell or used as the final output of the network.

By selectively updating and revealing information over time, LSTM can effectively capture long-term dependencies in sequential data. The use of gates allows the network to adaptively learn which information to remember, forget, or output, making it well-suited for tasks that require modeling complex sequential patterns.

LSTM has proven to be a powerful architecture in various applications and has greatly contributed to advancements in natural language processing, speech recognition, machine translation, and many other fields.

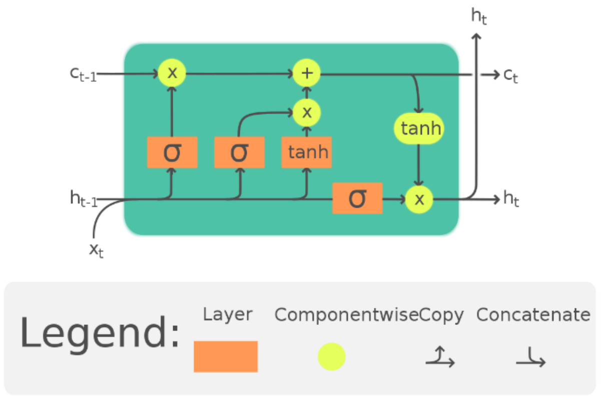


Figure 7 LSTM backpropagation formulas source: https://en.wikipedia.org/wiki/Long_short-term_memory

$$f_t = \sigma_g\left(W_f \times x_t + U_f \times h_{t-1} + b_f\right)$$
$$i_t = \sigma_g(W_i \times x_t + U_i \times h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o \times x_t + U_o \times h_{t-1} + b_o)$$
$$c'_t = \sigma_c(W_c \times x_t + U_c \times h_{t-1} + b_c)$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$
$$h_t = o_t \cdot \sigma_c(c_t)$$

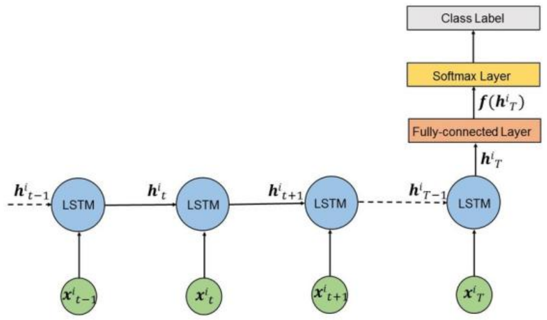


Figure 8 LSTM layer with single output source:https://www.researchgate.net/figure/Sequence-to-Label-classification-using-LSTM-NN_fig2_336899835

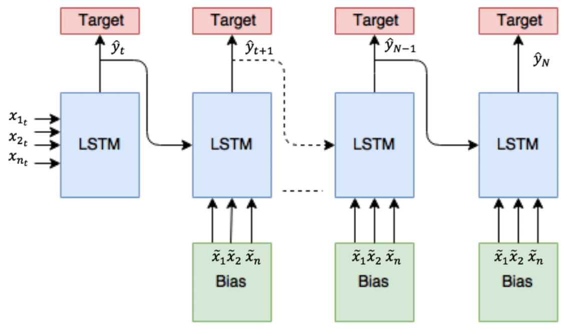


Figure 9 LSTM With multiple outputs source: https://www.researchgate.net/figure/Multivariate-LSTM-with-4-features-and-a-single-output-The-output-of-LSTM-at-time-t-is_fig5_324600237

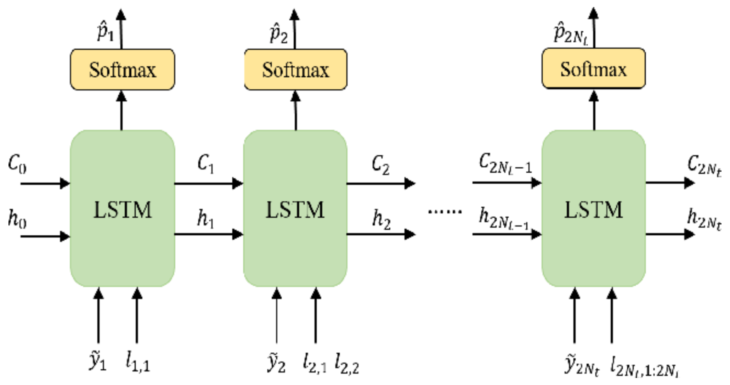


Figure 10 MIMO LSTM source: https://www.semanticscholar.org/paper/Learning-to-Search-for-MIMO-Detection-Sun-Zhang/

A one-dimensional Convolutional Neural Network (1D CNN) is a deep learning model specifically designed to process one-dimensional sequential data, such as time series or text data. It utilizes convolutional operations to extract relevant features from the input sequence.

Here's an overview of how a 1D CNN works:

1. Input Representation: The input to a 1D CNN is a one-dimensional sequence, typically represented as a vector. For example, in text data, each word may be represented as a vector or an index. In time series data, each element represents a point in time.
2. Convolutional Layers: The core component of a 1D CNN is the convolutional layer. It applies a set of filters to the input sequence, sliding them across the sequence to compute a dot product between the filter weights and the corresponding input values at each position. The dot product result is then passed through a non-linear activation function, such as ReLU (Rectified Linear Unit).

The filters capture local patterns or features present in the input sequence, enabling the network to learn and recognize important characteristics at different positions. The number of filters determines the number of output channels or feature maps generated by the convolutional layer.

3. Pooling Layers: Pooling layers are often used after convolutional layers to reduce the spatial dimensions of the feature maps and control the model's complexity. Max pooling is a common pooling technique where the maximum value within a fixed window is selected and retained, while discarding the other values. This downsamples the feature maps, preserving the most salient features and reducing the dimensionality.
4. Fully Connected Layers: Following the convolutional and pooling layers, one or more fully connected layers are typically employed. These layers connect every neuron in the previous layer to the subsequent layer. The fully connected layers capture global patterns and relationships among the learned features.
5. Output Layer: The final layer of the 1D CNN is the output layer, which depends on the specific task at hand. For example, in a classification problem, the output layer may employ a softmax activation function to produce class probabilities. In regression tasks, it may use a linear activation function to generate continuous values.
6. Training: During the training process, the 1D CNN learns the optimal filter weights and biases through backpropagation and gradient descent. It compares the predicted output with the true output labels using a loss function, such as categorical cross-entropy or mean squared error. The gradients are then propagated backward through the network to update the weights, iteratively improving the model's performance.

1D CNNs have been successful in a variety of applications, including natural language processing, speech recognition, time series analysis, and audio processing. They excel at capturing local dependencies and patterns in sequential data, enabling effective feature extraction and representation learning.

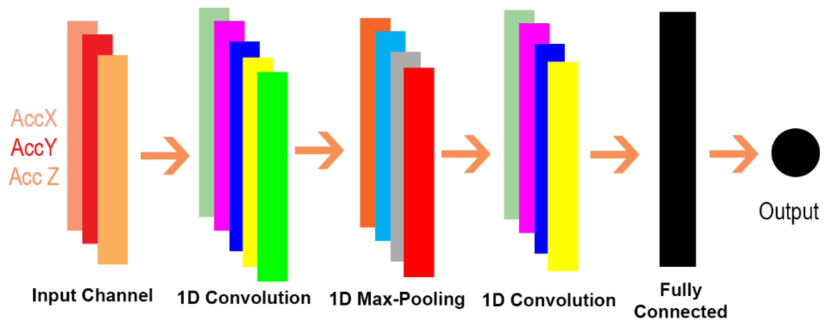


Figure 11 1D convolution to single output

Testing:

Cross-validation was used in each model with 5-folds each\

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5, shuffle=False)
```

Validation accuracy was calculated from 10% of the training data

```
x_train, x_val, y_train, y_val = train_test_split(
     x[train],
     y[train],
     test_size=0.10)
```

Work frame:

Keras was used on python for building those models

Example:

```
model = Sequential([
        LSTM(50, kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.L2(1e-4), input_shape=(256,3), return_sequences = True ),
        Dropout(0.2),
        LSTM(50, kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.L2(1e-4), input_shape=(256,3), return_sequences = False ),
        Dropout(0.2),
        Dense(2, activation = 'softmax')
    ])
```

Callbacks:

Learning rate scheduler used in most models (the models that don't use it will be noted):

```
def decayed_learning_rate(step):
        return initial_learning_rate * decay_rate ^ (step / decay_steps)
```

This was built in by Keras in the following function:

```
tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate =  0.001,
        decay_steps=205,
        decay_rate=0.96,
        staircase=True)
```

Model checkpoint was used to save the best model based on the validation accuracy

```
tf.keras.callbacks.ModelCheckpoint(
    filepath=f'best_{k}.hdf5',
    verbose=0, save_best_only=True,
    monitor='val_accuracy',
    mode='max')
```

K refers to the current fold

Early stopping was set to patience 25

```
tf.keras.callbacks.EarlyStopping(
    monitor='accuracy',
    patience=25)
```

Monitoring the accuracy was chosen because the training accuracy stagnates without affecting the highest validation accuracy, as well: saving the best model based on validation accuracy is used

Hyperbolic Tangent $tanh$ activation was used as it is the default function for GPU cudnn usage

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

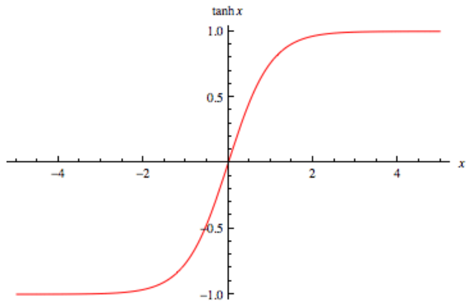


Figure 12 tanh function source: https://mathworld.wolfram.com/HyperbolicTangent.html

Optimizer:

Adam optimizer was used

$$v_t = \beta_1 * v_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{v_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$
$$\eta\text{: Initial Learning rate}$$
$$g_t\text{: Gradient at time t along } \omega^j$$
$$v_t\text{: Exponential Average of gradients along } \omega_j$$
$$s_t\text{: Exponential Average of squares of gradients along } \omega_j$$
$$\beta_1, \beta_2 \text{ : Hyperparameters}$$

```
tf.keras.optimizers.Adam(learning_rate = rlr)
```

Normalization:

Standard deviation normalization was used

$$z = \frac{(x-\mu)}{\sigma}$$

$$z: normalized\ data\ point$$

$$x: data\ point$$

$$\mu: data\ mean$$

$$\sigma: standard\ deviation$$

Clipping value:

Clipping value was used in some models

The clip value refers to a technique used to control the exploding gradients problem during the training process. Exploding gradients occur when the gradients in the network become extremely large, leading to unstable and ineffective learning.

During the backpropagation process, the gradients are computed and used to update the network's parameters. However, in deep networks, especially those with many recurrent connections like LSTMs, the gradients can accumulate and become very large or very small. This issue is known as the vanishing or exploding gradients problem.

To mitigate the exploding gradients problem, one common approach is to apply gradient clipping. Gradient clipping restricts the gradients to a predefined range, effectively limiting their magnitude. By doing so, the gradients are prevented from growing too large, which helps stabilize the learning process.

In the context of LSTM networks, gradient clipping can be applied to the gradients that flow through the network during backpropagation. The clip value is a threshold that determines the maximum allowed value for the gradients. If the norm of the gradients exceeds this threshold, they are rescaled to ensure they stay within the specified range.

More specifically, given a clip value **C**, the gradient vector **g** is scaled down if its L2 norm exceeds **C**. The rescaling is performed by calculating the ratio **C / ||g||**, where **||g||** represents the L2 norm of the gradient vector. If the norm is smaller than or equal to the clip value, the gradients remain unchanged.

The gradient clipping operation can be expressed mathematically as:

```
if ||g|| > C:
    g = (C / ||g||) * g
```

By applying gradient clipping in LSTM networks, the gradients are effectively limited to a specific range, preventing them from becoming too large. This ensures more stable and reliable training, allowing the network to learn effectively from the given data.

It's worth noting that gradient clipping is not specific to LSTMs and can be applied to other types of neural networks as well. The specific value chosen for the clip value is typically determined through experimentation and can vary depending on the network architecture and the nature of the problem being solved.

Solutions tree

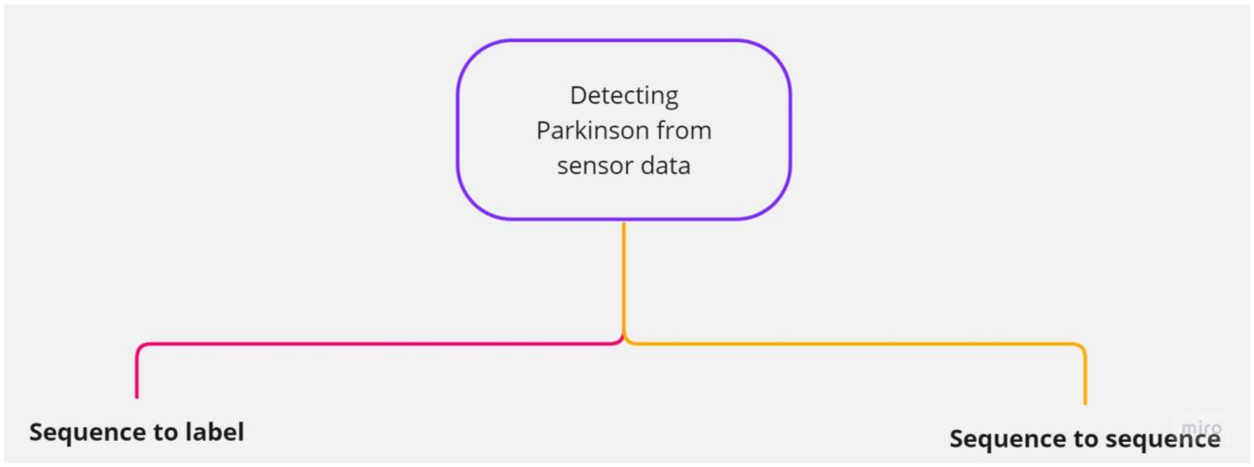


Figure 13 model map head:

Sequence to label models with 2 seconds of input:

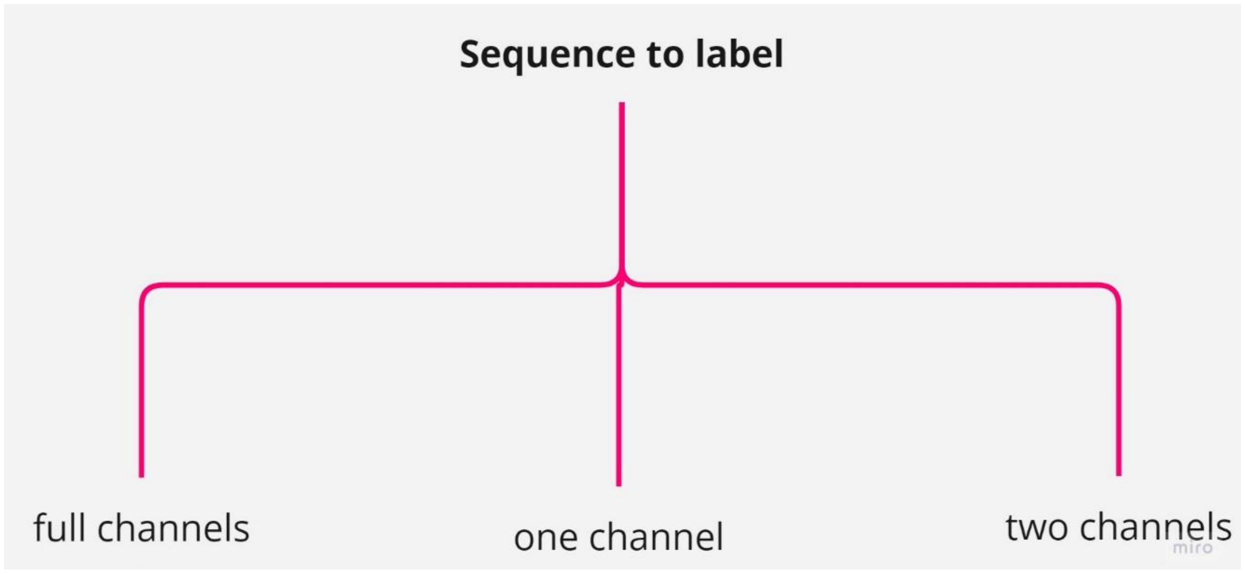


Figure 14 Sequence to label tree

Full channels

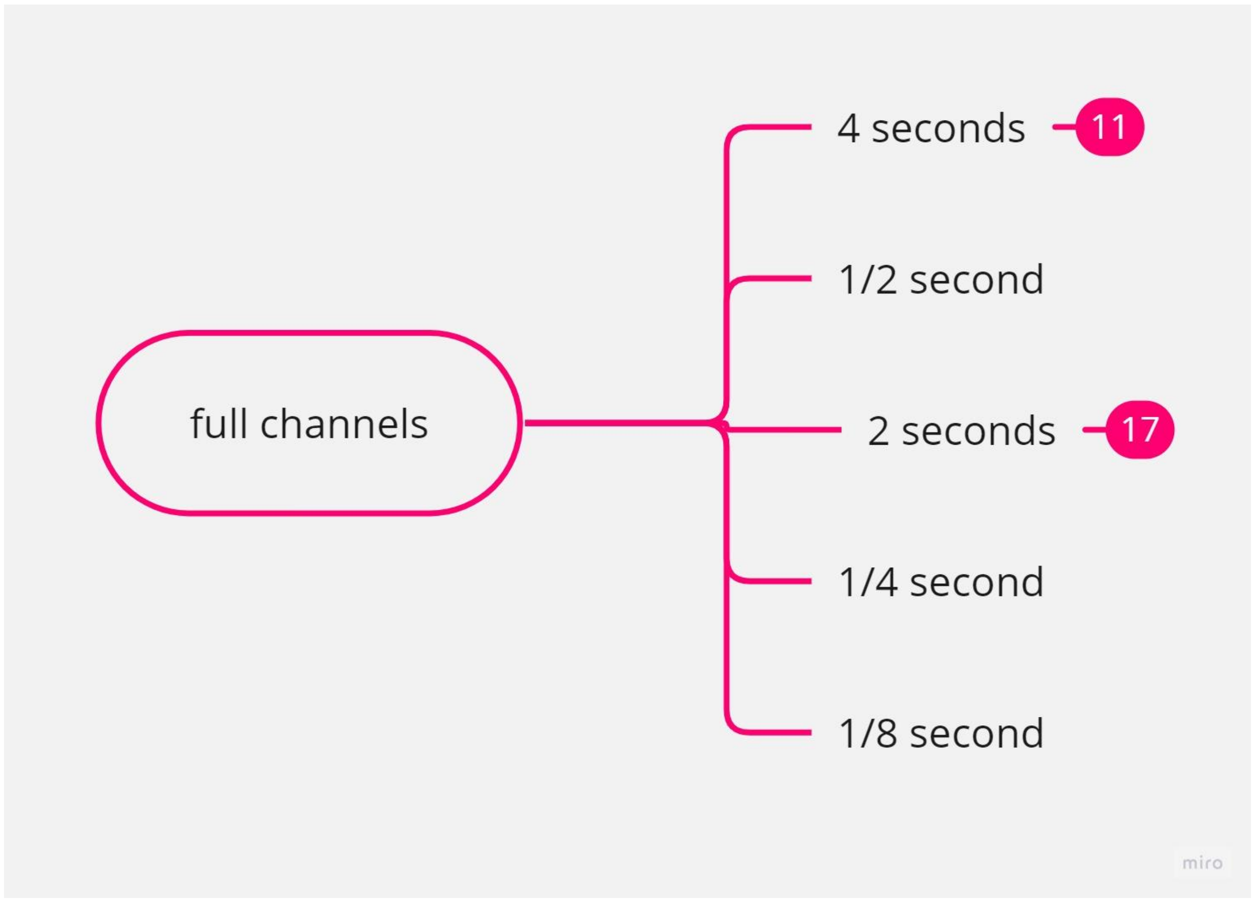


Figure 15 Number of seconds for full channels

Sequence to Sequence Models for 2 seconds:

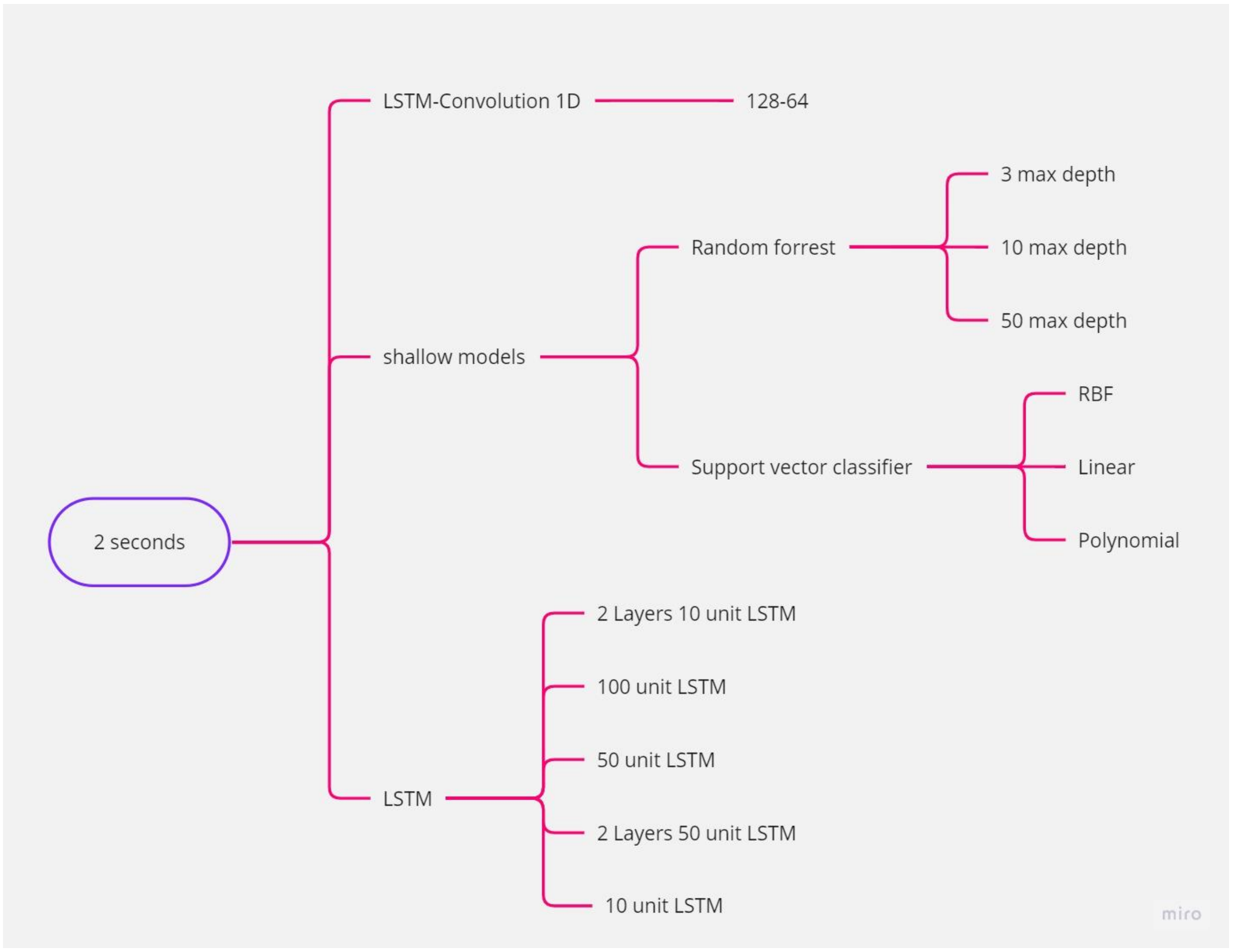


Figure 16 2 seconds intended models

Model 1:

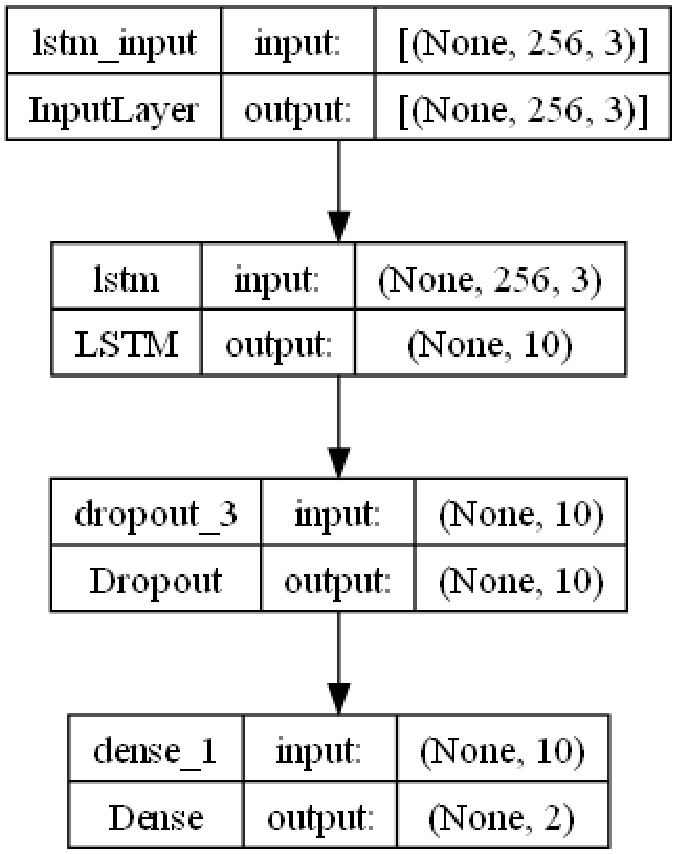
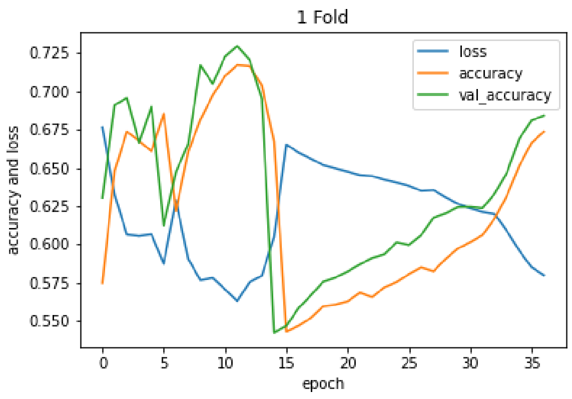


Figure 17 Model 1 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.717	0.7295	0.7494	11.0	0.7496	0.7493	0.7495
Fold 2	0.7511	0.7703	0.7676	12.0	0.7678	0.7672	0.7675
Fold 3	0.7542	0.7617	0.7721	25.0	0.7724	0.772	0.7722
Fold 4	0.8306	0.8305	0.793	104.0	0.7932	0.7928	0.793
Fold 5	0.8394	0.845	**0.8154**	117.0	0.8156	0.8152	0.8154
Average	0.77846	0.7874	0.7795	53.8	0.77972	0.7793	0.77952

Table 3 metrics for model 1



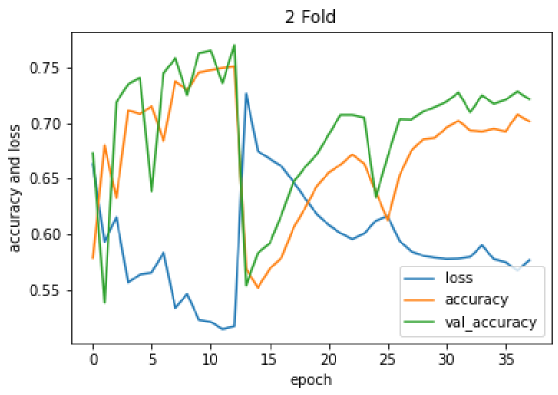
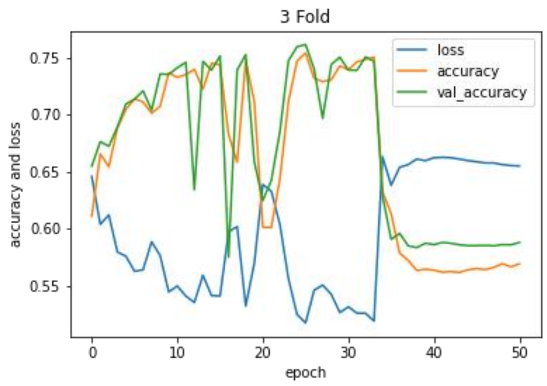


Figure 18 1st and 2nd Folds



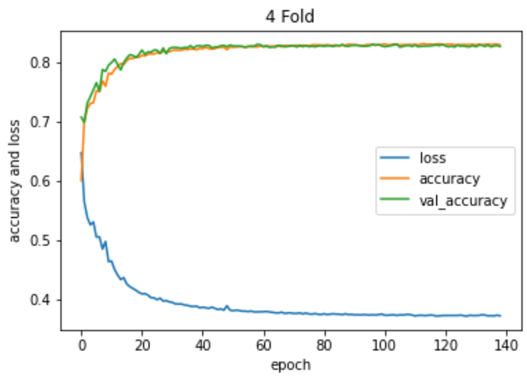


Figure 19 3rd and 4th Folds

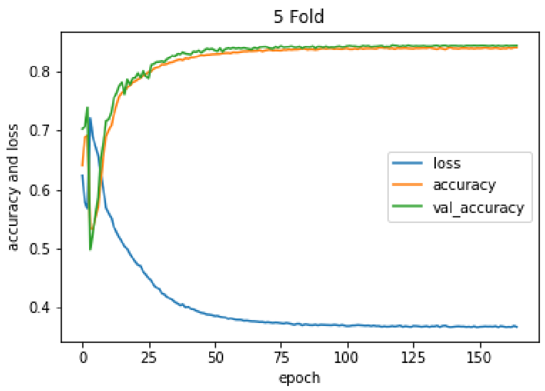


Figure 20 5th Fold

Model 2**:**

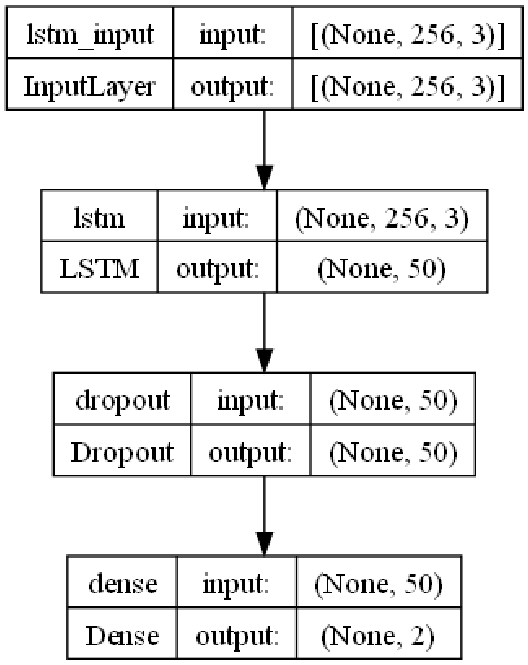


Figure 21 Model 2 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.8609	0.8683	0.8632	96.0	0.8634	0.8631	0.8633
Fold 2	0.8777	0.8853	0.8521	73.0	0.8524	0.8518	0.8521
Fold 3	0.8462	0.8289	0.8339	168.0	0.8343	0.8338	0.8341
Fold 4	0.7262	0.7416	0.7032	1.0	0.7035	0.7032	0.7033
Fold 5	0.8782	0.8798	**0.8555**	134.0	0.8558	0.8554	0.8556
Average	0.83783	0.84078	0.82158	94.4	0.8219	0.8215	0.8217

Table 4 Metrics for Model 2

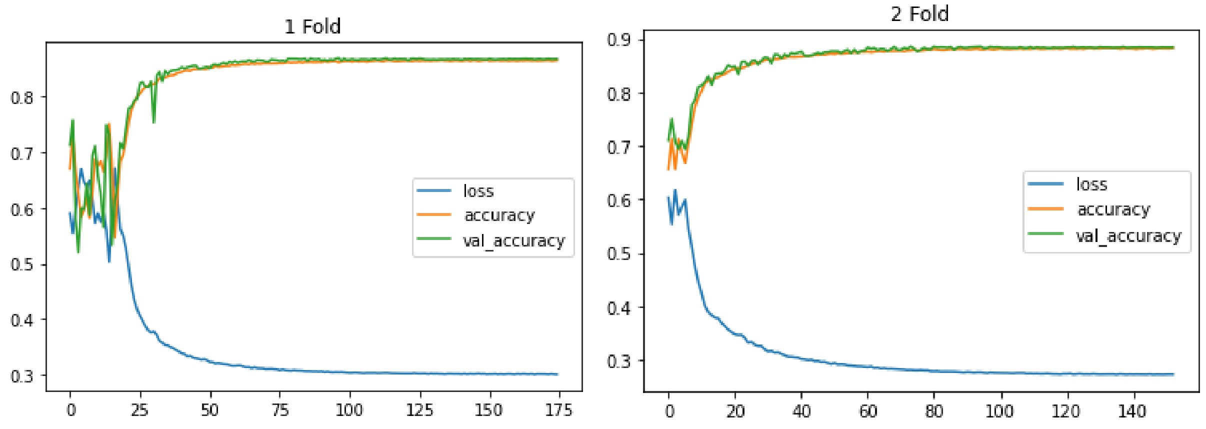


Figure 22 1st and 2nd Folds

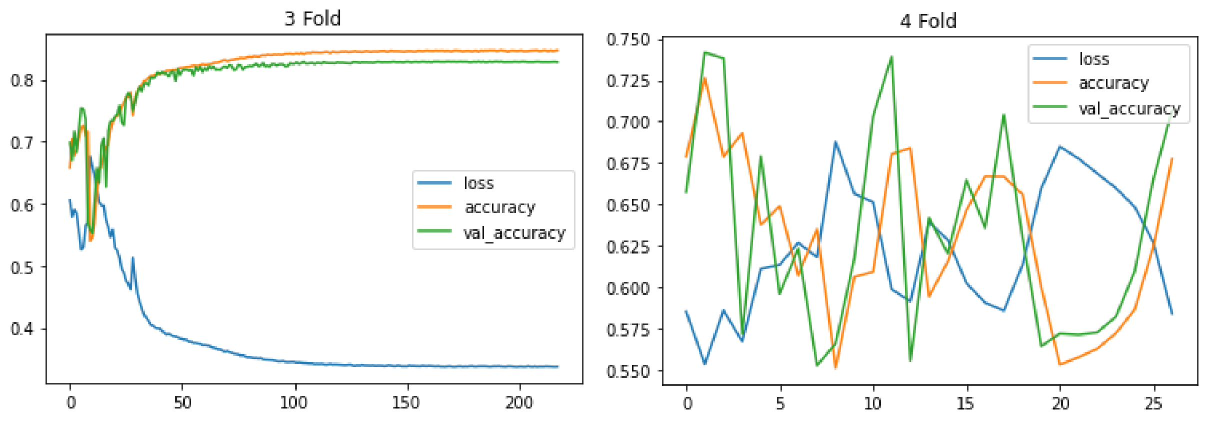


Figure 23 2nd and 3rd Folds

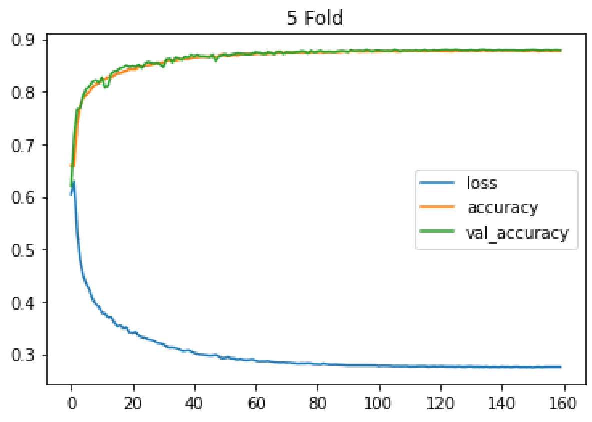


Figure 24 5th Fold

Model 3:

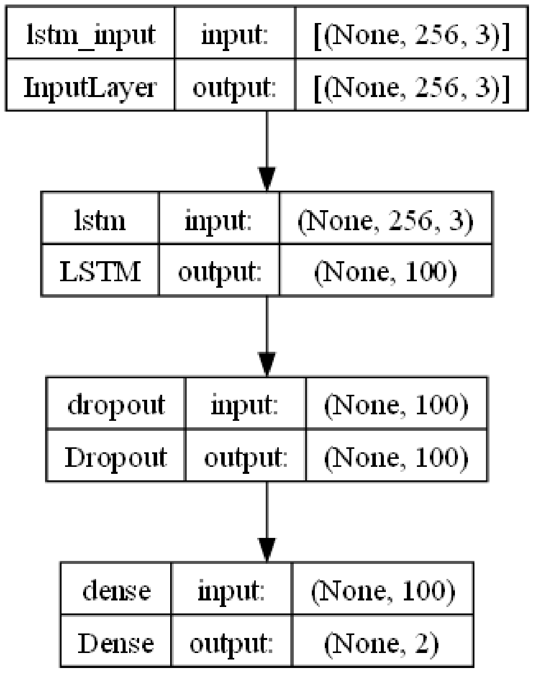
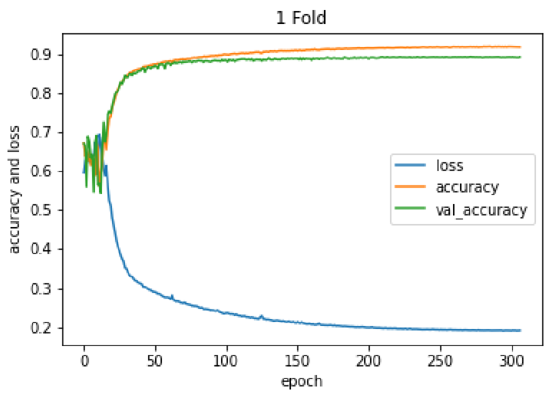


Figure 25 Model 3 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.9182	0.8923	0.87	272.0	0.8703	0.87	0.8701
Fold 2	0.9073	0.8888	**0.8552**	213.0	0.8553	0.8547	0.855
Fold 3	0.9661	0.9209	0.8433	302.0	0.8436	0.8432	0.8434
Fold 4	0.9681	0.9206	0.8486	323.0	0.8489	0.8484	0.8487
Fold 5	0.6858	0.7266	0.725	2.0	0.7251	0.7248	0.7249
Average	0.8891	0.8698	0.8284	222.4	0.8286	0.8282	0.8284

Table 5 Metrics for Model 3



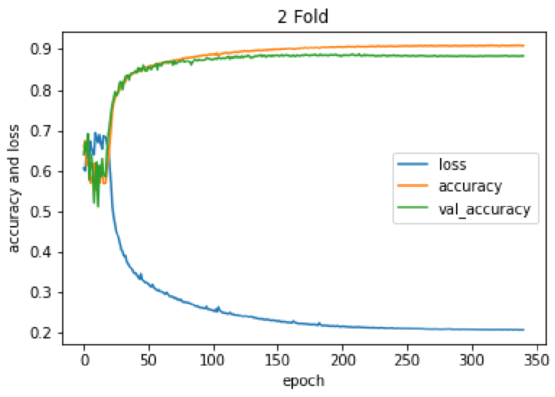
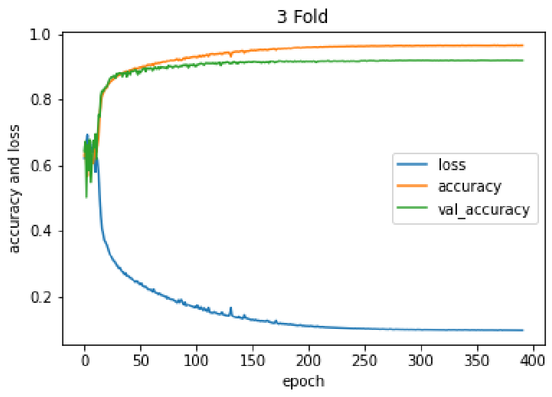


Figure 26 1st and 2nd Fold for model 3



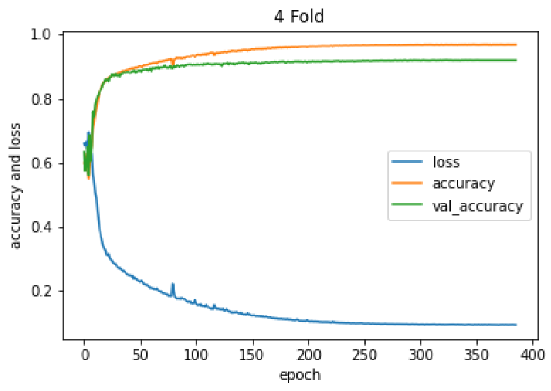


Figure 27 3rd and 4th Fold for Model 3

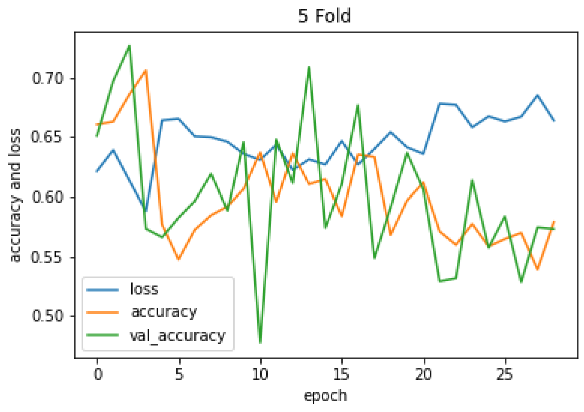


Figure 28 5th Fold for Model 3

Model 4:

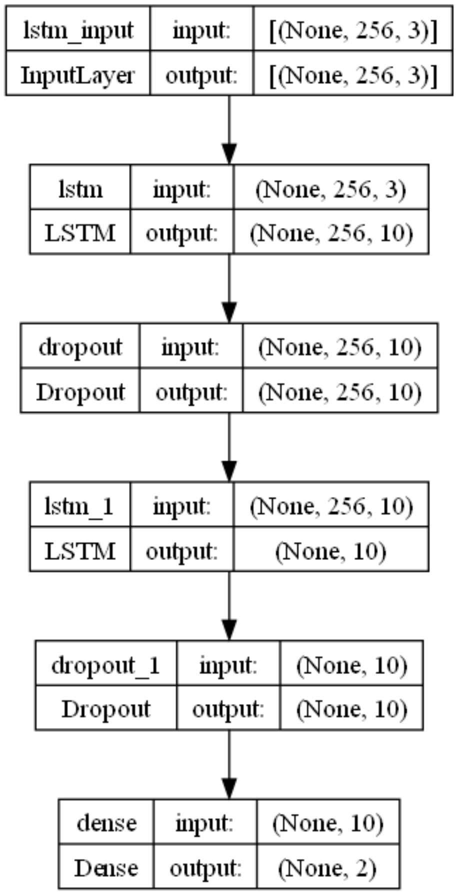
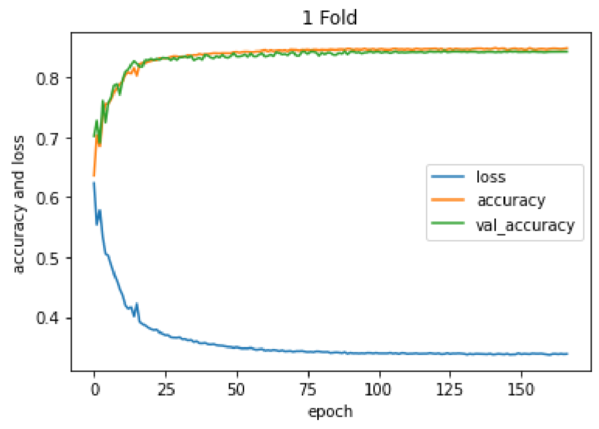

lstm_input | input: | [(None, 256, 3)]
InputLayer | output: | [(None, 256, 3)]
lstm | input: | (None, 256, 3)
LSTM | output: | (None, 256, 10)
dropout | input: | (None, 256, 10)
Dropout | output: | (None, 256, 10)
lstm_1 | input: | (None, 256, 10)
LSTM | output: | (None, 10)
dropout_1 | input: | (None, 10)
Dropout | output: | (None, 10)
dense | input: | (None, 10)
Dense | output: | (None, 2)

Figure 29 Model 4 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.8455	0.8445	0**.8437**	123	0.8439	0.8436	0.8438
Fold 2	0.7965	0.8054	0.8034	27	0.8036	0.8031	0.8033
Fold 3	0.7809	0.7786	0.7878	102	0.7881	0.7877	0.7879
Fold 4	0.8468	0.8497	0.8242	69	0.8245	0.8241	0.8243
Fold 5	0.7648	0.7682	0.7536	63	0.7540	0.7536	0.7538
Average	0.8069	0.8093	0.8026	76	0.8028	0.8024	0.8026

Table 6 Metrics for Model 4



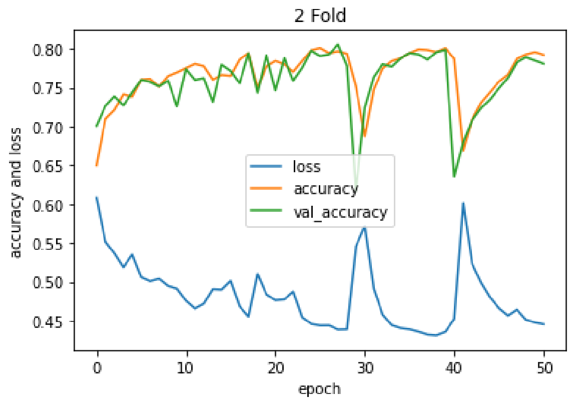
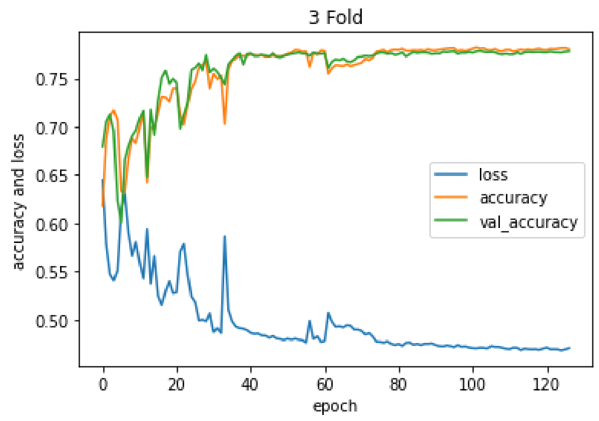


Figure 30 1st and 2nd Folds for model 4



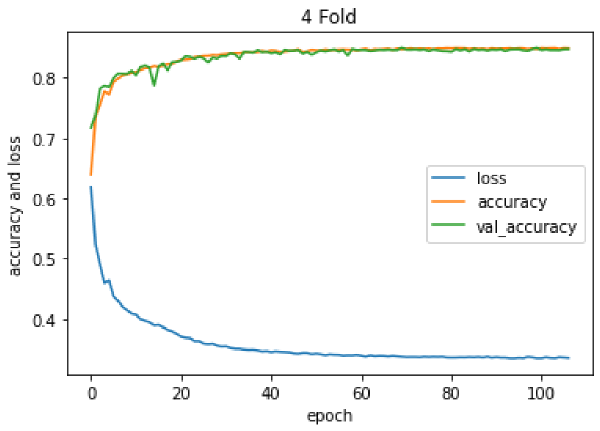


Figure 31 3rd and 4th Folds for model 4

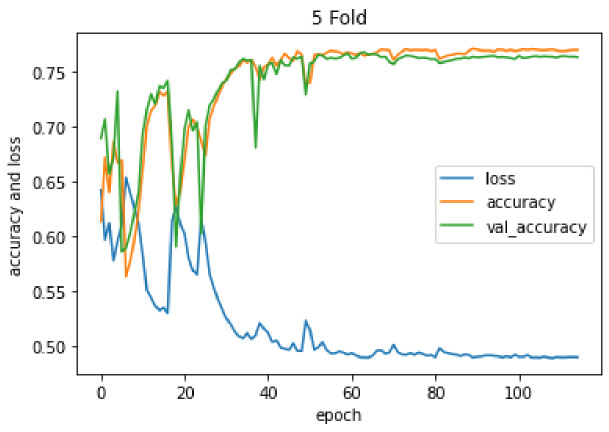


Figure 32 5th Fold for model 4

Model 5:

Cyclic learning rate used

$$scale(x) = \frac{1}{1.2^{x-1}}$$

Maximum learning rate was set to 10^-3 and minimum learning rate was set to 10^-5

Using tensor flow implementation

```
tfa.optimizers.CyclicalLearningRate(
      initial_learning_rate=1e-4,
      maximal_learning_rate=1e-2,
      scale_fn= scale ,
      step_size= 205,
    )
```

Early stopping patience was set to 15 as the model stagnates for a long time

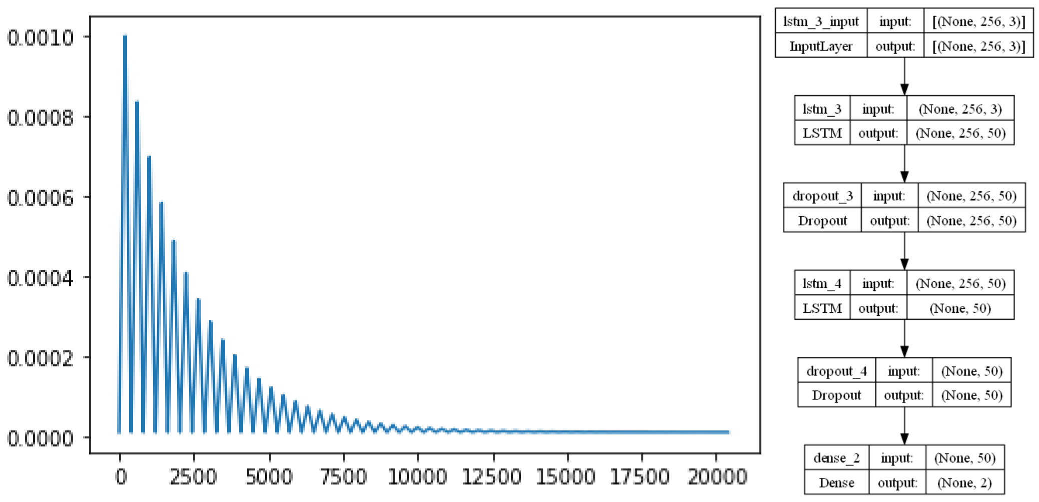


Figure 33 Cyclic learning rate used function and the model architecture

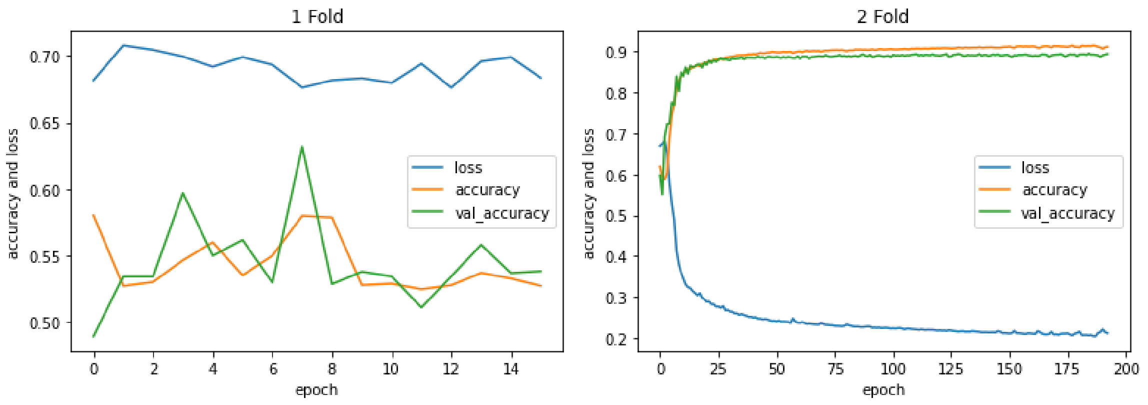


Figure 34 Model 5 1st and 2nd Folds

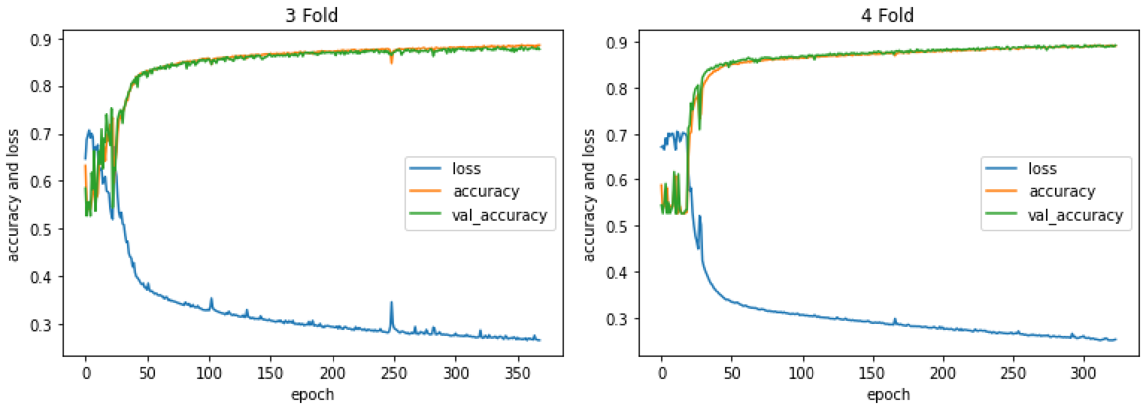


Figure 35 Model 5 3rd and 4th Folds

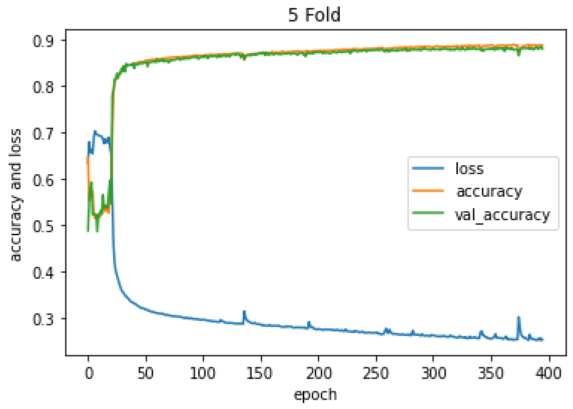


Figure 36 Model 4 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.5798	0.6317	0.5619	7.0	0.562	0.5618	0.5619
Fold 2	0.9133	0.895	0.8525	184.0	0.8528	0.8522	0.8525
Fold 3	0.8858	0.8825	0.8583	353.0	0.8586	0.8582	0.8584
Fold 4	0.8893	0.8929	**0.8565**	301.0	0.8567	0.8563	0.8565
Fold 5	0.8892	0.8851	0.848	394.0	0.8483	0.8479	0.8481
Average	0.8315	0.8375	0.7955	247.8	0.7957	0.7953	0.7955

Figure 37 Model 5 metrics

Model 6:

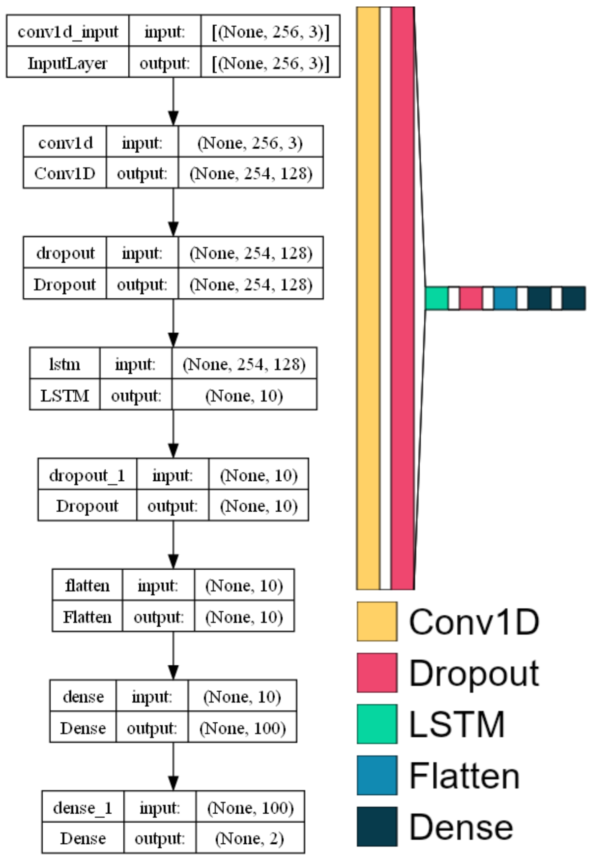
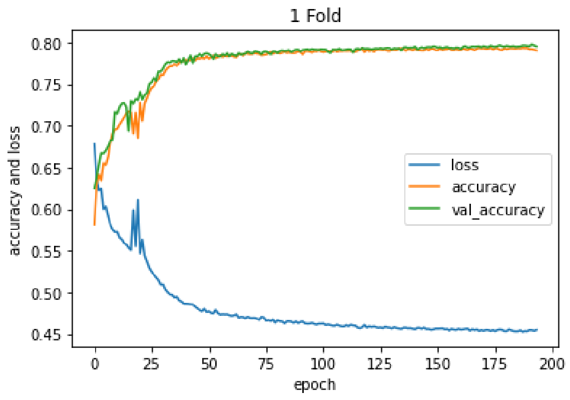

conv1d_input | input: | [(None, 256, 3)]
InputLayer | output: | [(None, 256, 3)]
conv1d | input: | (None, 256, 3)
Conv1D | output: | (None, 254, 128)
dropout | input: | (None, 254, 128)
Dropout | output: | (None, 254, 128)
lstm | input: | (None, 254, 128)
LSTM | output: | (None, 10)
dropout_1 | input: | (None, 10)
Dropout | output: | (None, 10)
flatten | input: | (None, 10)
Flatten | output: | (None, 10)
dense | input: | (None, 10)
Dense | output: | (None, 100)
dense_1 | input: | (None, 100)
Dense | output: | (None, 2)
Conv1D
Dropout
LSTM
Flatten
Dense

Figure 38 Model 6 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.792	0.7978	**0.8185**	191.0	0.8186	0.8184	0.8185
Fold 2	0.8025	0.8094	0.8109	99.0	0.811	0.8104	0.8107
Fold 3	0.7906	0.8054	0.8114	141.0	0.8117	0.8113	0.8115
Fold 4	0.7977	0.7995	0.7641	113.0	0.7643	0.764	0.7642
Fold 5	0.8022	0.8244	0.7756	108.0	0.7759	0.7755	0.7757
Average	0.797	0.8073	0.7961	130.4	0.7963	0.7959	0.7961

Table 7 Metrics for Model 6



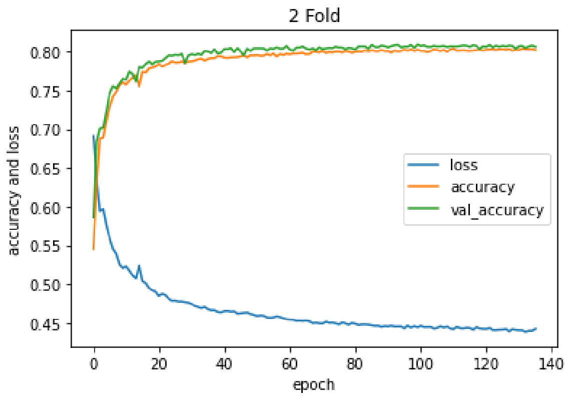
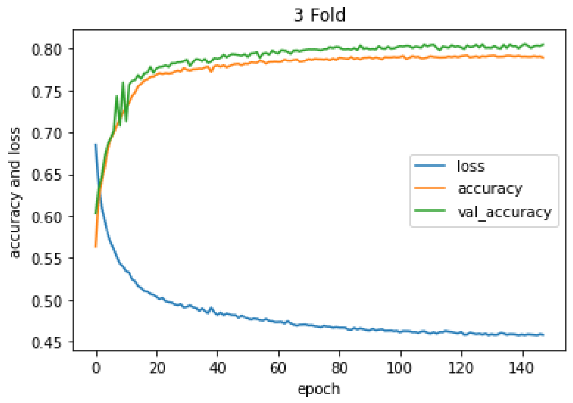


Figure 39 Model 6 1st and 2nd Folds



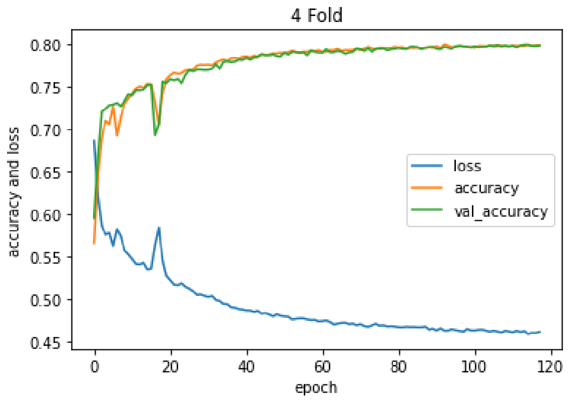


Figure 40 Model 6 3rd and 4th Folds

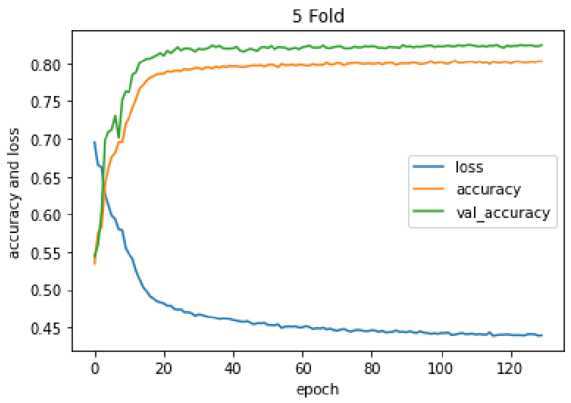


Figure 41 Model 6 5th Fold

Shallow models:

2 seconds engineered features: (*md stands for max depth*)

	Training accuracy	Test accuracy
SVC radial bases function kernel	0.7374	0.7343
SVC Polynomial kernel	0.6994	0.6948
SVC linear kernel	0.6547	0.6438
Random Forest 3 md	0.6979	0.6908
Random Forest 10 md	0.7828	**0.7511**
Random Forest 50 md	1	0.7488

Table 8 Metrics for SVC and Random Forest for 2 seconds engineered features

4 seconds engineered features: (*md stands for max depth*)

	Training accuracy	Test accuracy
SVC radial bases function kernel	0.7203	0.7328
SVC polynomial kernel	0.6757	0.6850
SVC linear kernel	0.6187	0.6364
Random Forest 3 md	0.6718	0.6734
Random Forest 4 md	0.6806	0.6777
Random Forest 10 md	0.7922	**0.7430**
Random Forest 50 md	0.9999	0.7383

Table 9 Metrics for SVC and Random Forest for 4 seconds engineered features

Models For different intervals of time:

To answer the question of what is the smallest interval I can get to until the model accuracy goes down, a standard 2 layer 10 unit LSTM was trained for each sequence length, then comparing the accuracy of the different sequences, the drop in accuracy starts to show, until it’s better to just use dense layers.

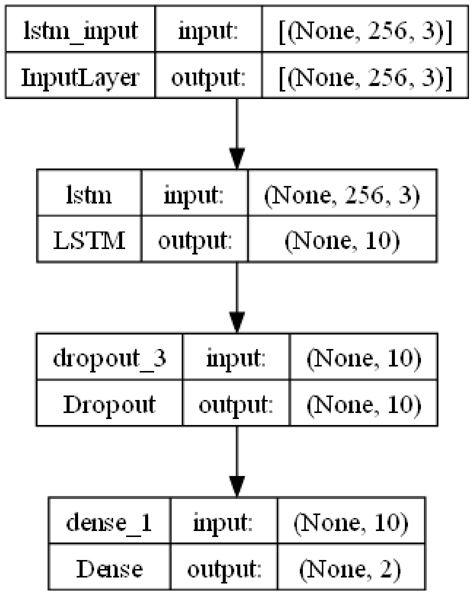


Figure 42 Model used in different time intervals

Half a second results:

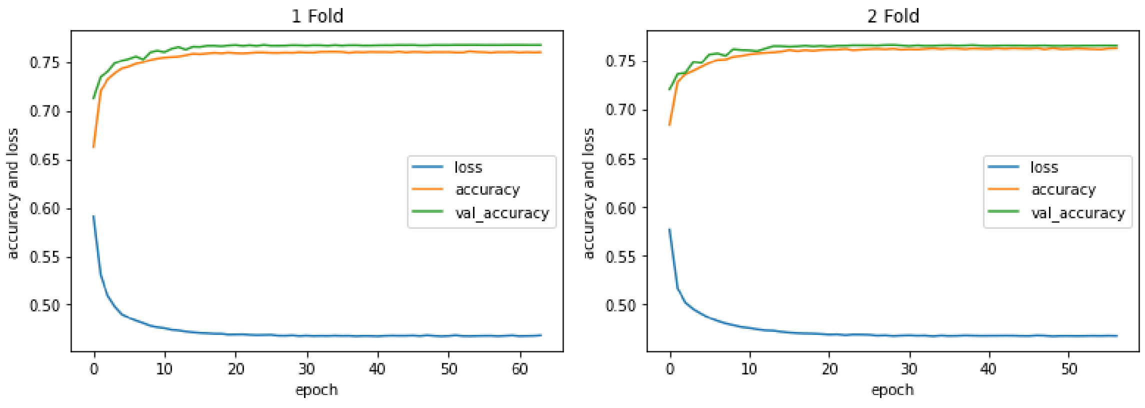


Figure 43 1st and 2nd Folds

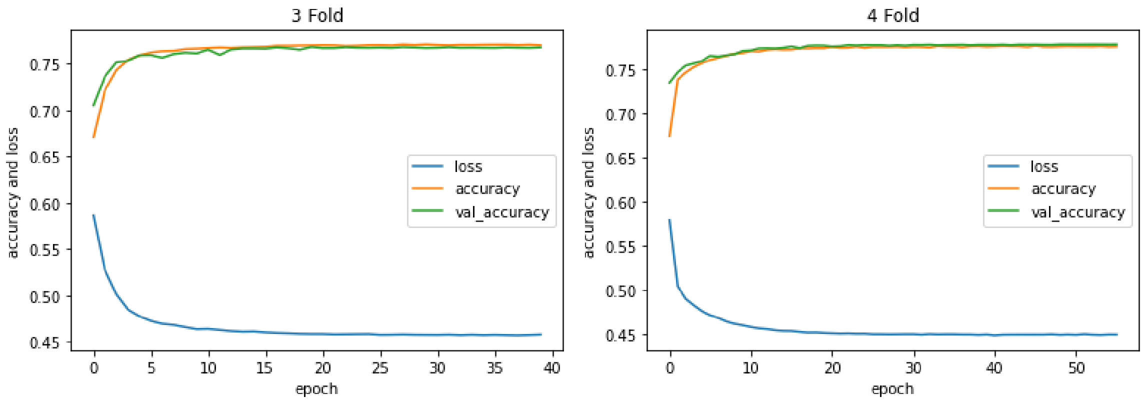


Figure 44 3rd and 4th folds

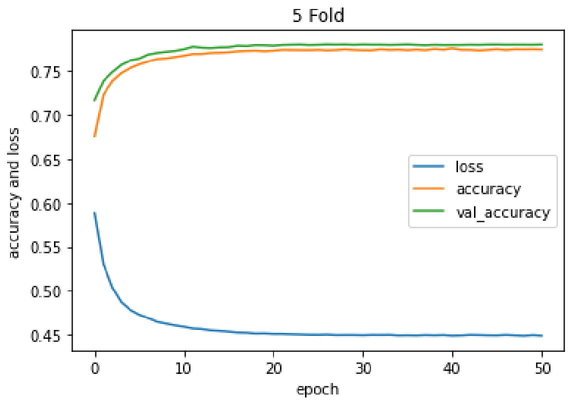
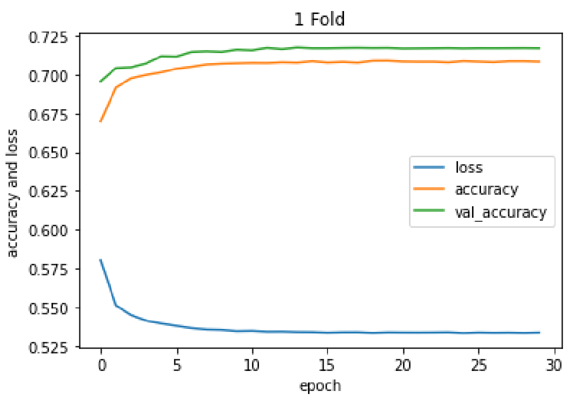


Figure 45 5th fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.9182	0.8923	0.87	272.0	0.8703	0.87	0.8701
Fold 2	0.9073	0.8888	**0.8552**	213.0	0.8553	0.8547	0.855
Fold 3	0.9661	0.9209	0.8433	302.0	0.8436	0.8432	0.8434
Fold 4	0.9681	0.9206	0.8486	323.0	0.8489	0.8484	0.8487
Fold 5	0.6858	0.7266	0.725	2.0	0.7251	0.7248	0.7249
Average	0.8891	0.8698	0.8284	222.4	0.8286	0.8282	0.8284

Table 10 Interval's metrics

One fourth of a second model:



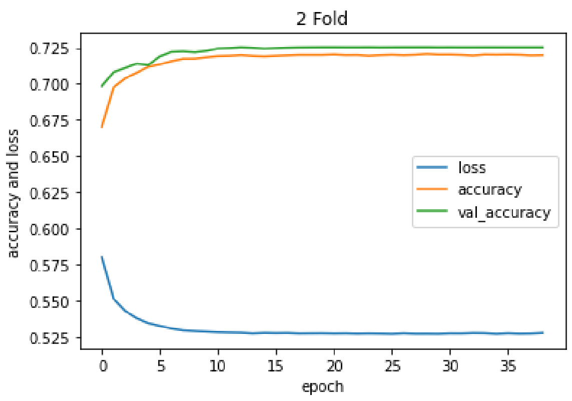
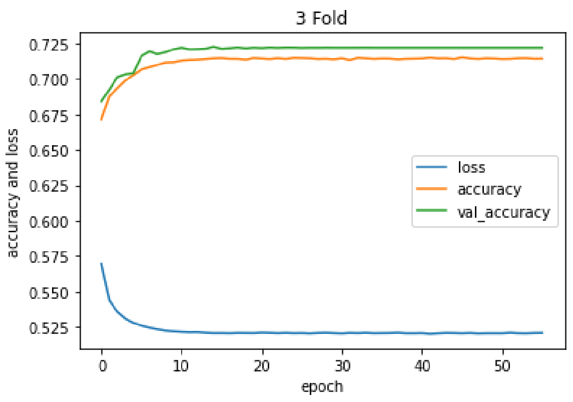


Figure 46 Interval's 1st and 2nd Folds



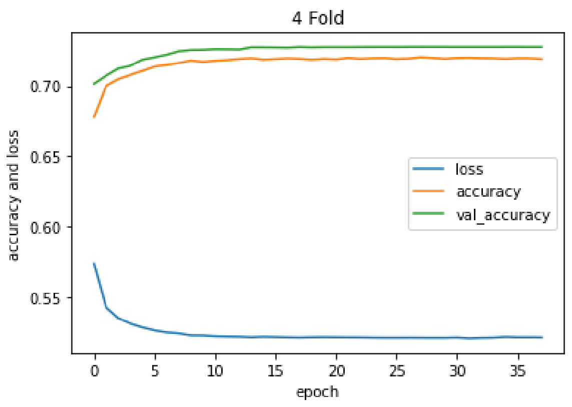


Figure 47 Interval's 3rd and 4th Folds

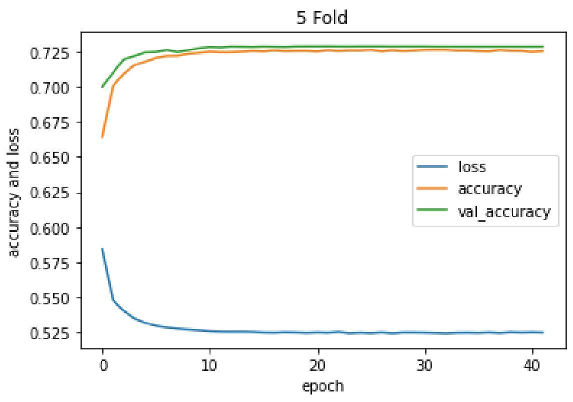


Figure 48 Interval's 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7076	0.7174	0.7078	13.0	0.7079	0.7078	0.7078
Fold 2	0.7206	0.7253	0.7094	28.0	0.7095	0.7093	0.7094
Fold 3	0.7145	0.7226	**0.7343**	14.0	0.7344	0.7342	0.7343
Fold 4	0.7201	0.7274	0.6792	27.0	0.6792	0.679	0.6791
Fold 5	0.7258	0.7286	0.704	21.0	0.704	0.7038	0.7039
Average	0.7177	0.7243	0.7069	20.6	0.707	0.7068	0.7069

Figure 49 Interval's metrics

One eighth of a second model:

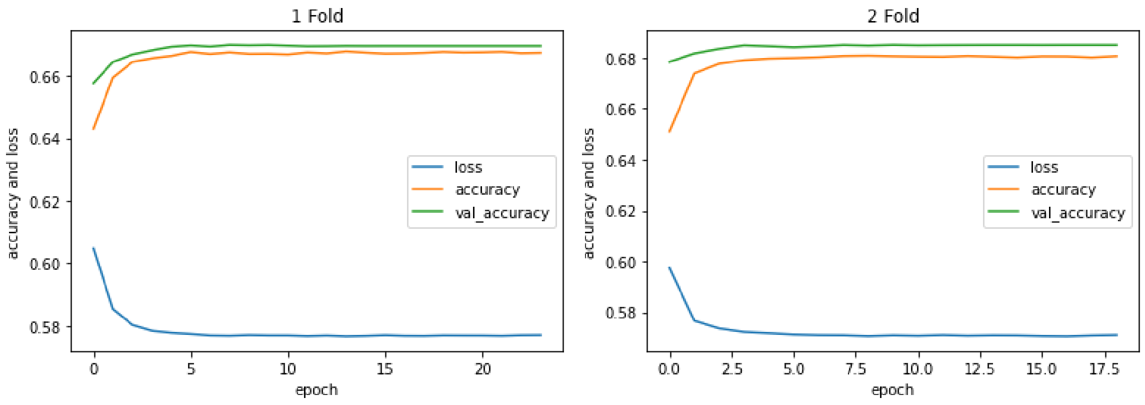


Figure 50 Figure 46 Interval's 1st and 2nd Folds

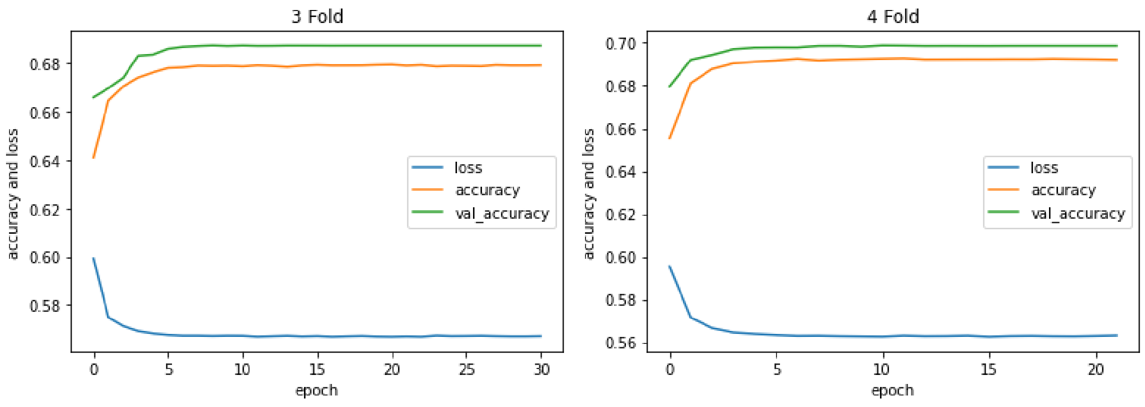


Figure 51 Figure 46 Interval's 3rd and 4th Folds

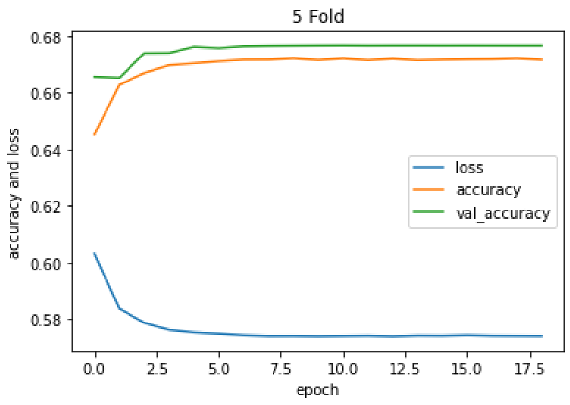


Figure 52 Figure 46 Interval's 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.6675	0.6698	0.6643	7.0	0.6643	0.6642	0.6642
Fold 2	0.6805	0.6849	0.6673	9.0	0.6673	0.6672	0.6672
Fold 3	0.6787	0.6871	**0.6767**	8.0	0.6768	0.6766	0.6767
Fold 4	0.6925	0.6987	0.6411	10.0	0.6411	0.641	0.6411
Fold 5	0.6721	0.6767	0.6672	10.0	0.6672	0.667	0.6671
Average	0.6783	0.6835	0.6633	8.8	0.6634	0.6632	0.6633

Figure 53 Interval's Metrics

Models for Different channels:

100 unit LSTM was used to determine what the best channel combination is. (Model_3)

One channel:

AccAP channel:

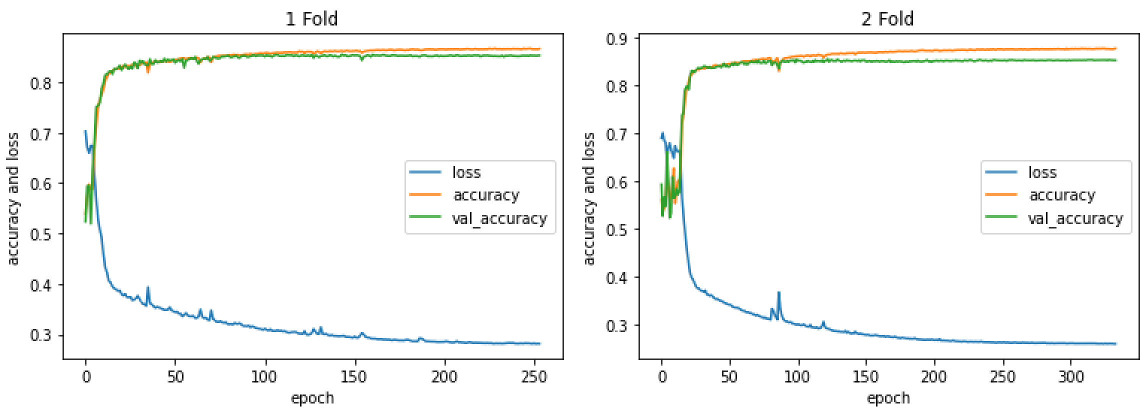


Figure 54 1st and 2nd Folds

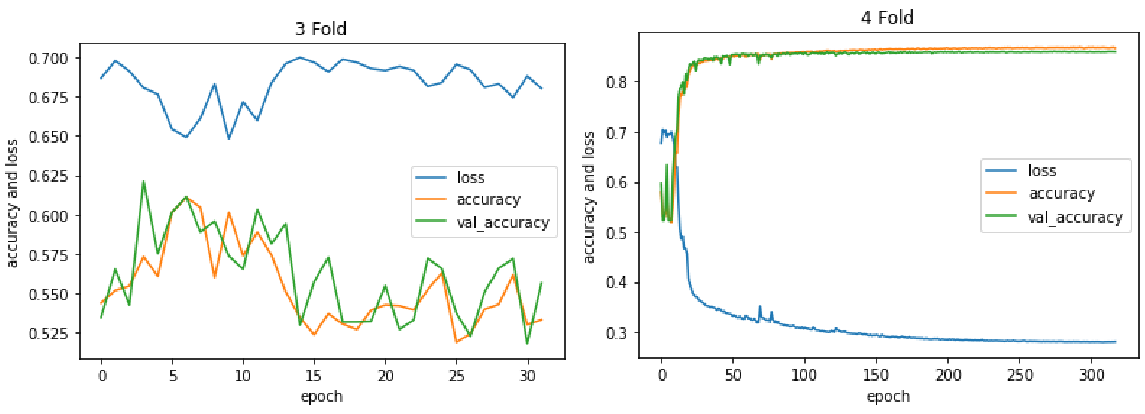


Figure 55 3rd and 4th Fold

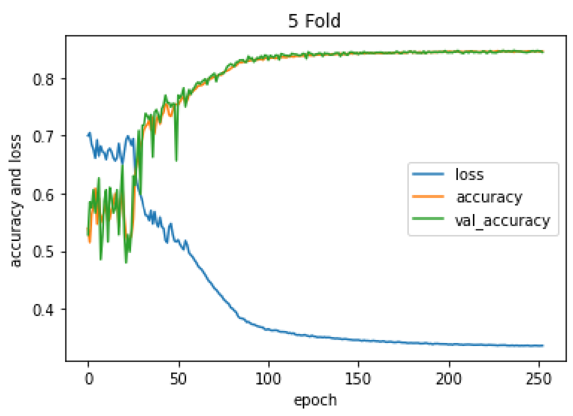


Figure 56 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.8629	0.8561	**0.8495**	143.0	0.8495	0.8492	0.8494
Fold 2	0.8653	0.8563	0.8349	122.0	0.8353	0.8347	0.835
Fold 3	0.5731	0.6211	0.6469	3.0	0.647	0.6467	0.6469
Fold 4	0.8677	0.8602	0.8438	274.0	0.8441	0.8437	0.8439
Fold 5	0.8445	0.8478	0.8176	236.0	0.8176	0.8172	0.8174
Average	0.8027	0.8083	0.7985	155.6	0.7987	0.7983	0.7985

Figure 57 Metrics

AccML channel:

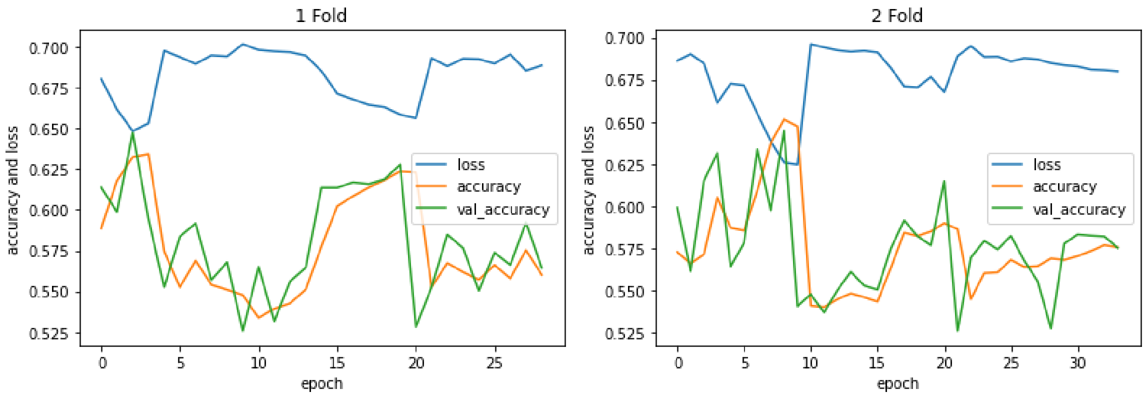


Figure 58 1st and 2nd Folds

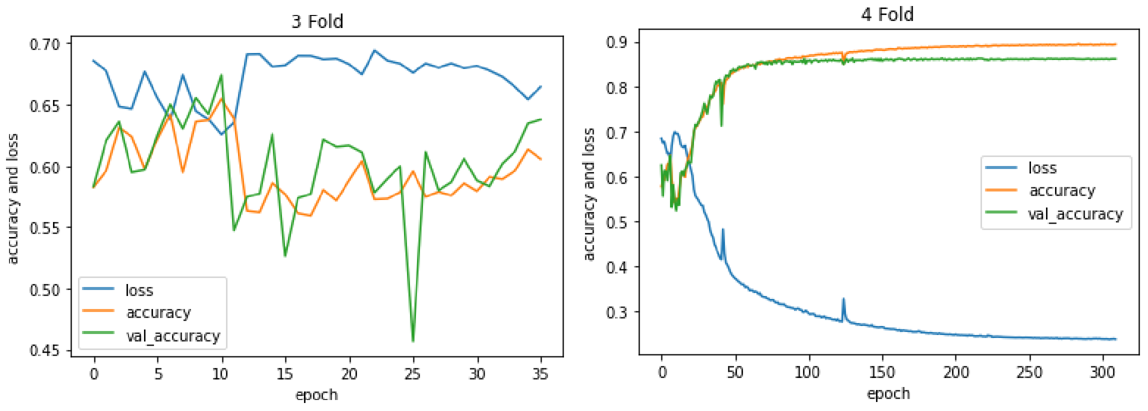


Figure 59 3rd and 4th Folds

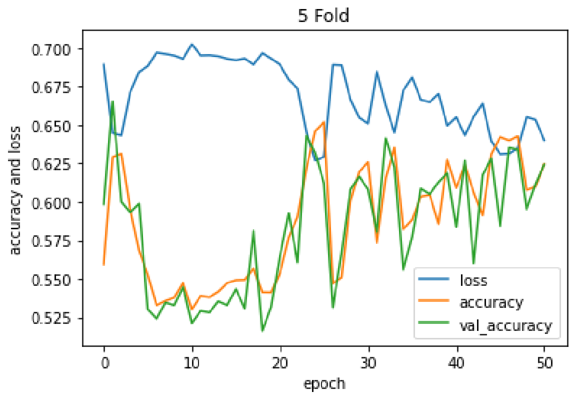


Figure 60 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.6322	0.6471	0.597	2.0	0.5971	0.5969	0.597
Fold 2	0.6514	0.6448	0.6765	8.0	0.6768	0.6763	0.6766
Fold 3	0.6546	0.674	0.5995	10.0	0.5996	0.5993	0.5995
Fold 4	0.8883	0.8628	**0.8267**	191.0	0.8267	0.8263	0.8265
Fold 5	0.6291	0.6653	0.7363	1.0	0.7365	0.7361	0.7363
Average	0.6911	0.6988	0.6872	42.4	0.6873	0.687	0.6872

Figure 61 Metrics

AccV channel:

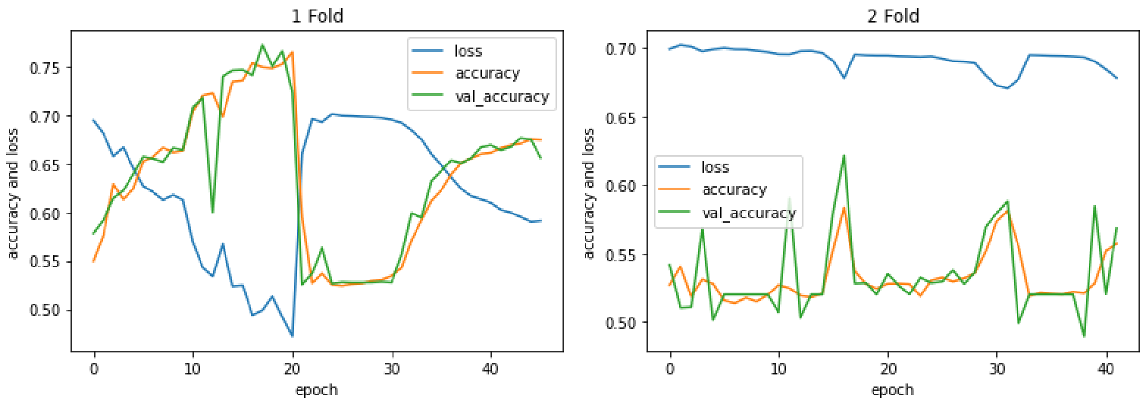


Figure 62 1st and 2nd Folds

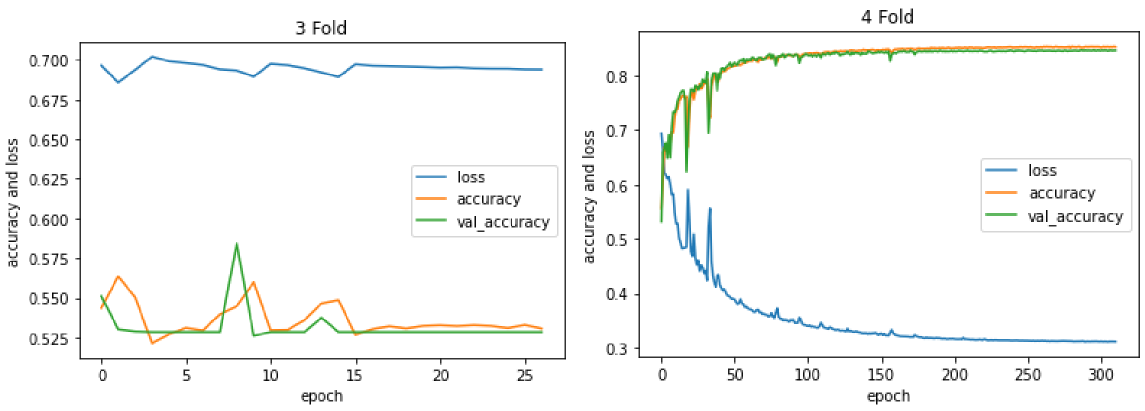


Figure 63 3rd and 4th Folds

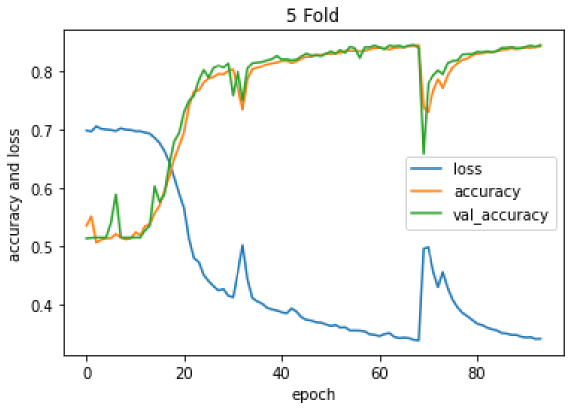


Figure 64 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7493	0.7723	0.766	17.0	0.7662	0.7659	0.7661
Fold 2	0.5837	0.6216	0.6584	16.0	0.6587	0.6582	0.6584
Fold 3	0.5446	0.5842	0.5752	8.0	0.5751	0.5748	0.575
Fold 4	0.8525	0.8471	**0.8233**	295.0	0.8235	0.8231	0.8233
Fold 5	0.8439	0.8457	0.8197	67.0	0.8199	0.8195	0.8197
Average	0.7148	0.7342	0.7285	80.6	0.7287	0.7283	0.7285

Figure 65 Metrics

Two channels:

AccML and AccAP:

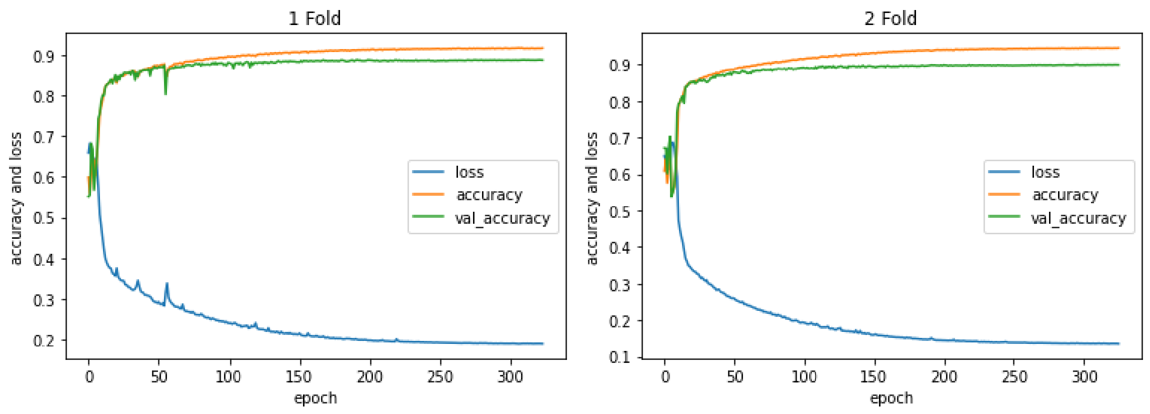


Figure 66 1st and 2nd Folds

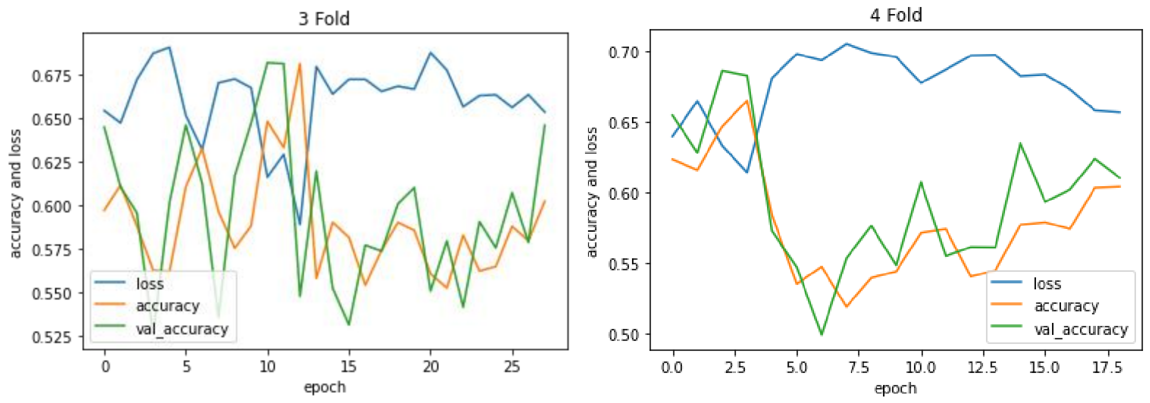


Figure 67 3rd and 4th Folds

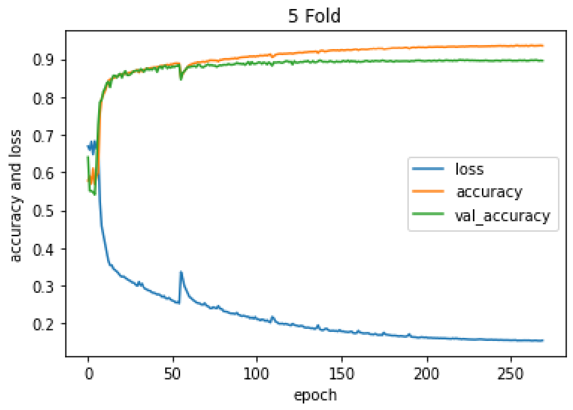
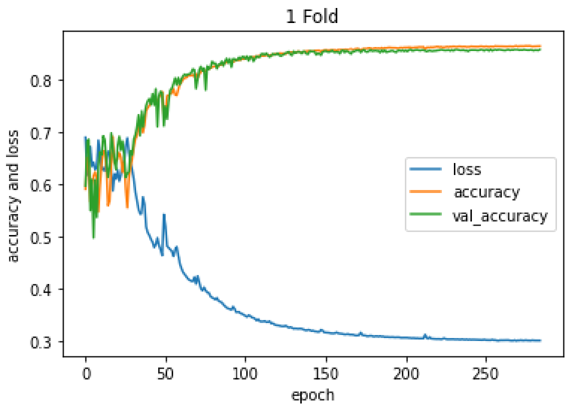


Figure 68 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.9118	0.8884	**0.8654**	191.0	0.8656	0.8653	0.8654
Fold 2	0.9447	0.9	0.8435	295.0	0.8438	0.8432	0.8435
Fold 3	0.6483	0.6819	0.6556	10.0	0.6558	0.6554	0.6556
Fold 4	0.6465	0.6858	0.6434	2.0	0.6436	0.6433	0.6434
Fold 5	0.9334	0.8972	0.8276	220.0	0.8278	0.8274	0.8276
Average	0.8169	0.8107	0.7671	143.6	0.7673	0.7669	0.7671

Figure 69 Metrics

AccV and AccAP:



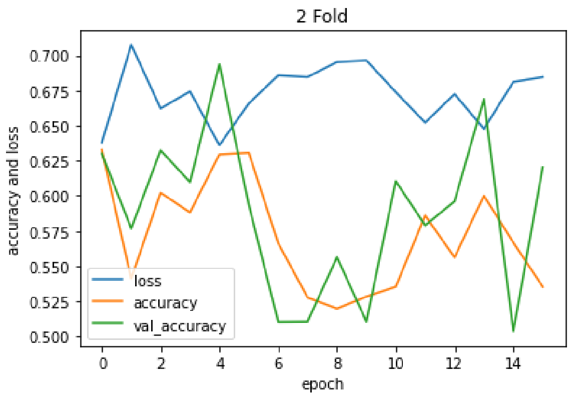
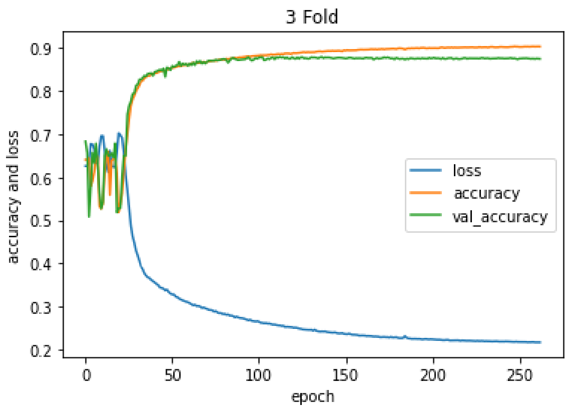


Figure 70 1st and 2nd Folds



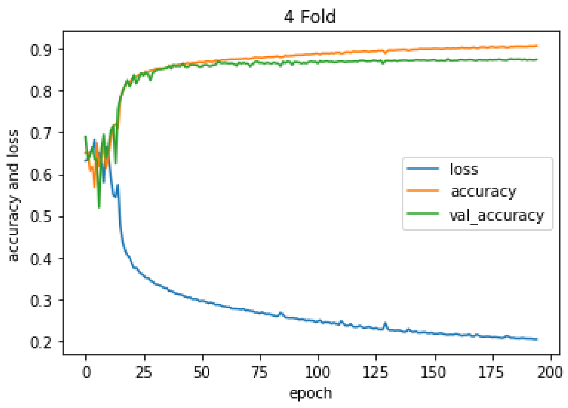


Figure 71 3rd and 4th Folds

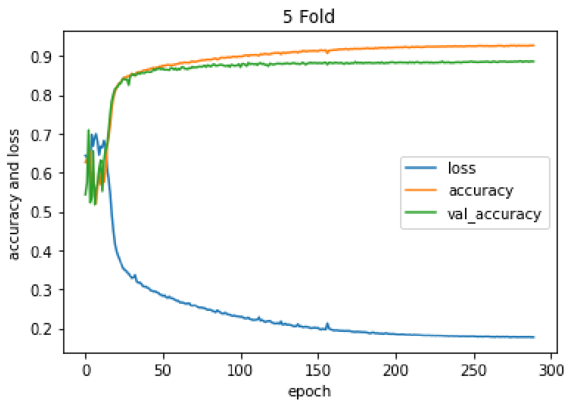
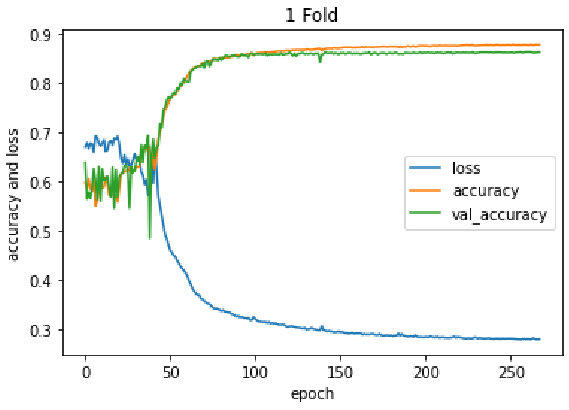


Figure 72 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.8652	0.8593	0.8455	253.0	0.8457	0.8454	0.8456
Fold 2	0.6294	0.6935	0.703	4.0	0.7034	0.7029	0.7031
Fold 3	0.8853	0.8796	**0.8544**	112.0	0.8547	0.8543	0.8545
Fold 4	0.9009	0.8751	0.8487	156.0	0.8489	0.8484	0.8487
Fold 5	0.9258	0.887	0.828	280.0	0.8282	0.8278	0.828
Average	0.8413	0.8389	0.8159	161.0	0.8162	0.8158	0.816

Figure 73 Metrics

AccV and AccML:



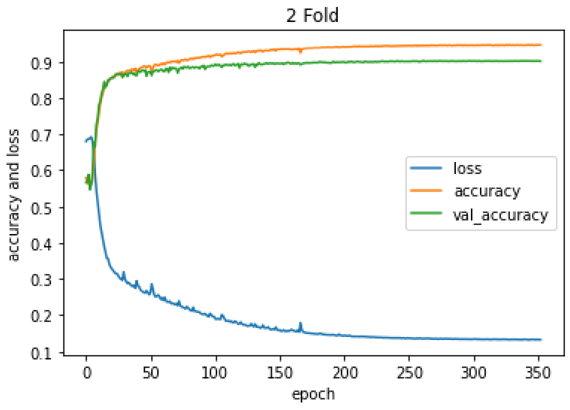
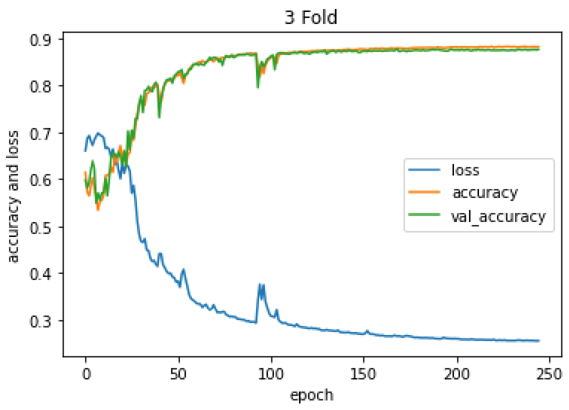


Figure 74 1st and 2nd Folds



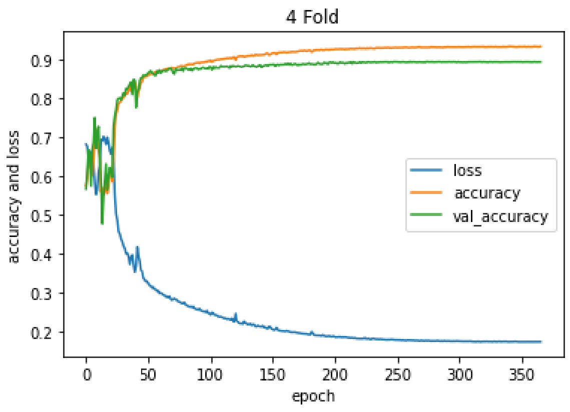


Figure 75 3rd and 4th Fold

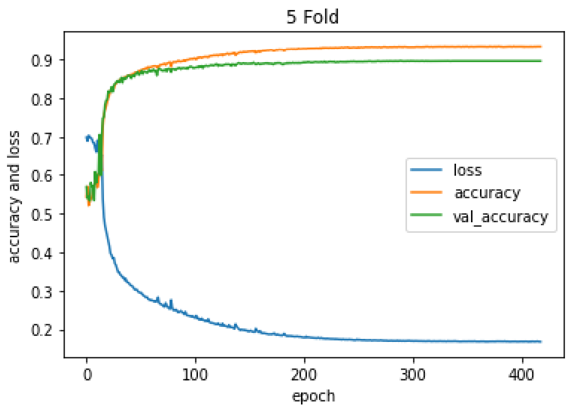


Figure 76 5th Fold

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.868	0.8623	**0.8612**	141.0	0.8613	0.861	0.8612
Fold 2	0.9448	0.9026	0.8335	261.0	0.8338	0.8332	0.8335
Fold 3	0.8788	0.8768	0.8379	168.0	0.8381	0.8377	0.8379
Fold 4	0.9294	0.8936	0.8369	260.0	0.837	0.8365	0.8367
Fold 5	0.933	0.8964	0.8348	289.0	0.8352	0.8348	0.835
Average	0.9108	0.8863	0.8409	223.8	0.8411	0.8407	0.8409

Figure 77 Metrics

Sequence to Sequence Models:

Model 1:

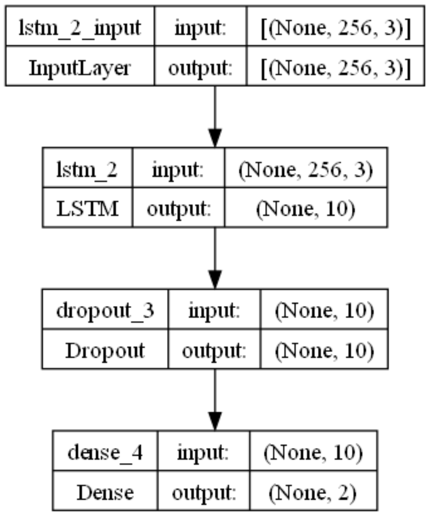

lstm_2_input | input: | [(None, 256, 3)]
InputLayer | output: | [(None, 256, 3)]
lstm_2 | input: | (None, 256, 3)
LSTM | output: | (None, 10)
dropout_3 | input: | (None, 10)
Dropout | output: | (None, 10)
dense_4 | input: | (None, 10)
Dense | output: | (None, 2)

Figure 78 Model 1 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7349	0.746	0.7438	92.0	0.7438	0.7438	0.7438
Fold 2	0.7116	0.7231	0.7073	5.0	0.7073	0.7073	0.7073
Fold 3	0.6954	0.7345	0.7196	7.0	0.7196	0.7196	0.7196
Fold 4	0.7286	0.7273	0.6936	20.0	0.6936	0.6936	0.6936
Fold 5	0.7263	0.7453	**0.7452**	30.0	0.7452	0.7452	0.7452
Average	0.7193	0.7353	0.7219	30.8	0.7219	0.7219	0.7219

Figure 79 Model 1 Metrics

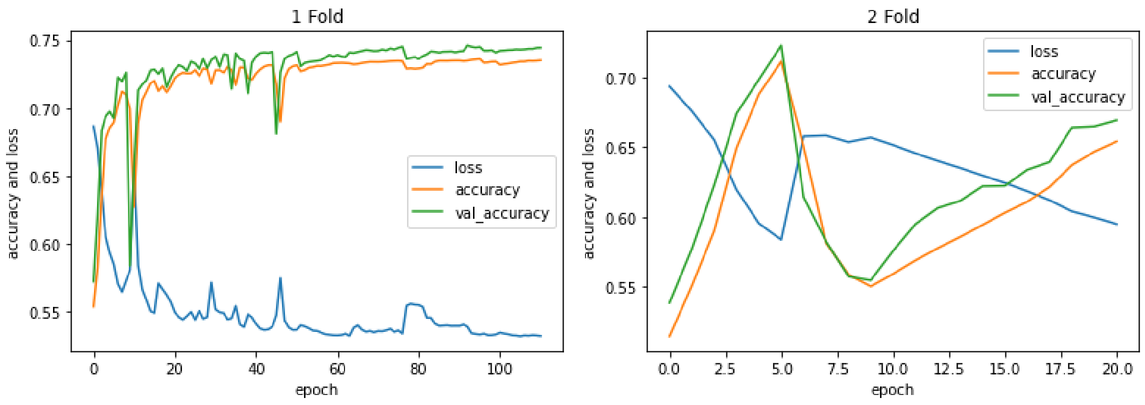


Figure 80 Model 1 1st and 2nd Folds

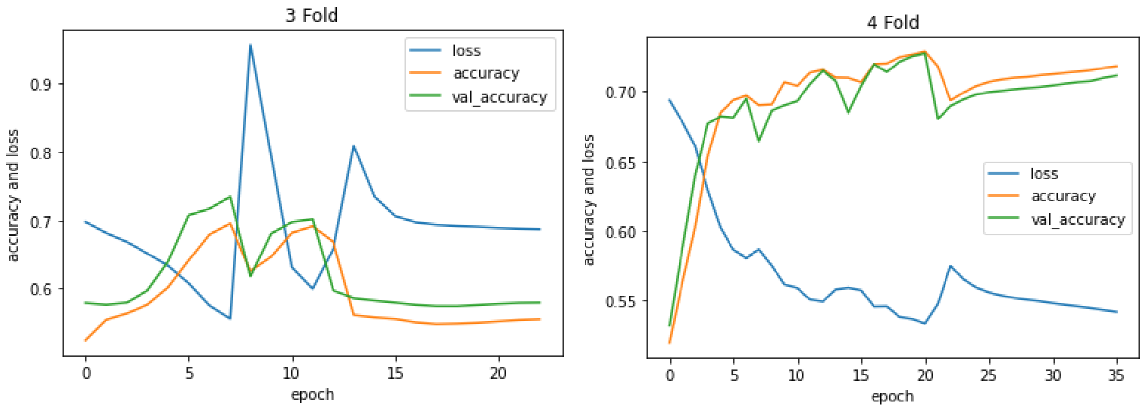


Figure 81 Model 1 2nd and 3rd Folds

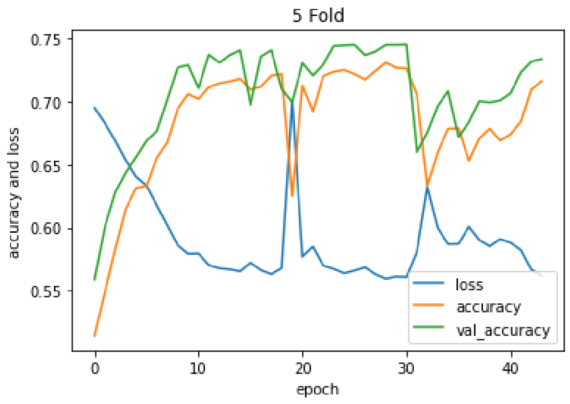


Figure 82 Model 1 5th Fold

Model 2:

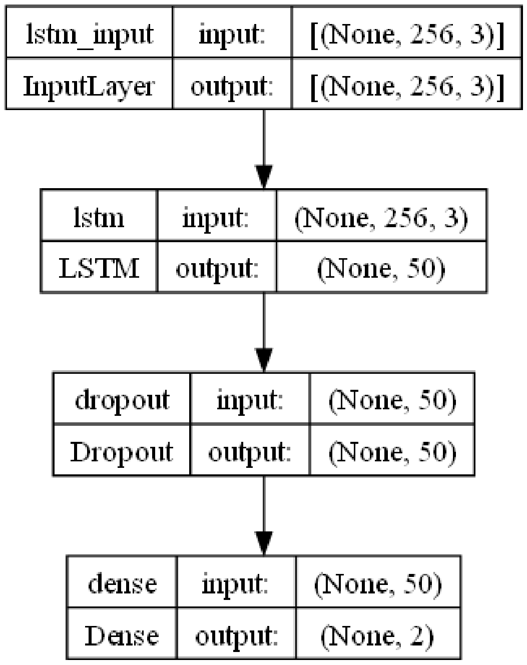


Figure 83 Model 2 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7607	0.7612	0.7534	26.0	0.7534	0.7534	0.7534
Fold 2	0.799	0.8052	0.7664	119.0	0.7664	0.7664	0.7664
Fold 3	0.7483	0.7439	0.7582	18.0	0.7582	0.7582	0.7582
Fold 4	0.7644	0.7802	0.7209	6.0	0.7209	0.7209	0.7209
Fold 5	0.795	0.8111	**0.7805**	54.0	0.7804	0.7805	0.7805
Average	0.7735	0.7803	0.7559	44.6	0.7559	0.7559	0.7559

Figure 84 Model 2 Metrics

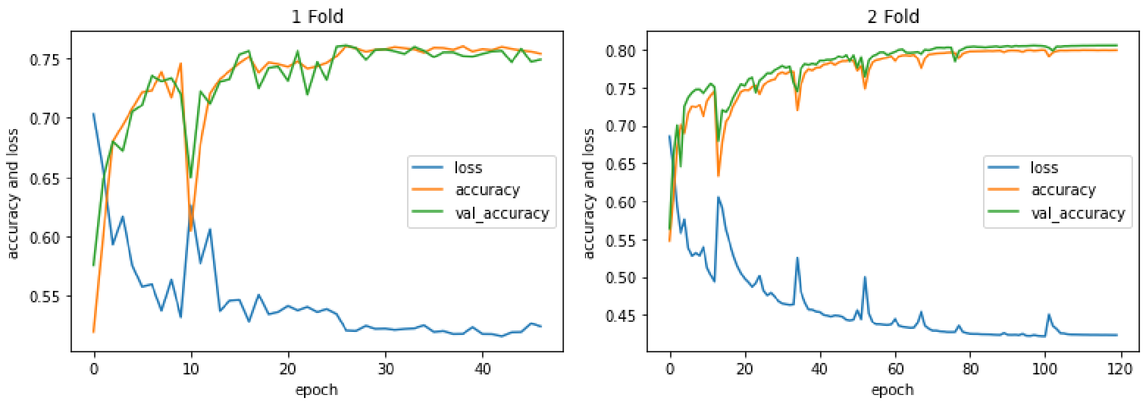


Figure 85 Model 2 1st and 2nd Folds

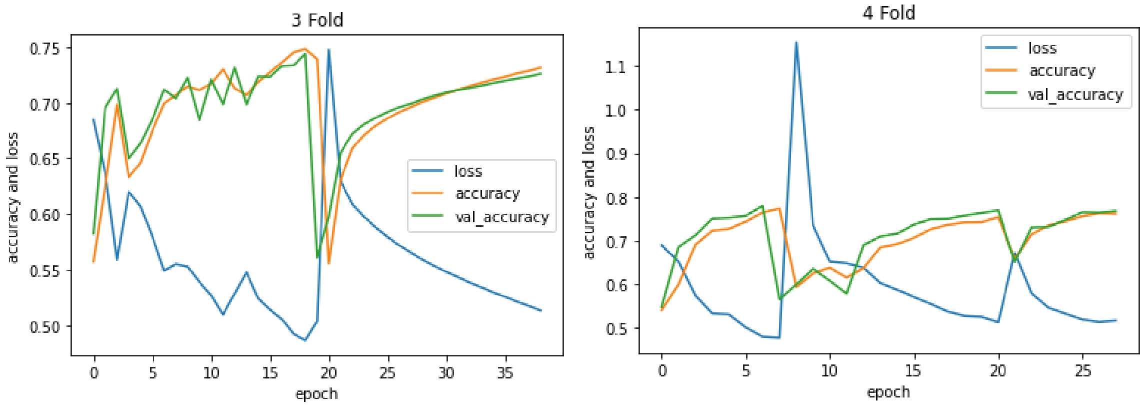


Figure 86 Model 2 3rd and 4th Folds

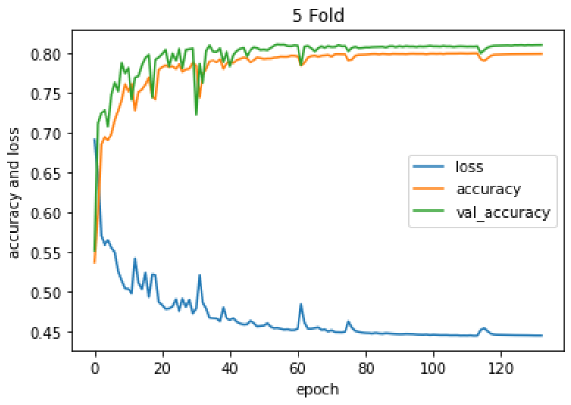
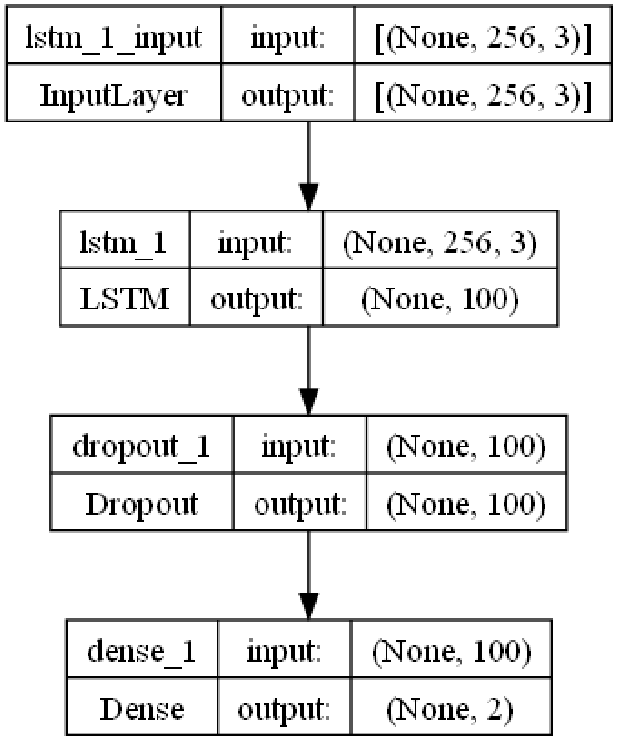


Figure 87 Model 2 5th Fold

Model 3:

Model 3 was done on three different settings

Setting 1: Exponential decay learning rate initially 0.001 decaying by 90% with 128 batch size



lstm_1_input | input: | [(None, 256, 3)]
InputLayer | output: | [(None, 256, 3)]
lstm_1 | input: | (None, 256, 3)
LSTM | output: | (None, 100)
dropout_1 | input: | (None, 100)
Dropout | output: | (None, 100)
dense_1 | input: | (None, 100)
Dense | output: | (None, 2)

Figure 88 Model 3 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7617	0.7732	**0.7675**	30.0	0.7675	0.7675	0.7675
Fold 2	0.733	0.7493	0.7024	19.0	0.7024	0.7024	0.7024
Fold 3	0.7564	0.7424	0.7507	22.0	0.7507	0.7507	0.7507
Fold 4	0.732	0.7501	0.7186	11.0	0.7186	0.7186	0.7186
Fold 5	0.7369	0.7597	0.7348	15.0	0.7348	0.7348	0.7348
Average	0.744	0.7549	0.7348	19.4	0.7348	0.7348	0.7348

Figure 89 Model 3 settings 1 Metrics

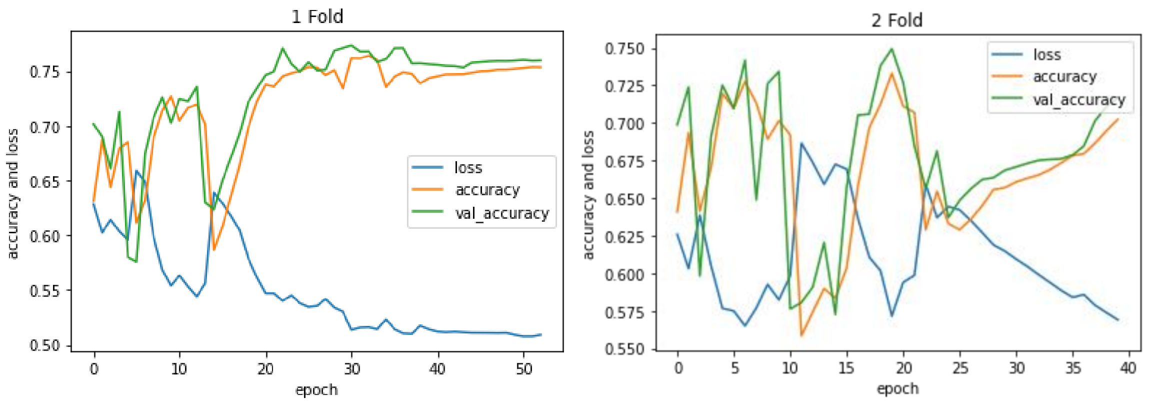


Figure 90 1st and 2nd Folds

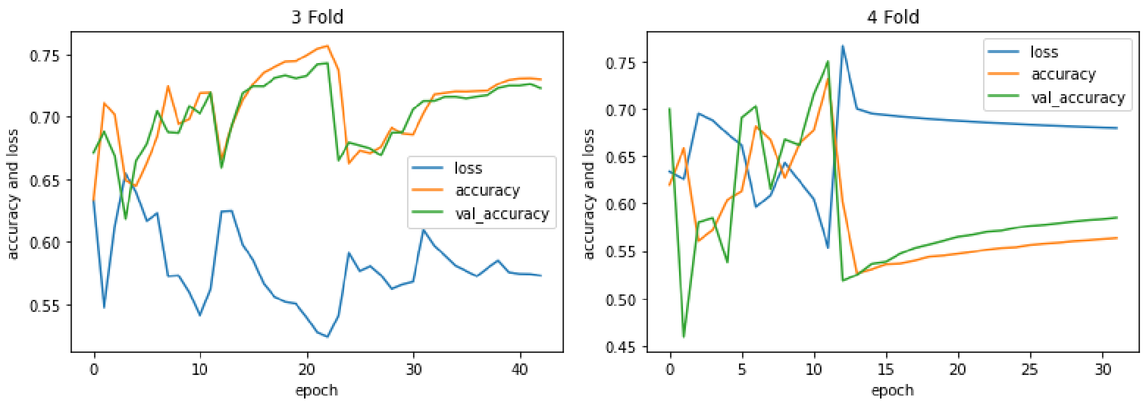


Figure 91 3rd and 4th Folds

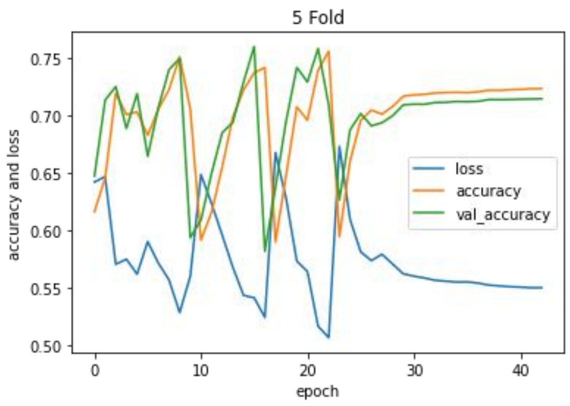


Figure 92 5th Fold

Setting 2: learning rate set to 1e-5 with no decay and normal Xavier weight initializer

Normal Xavier initialization: draw each weight, w, from a normal distribution with a mean of 0, and a standard deviation

$$\sigma = \sqrt{\frac{2}{\text{inputs} + \text{outputs}}}$$

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7344	0.7251	0.7395	129.0	0.7395	0.7395	0.7395
Fold 2	0.754	0.7509	0.7275	140.0	0.7275	0.7275	0.7275
Fold 3	0.7326	0.7294	0.7455	77.0	0.7455	0.7455	0.7455
Fold 4	0.7597	0.7496	0.6883	77.0	0.6883	0.6883	0.6883
Fold 5	0.7479	0.7361	**0.7576**	180.0	0.7576	0.7576	0.7576
Average	0.7457	0.7382	0.7317	120.6	0.7317	0.7317	0.7317

Figure 93 Model 3 settings 2 Metrics

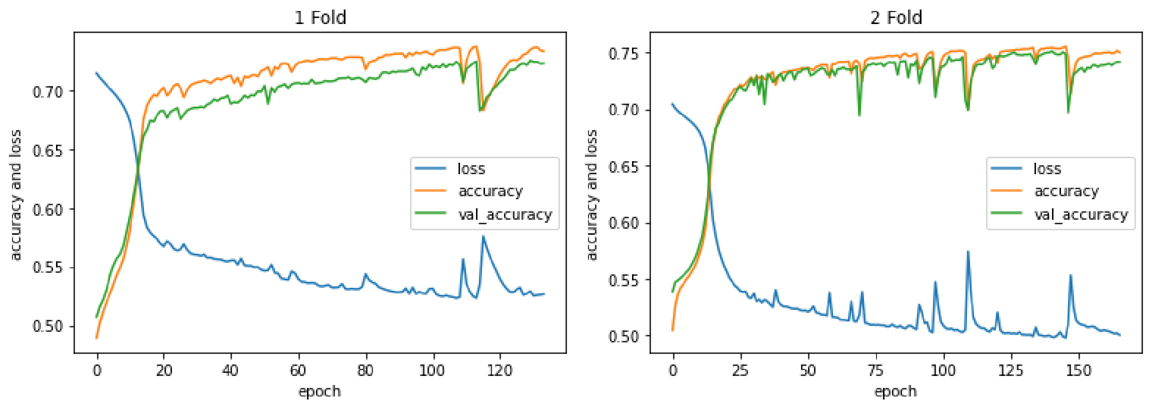


Figure 94 1st and 2nd Folds

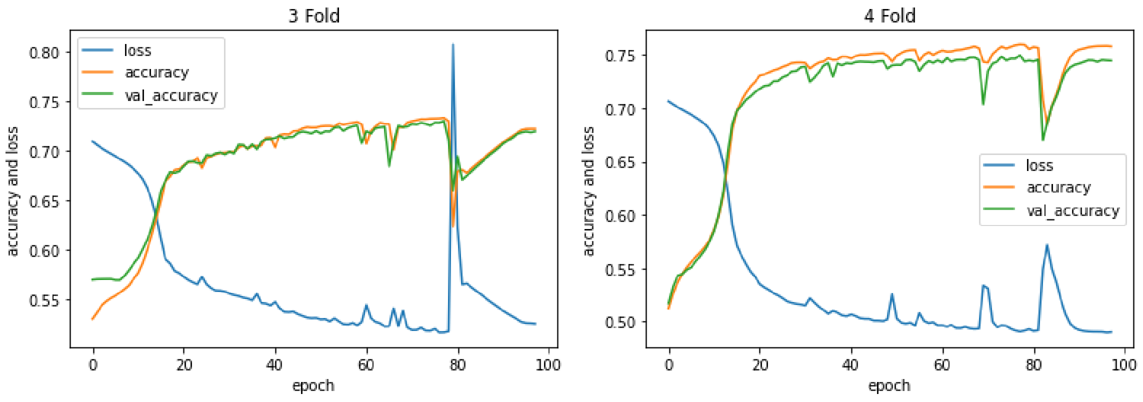


Figure 95 3rd and 4th Folds

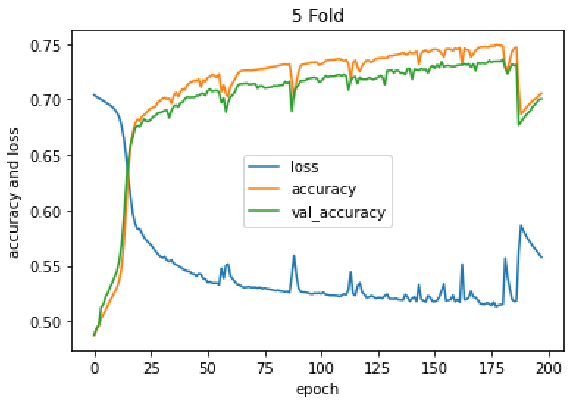


Figure 96 5th Fold

Setting 3: Exponential decay learning rate initially 0.0001 decaying by 90% and 256 batch size with normal Xavier initializer and 1 clip value

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7952	0.7956	0.8081	101.0	0.8081	0.8081	0.8081
Fold 2	0.8181	0.8265	0.7809	436.0	0.7809	0.7809	0.7809
Fold 3	0.7677	0.7732	0.7915	37.0	0.7915	0.7915	0.7915
Fold 4	0.8315	0.8359	0.7925	456.0	0.7925	0.7925	0.7925
Fold 5	0.827	0.8339	**0.8177**	475.0	0.8177	0.8177	0.8177
Average	0.8079	0.813	0.7981	301.0	0.7981	0.7981	0.7981

Figure 97 Model 3 settings 3 Metrics

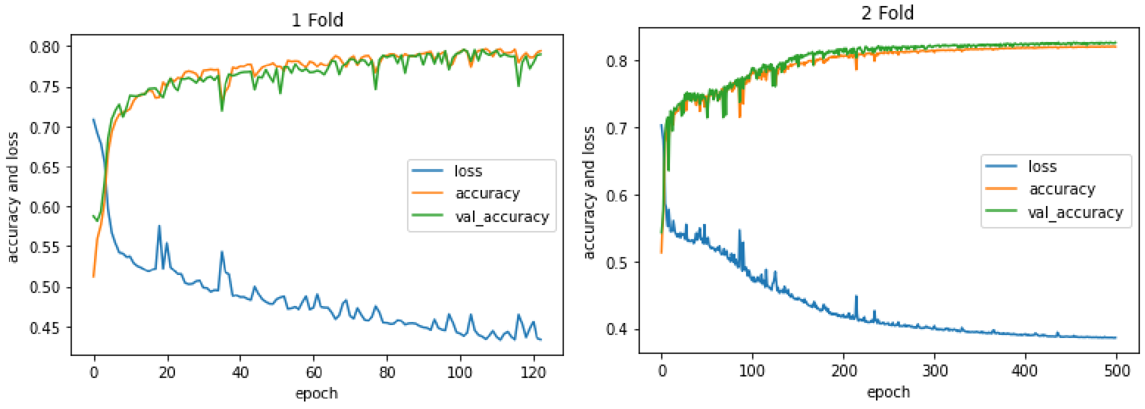


Figure 98 1st and 2nd Folds

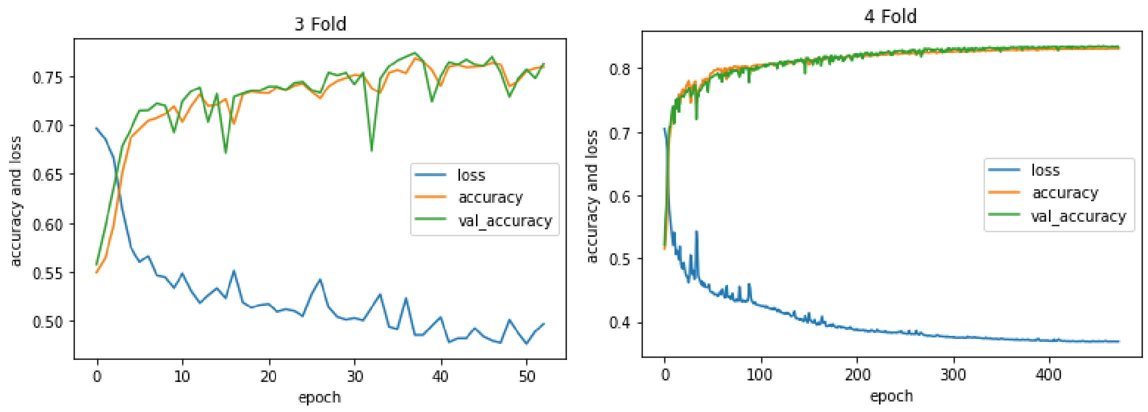


Figure 99 3rd and 4th Folds

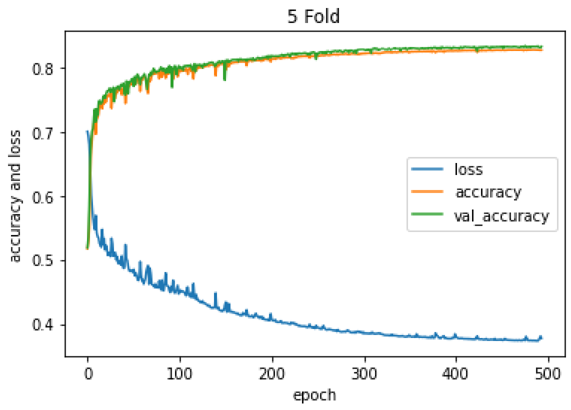


Figure 100 5th Fold

Model 4:

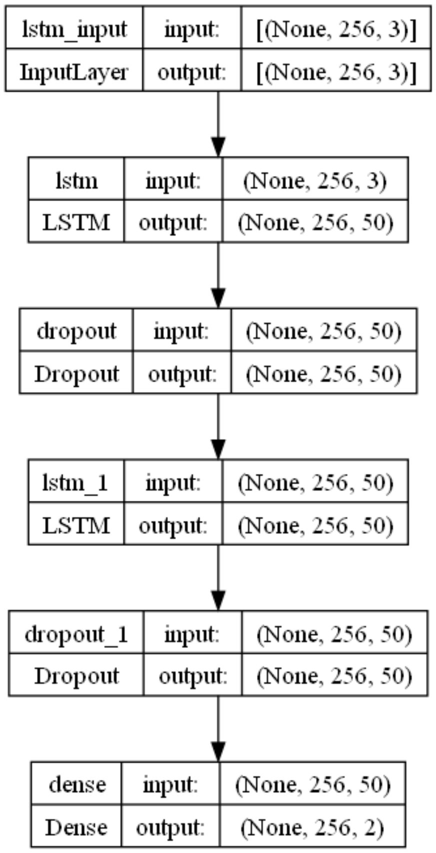

lstm_input | input: | [(None, 256, 3)]
InputLayer | output: | [(None, 256, 3)]
lstm | input: | (None, 256, 3)
LSTM | output: | (None, 256, 50)
dropout | input: | (None, 256, 50)
Dropout | output: | (None, 256, 50)
lstm_1 | input: | (None, 256, 50)
LSTM | output: | (None, 256, 50)
dropout_1 | input: | (None, 256, 50)
Dropout | output: | (None, 256, 50)
dense | input: | (None, 256, 50)
Dense | output: | (None, 256, 2)

Figure 101 Model 4 architecture

	Training accuracy	Validation accuracy	Test accuracy	epochs	Recall	Precision	F1
Fold 1	0.7512	0.7563	0.772	43.0	0.772	0.772	0.772
Fold 2	0.7984	0.7934	0.7728	177.0	0.7728	0.7728	0.7728
Fold 3	0.7604	0.7754	**0.7779**	66.0	0.7779	0.7779	0.7779
Fold 4	0.7937	0.8082	0.7599	326.0	0.7599	0.7599	0.7599
Fold 5	0.7437	0.7565	0.759	43.0	0.759	0.759	0.759
Average	0.7695	0.778	0.7683	131.0	0.7683	0.7683	0.7683

Figure 102 Model 4 Metrics

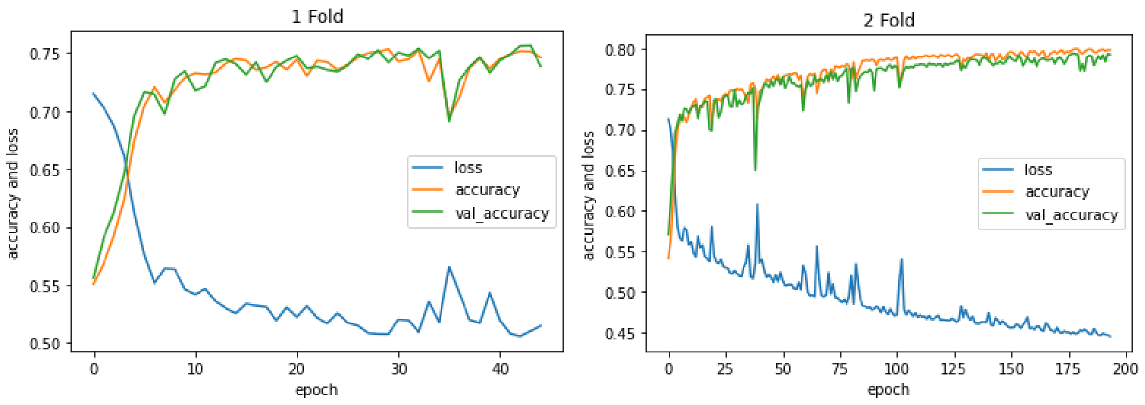


Figure 103 1st and 2nd Folds

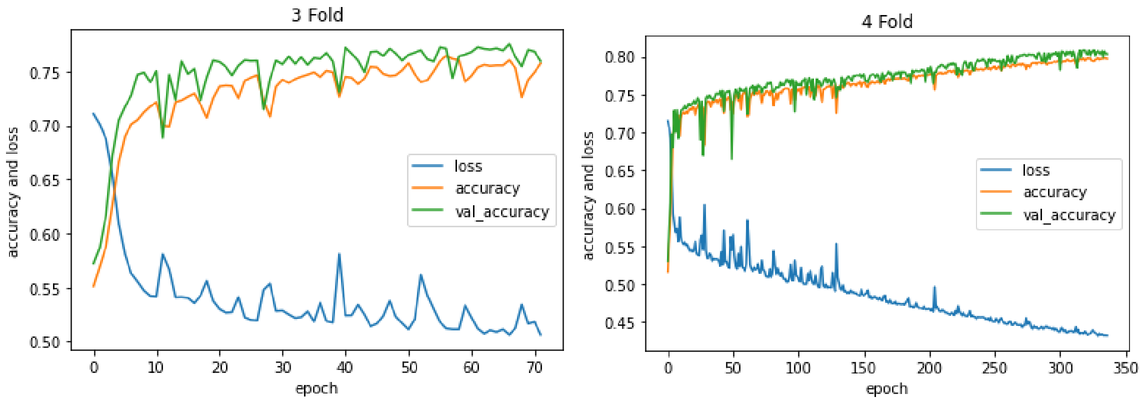


Figure 104 3rd and 4th Folds

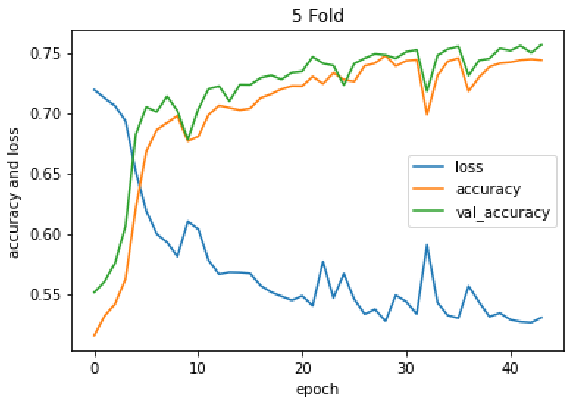


Figure 105 5th Fold

Results

What is the best Model?

Best model in the sequence to label category was found to be Model 3 for cross validation

And Model 5 using cyclic learning for single Fold

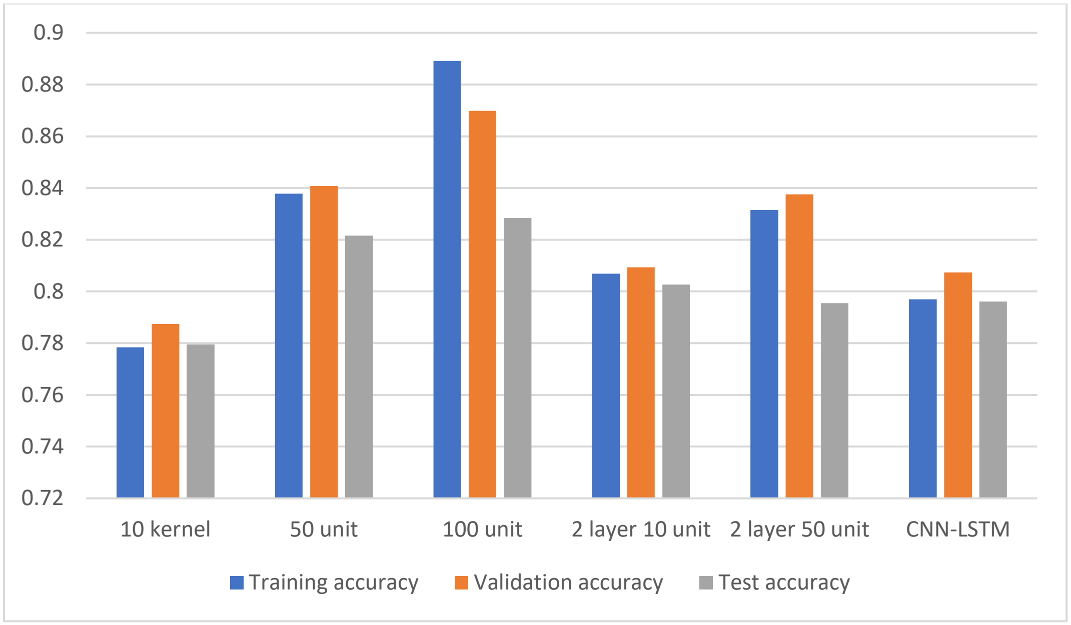


Figure 106 Sequence to sequence model metrics

What is the best time interval?

Best time interval was found to be ½ a second

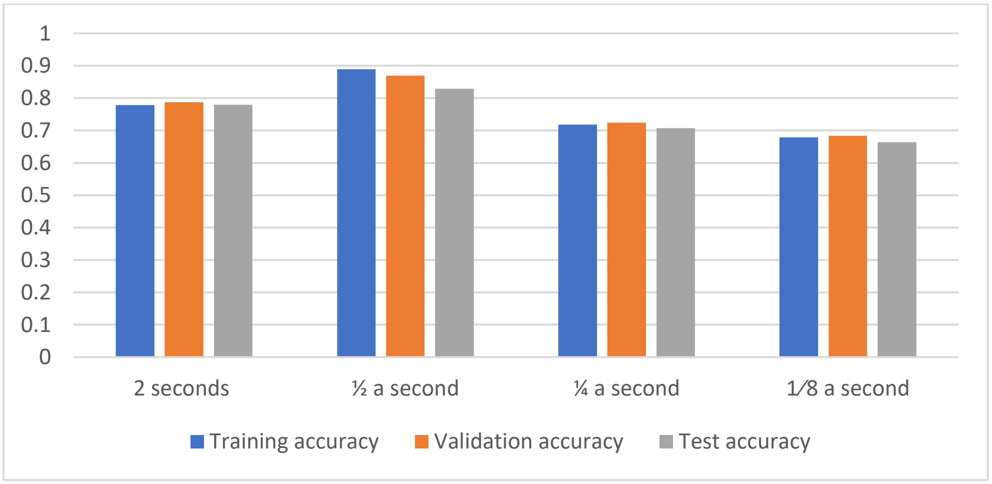


Figure 107 Time intervals metrics

What is the best combinations of channels?

Best channel combination was AccML and AccAP reaching test accuracy of 0.8654

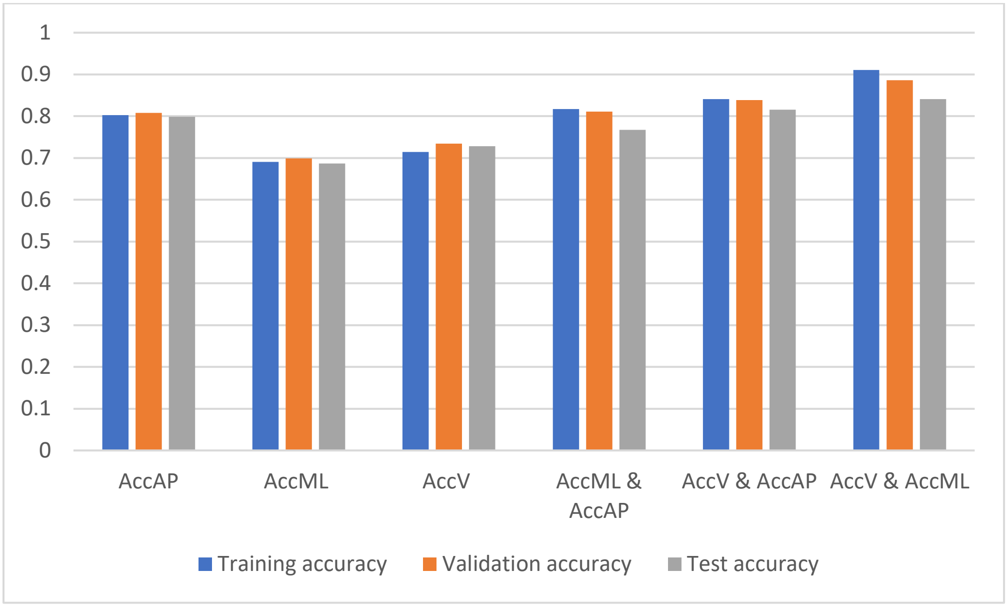
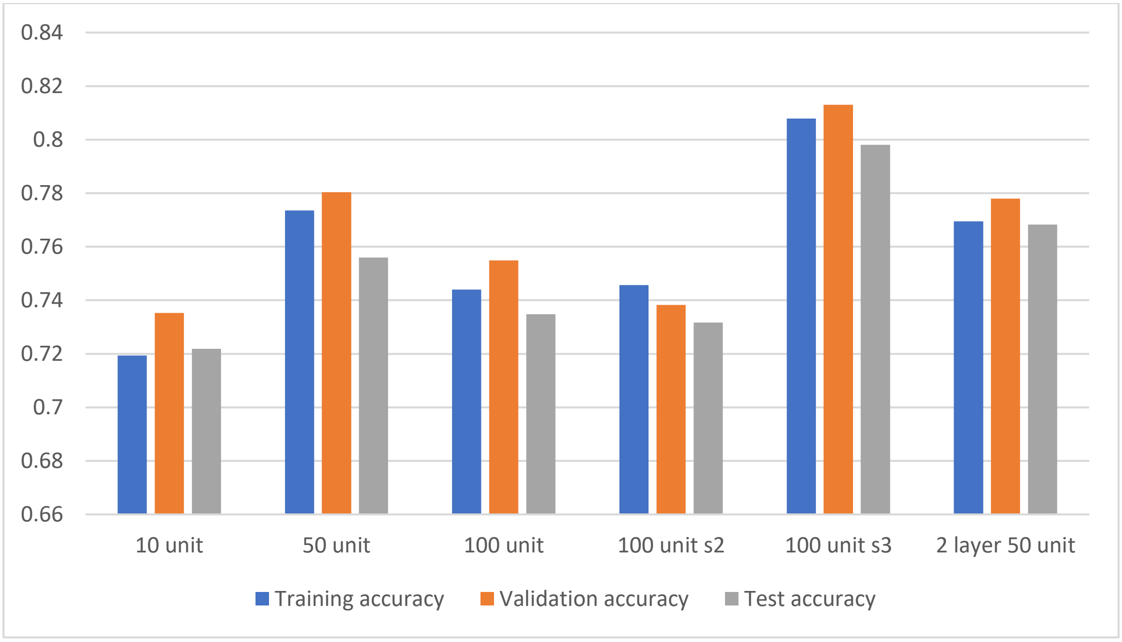


Figure 108 Different channels metrics

What is the best sequence to sequence Model?

Best sequence to sequence model reached was model 3 with settings 3 reaching test accuracy of **0.8177**



What is the best engineered features shallow model?

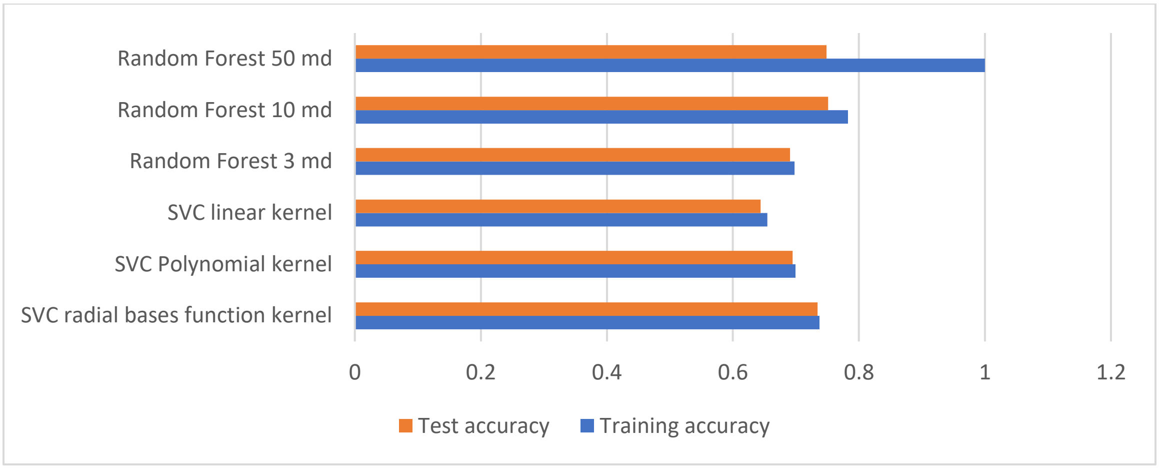


Figure 109 Best shallow model for 2 seconds

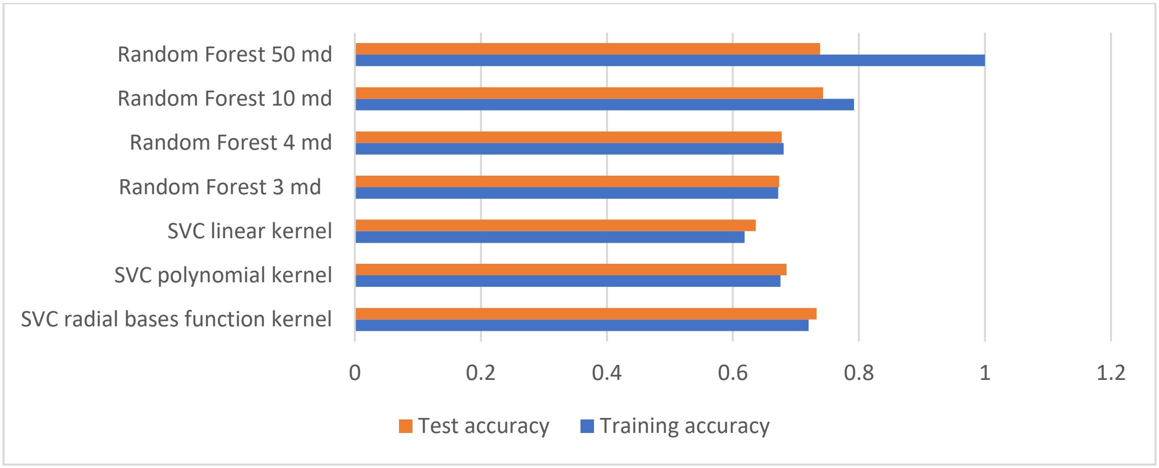


Figure 110 Best shallow model for 4 seconds