# CSE6250: Big Data Analytics in Healthcare Homework 4

Mustafa Al Salem

## 1.2 b

**Trainable parameters and approximate computations for the MLP**

We analyze a 3-layer MLP with input dimension 178, one hidden layer with 16 units and sigmoid activation, and an output layer with 5 units linear.

**Layer 1** $(178 \to 16)$

$$weights = 178 \times 16 = 2{,}848, \quad biases = 16 \quad \Rightarrow \quad params = 2{,}848 + 16 = 2{,}864$$

**Layer 2** $(16 \to 5)$

$$weights = 16 \times 5 = 80, \quad biases = 5 \quad \Rightarrow \quad params = 80 + 5 = 85$$

**Total.**

$$params(trainable) = 2{,}864 + 85 = 2{,}949$$

**Layer 1** $(178 \to 16)$**.** Each of the 16 hidden units takes input from all 178 features. Assuming $\approx 3$ , we get

$$178 \times 16 \times 3 = 8544$$

**Layer 2** $(16 \to 5)$

$$16 \times 5 \times 3 = 240$$

**Sigmoid on hidden layer** $(16$ **units).**

$$16 \times 3 = 48$$

**Total =** 8544 + 240 + 48 = 8832 floating–point operations (approx.)

### 1.2 c Number of epochs = 30

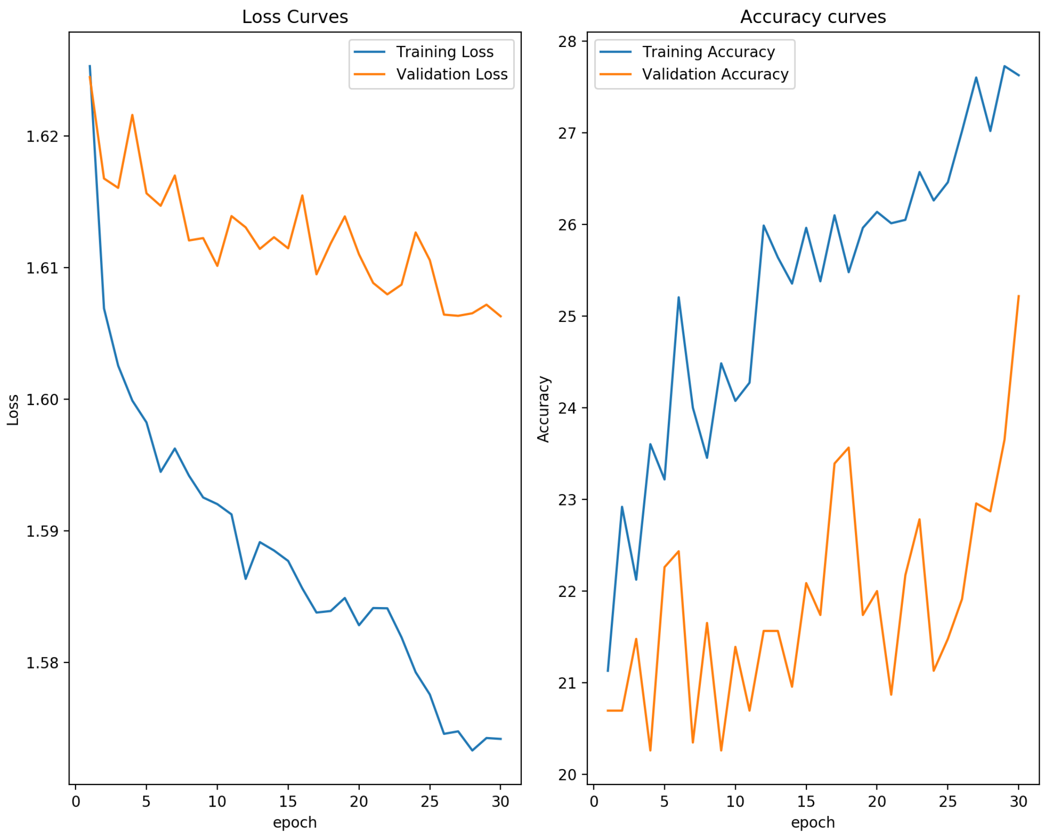

Figure 1: learning curves MLP unimpr

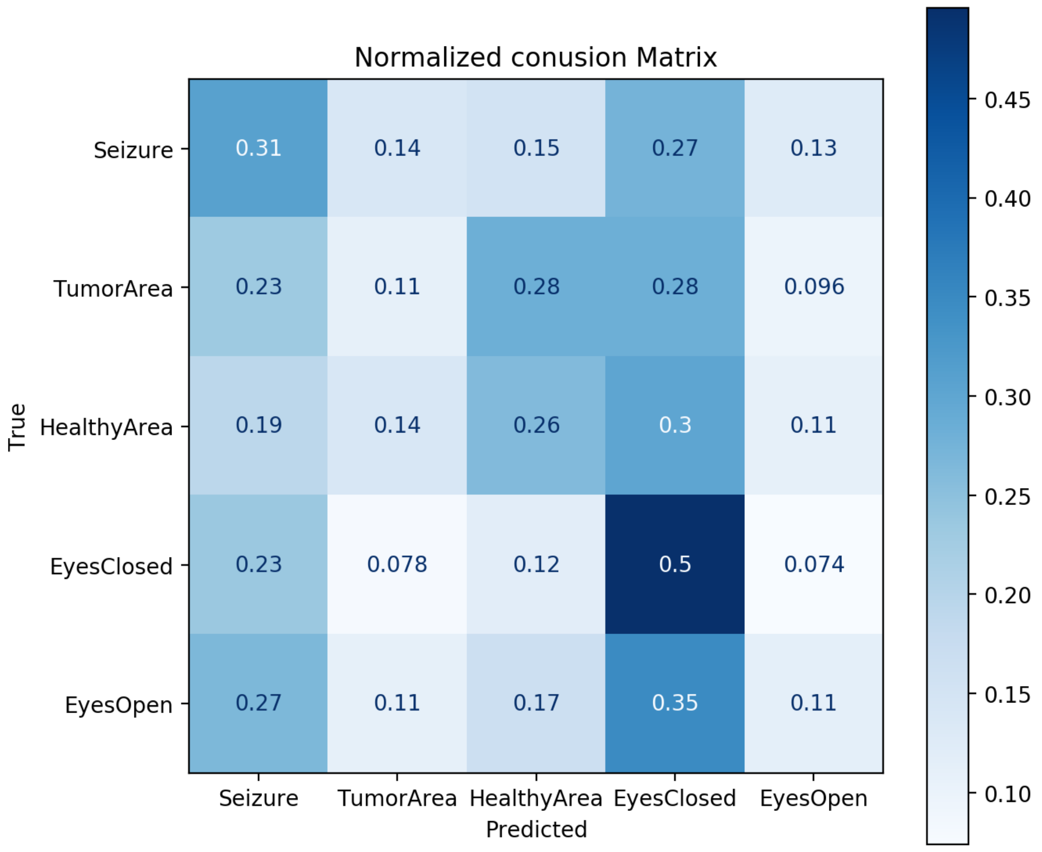

Figure 2: confusion matrix MLP unimpr

### 1.2 d Number of epochs = 30

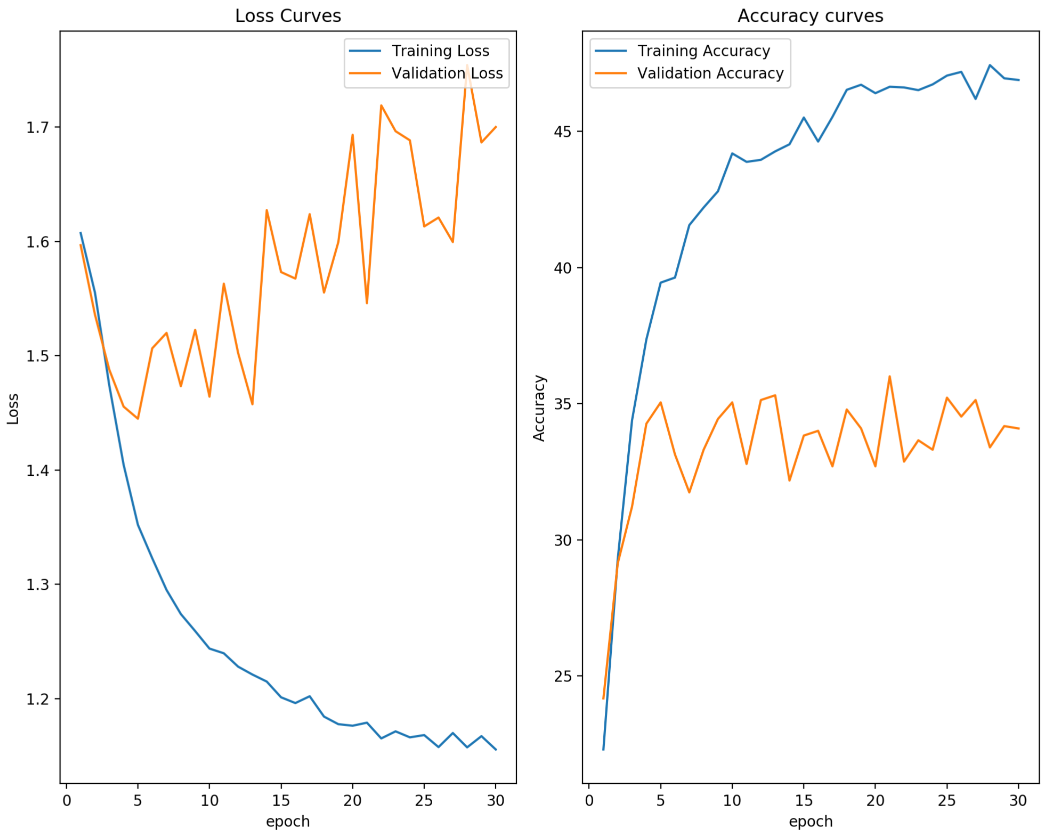

Figure 3: learning curves MLP improved

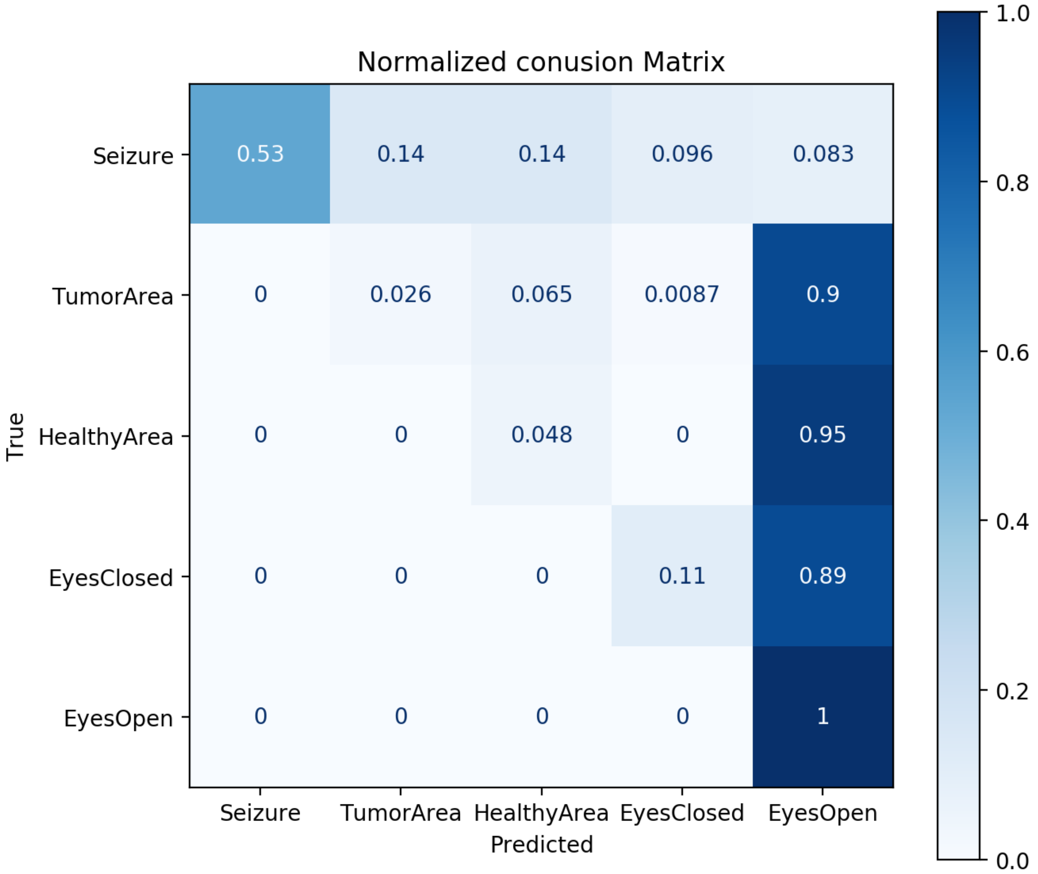

Figure 4: confusion matrix MLP impr

### 1.2 e

We applied several modifications to improve MLP capacity and generalization while still keeping the overall structure simple.

**1. Input normalization (178).)** Before the features are passed to the network, we normalize each mini-batch using normalization layer with 178 features:

$$BatchNorm1d(178).$$

This helps reduce internal covariate shift, stabilizes training over the full 30 epochs (batch size = 32), and also acts as a mild regularizer, which improves generalization.

**2. Wider first hidden layer (178 $\rightarrow$ 60).** We used a relatively narrow projection (e.g. $178 \rightarrow 16$) in our original MLP, which could very likely discard information too aggressively. But in the improved version we used

$$Linear(178, 60)$$

the original 178-dimensional input into a 60-dimensional hidden representation. This increases the model capacity and allows the network to learn richer, non-linear combinations of the input features.

**3. Dropout with $p = 0.5$.** Immediately after the first linear layer we apply dropout:

$$Dropout(p = 0.5),$$

which randomly zeroes out 50% of the hidden activations during training. This prevents the model from relying on a small subset of units and reduces overfitting, which is especially important because we train for 30 epochs on a dataset of limited size. After normalization, linear projection, and dropout, we apply a sigmoid activation, this introduces the non-linearity. The resulting hidden vector is then fed to the final classification layer

$$Linear(60, 5),$$

It was also noticed that MLP is the least expensive model in terms of comupation time.

## 1.3 b

**Trainable parameters and approximate computations for the CNN**

**Training setup.** We trained the CNN for 30 epochs with a batch size of 32.

**Parameter counting.** We approximate the number of trainable parameters layer by layer as follows.

- **Convolutional layer 1.** The layer uses kernel size $k = 5$, stride $s = 1$, and $F = 6$ filters. Using the given rule

  $$\big(k \times s\big) + 1 \quad thenmultipliedbythenumberoffilters,$$

  we get

  $$\big(5 \times 1\big) + 1 = 6 \quad \Rightarrow \quad 6 \times 6 = 36 parameters.$$

- **Convolutional layer 2.** This layer uses kernel size $k = 5$, takes 6 input channels, and has 16 output filters. With the same rule:

  $$\big(5 \times 6\big) + 1 = 31$$

### 1.3 c Number of epochs = 30

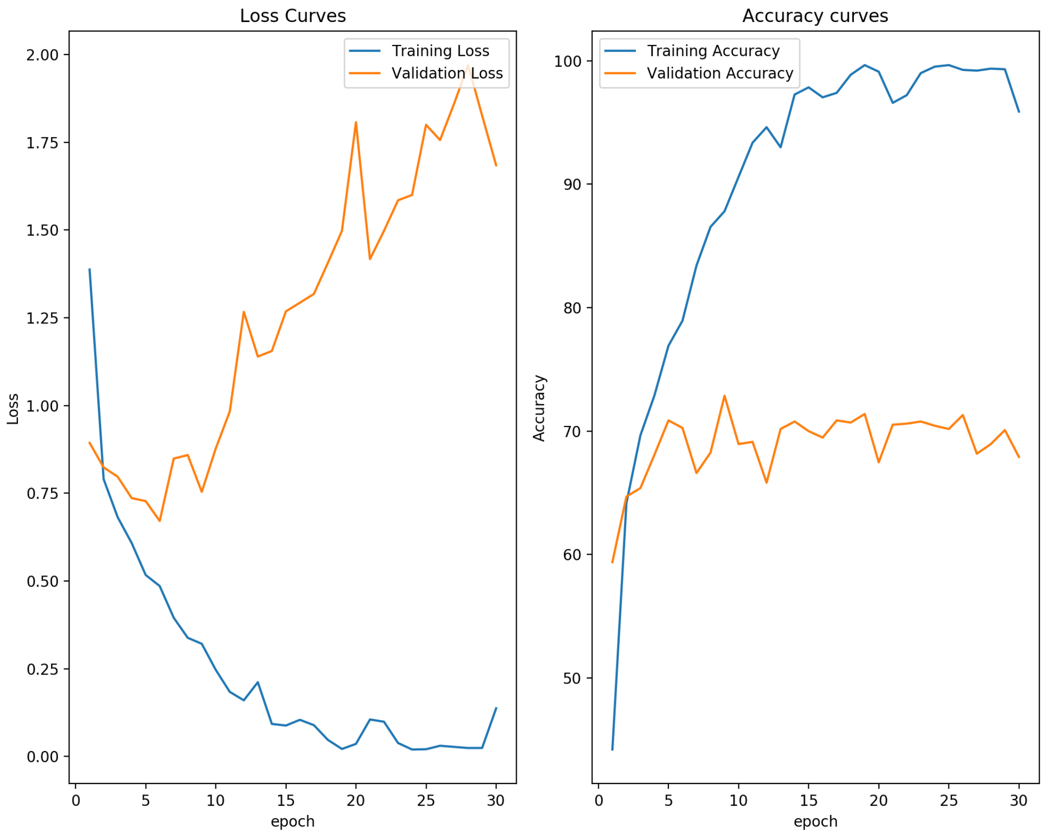

Figure 5: learning curves CNN unimpr

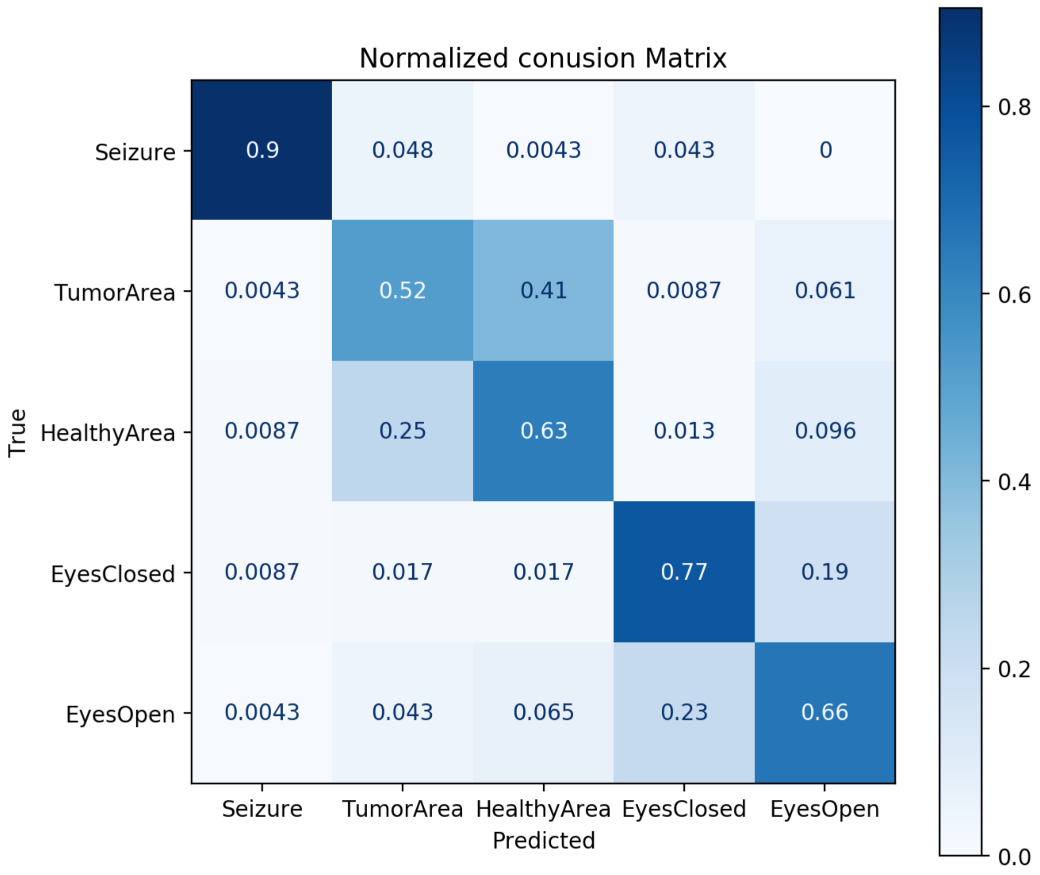

Figure 6: confusion matrix CNN unimpr

## 1.3 d Number of epochs = 30

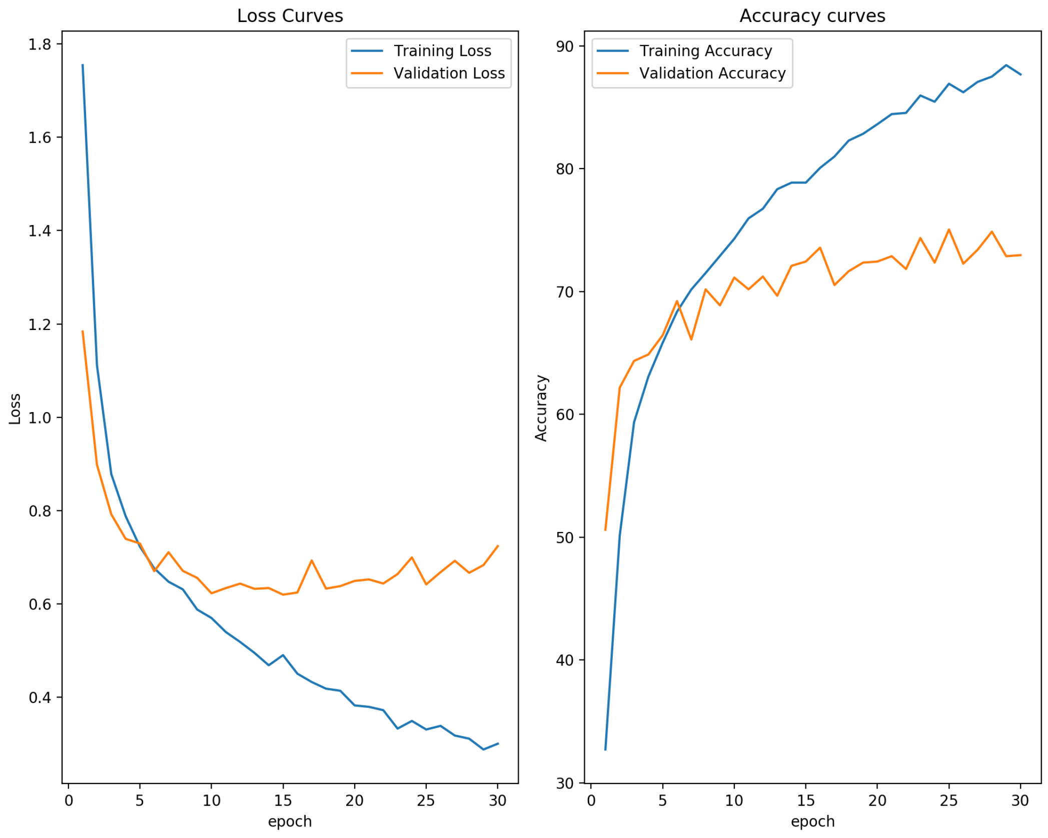

Figure 7: learning curves CNN impr

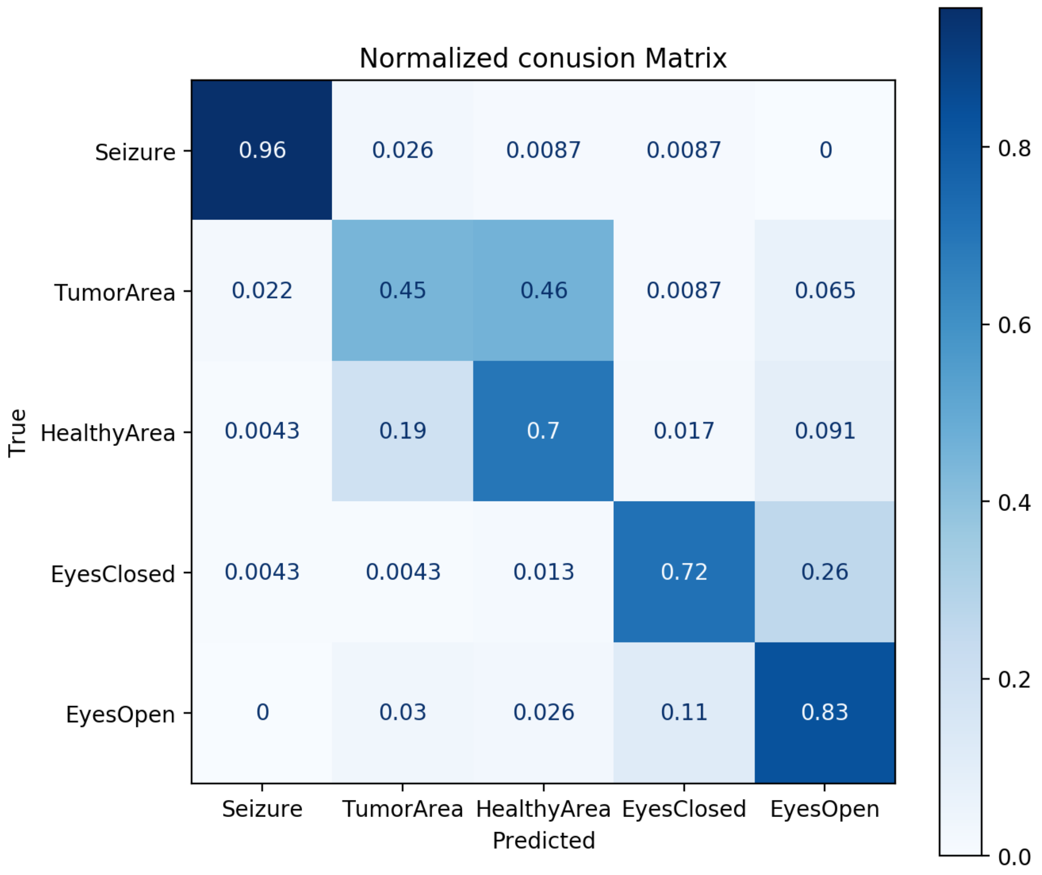

Figure 8: confusion matrix CNN impr

The only change that was done the dropout = 0.2 and everything else was ketp the same.Nothing else was changed because we wanted to see the impact of only dropout on the performance. I experimented with different dropout values and it was found that going beyond 0.2 decreased performance significantly. Also, when going below dropout ¡ 0.2 did not improve performance much and that why it seems 0.2 to be the ideal dropout value for this CNN model. From the plots above, it can be noticed the performance was positively improved just based on my ideal dropout value of 0.2.

## 1.4 b Number of epochs = 10

**Number of trainable parameters in the GRU.** A GRU layer with input size $I$, hidden size $H$, and $G = 3$ gates (reset, update, and candidate) has

$$Params = G\big[H(H + I) + H\big]$$

because each gate has an input-to-hidden weight $(H \times I)$, a hidden-to-hidden weight $(H \times H)$, and a bias $(H)$.

Substituting $I = 1$ and $H = 16$:

$$Params = 3\big[16(16+1)+16\big] = 3\big[16 \cdot 17 + 16\big] = 3\big[272+16\big] = 3 \times 288 = 864.$$

Therefore, the GRU has 864 trainable parameters.

## 1.4 c Number of epochs = 10

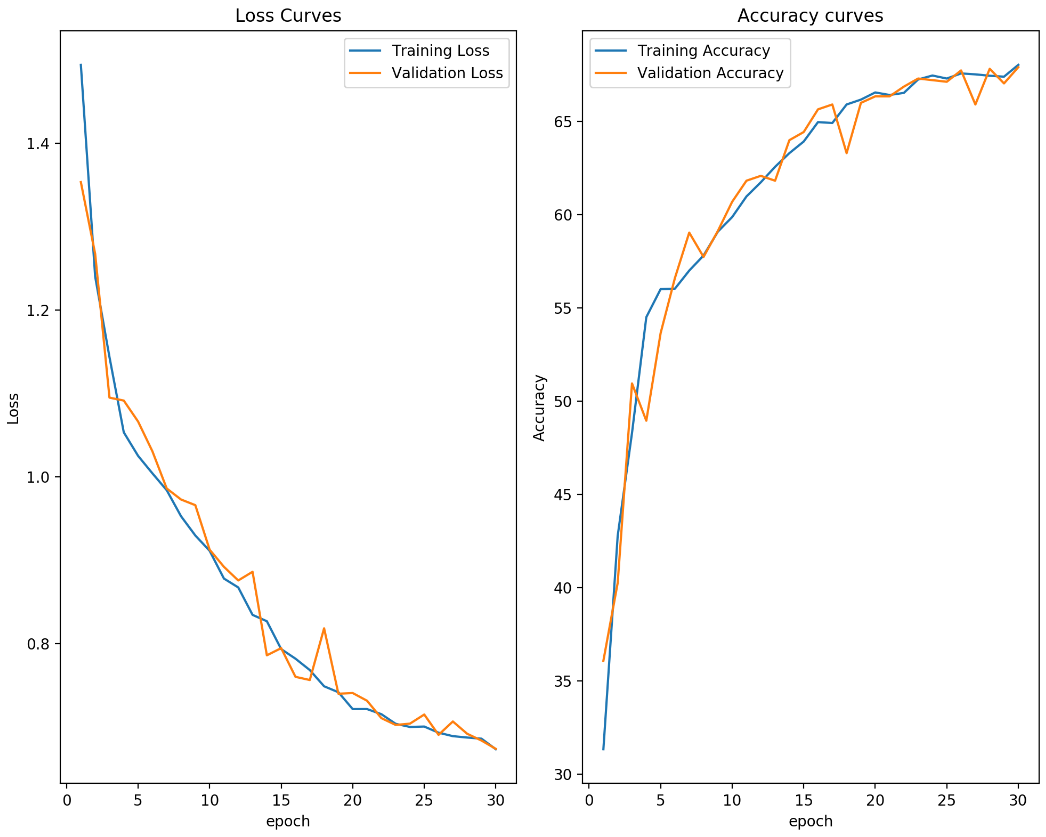

Figure 9: learning curves RNN unimpr

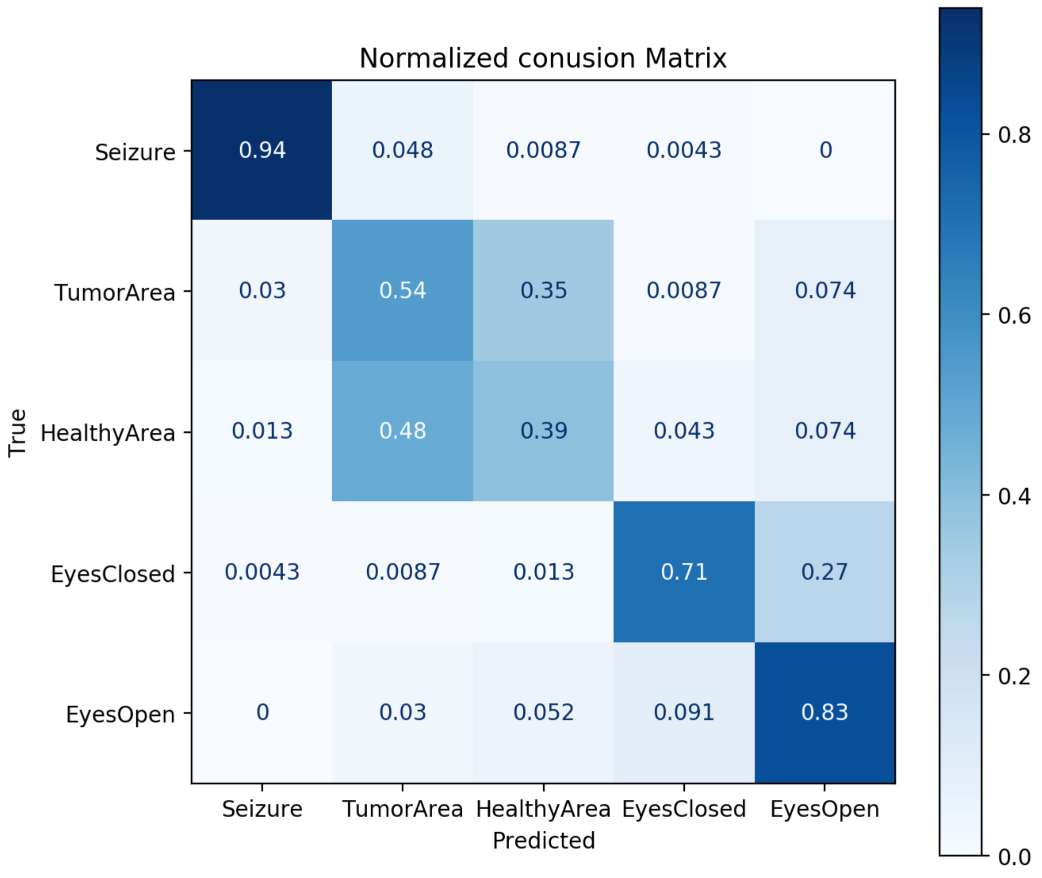

Figure 10: confusion matrix RNN unimpr

## 1.4 d Number of epochs = 10

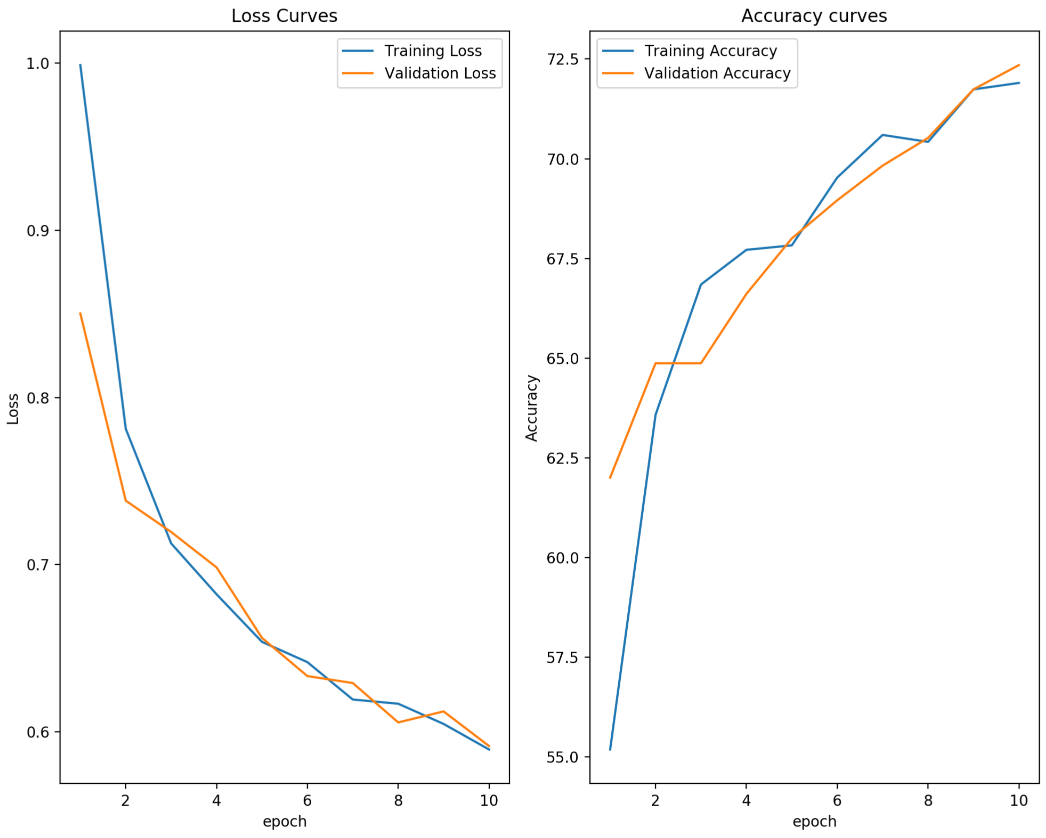

Figure 11: learning curves RNN impr

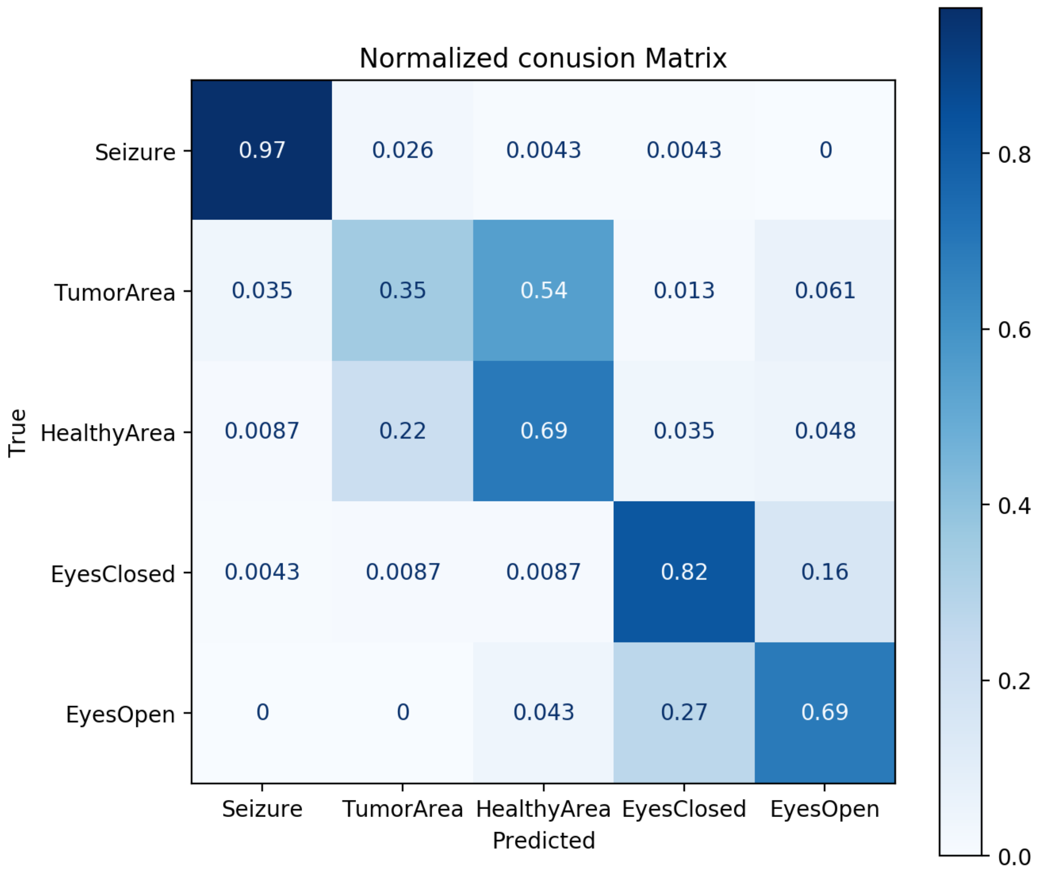

Figure 12: confusion matrix RNN impr

### 2.3 a Number of epochs = 30

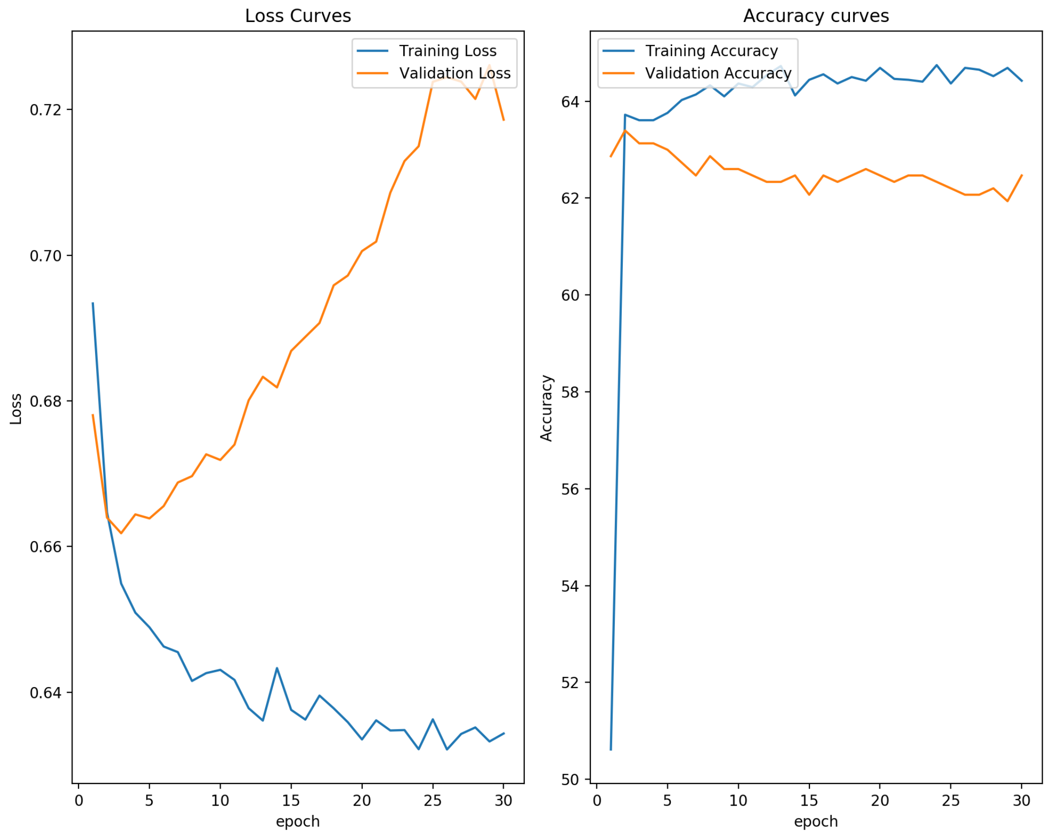

Figure 13: learning curves RNN var unimpr

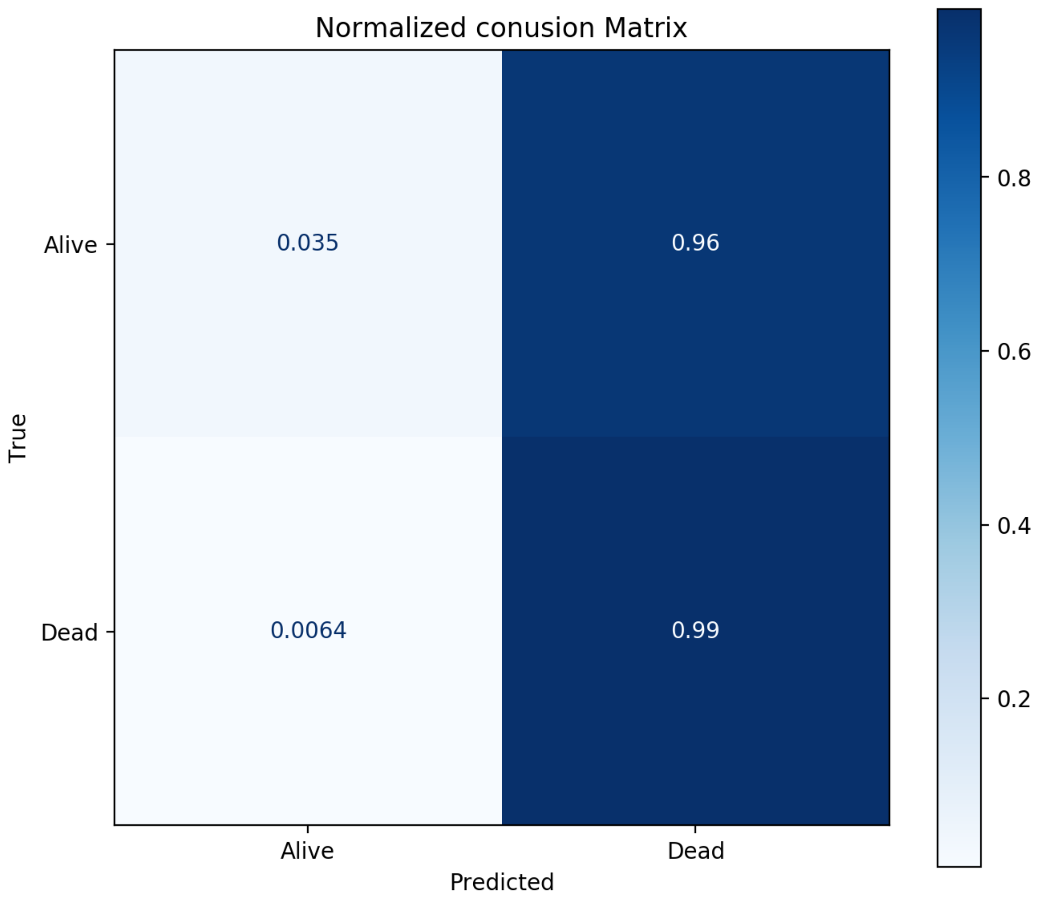

Figure 14: confusion matrix RNN var unimpr

## 2.3 d Number of epochs = 30

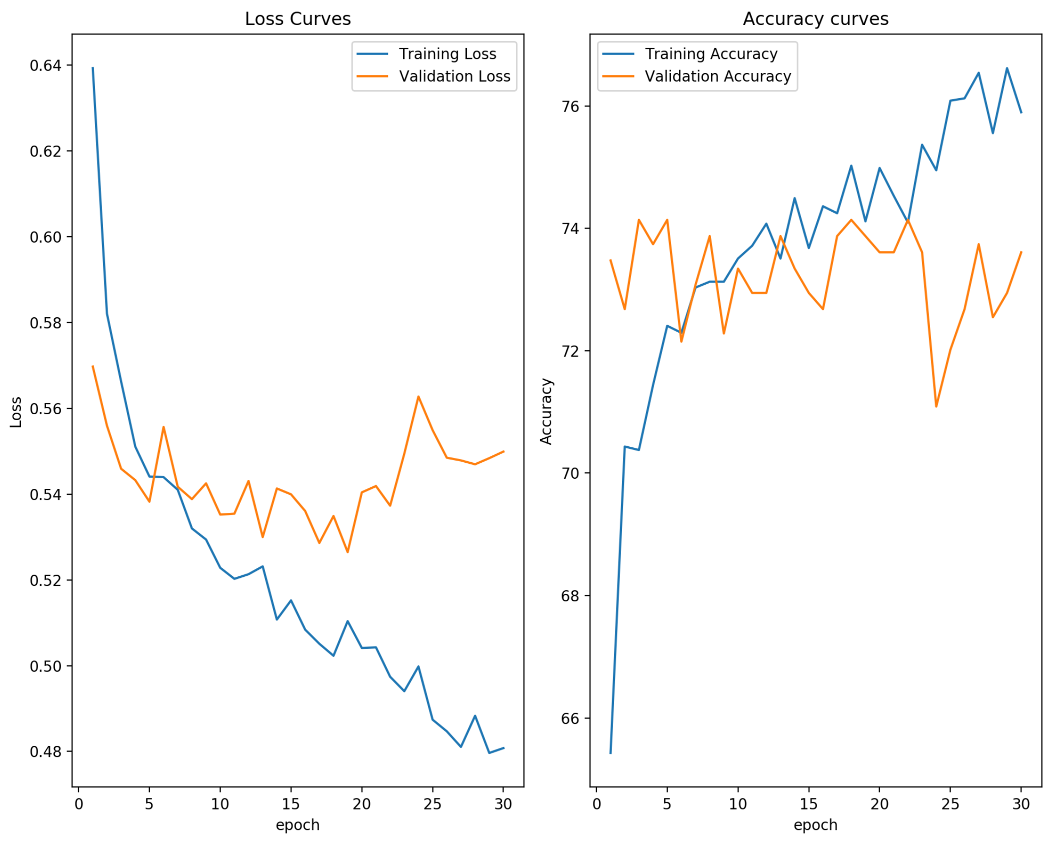

Figure 15: learning curves RNN var impr

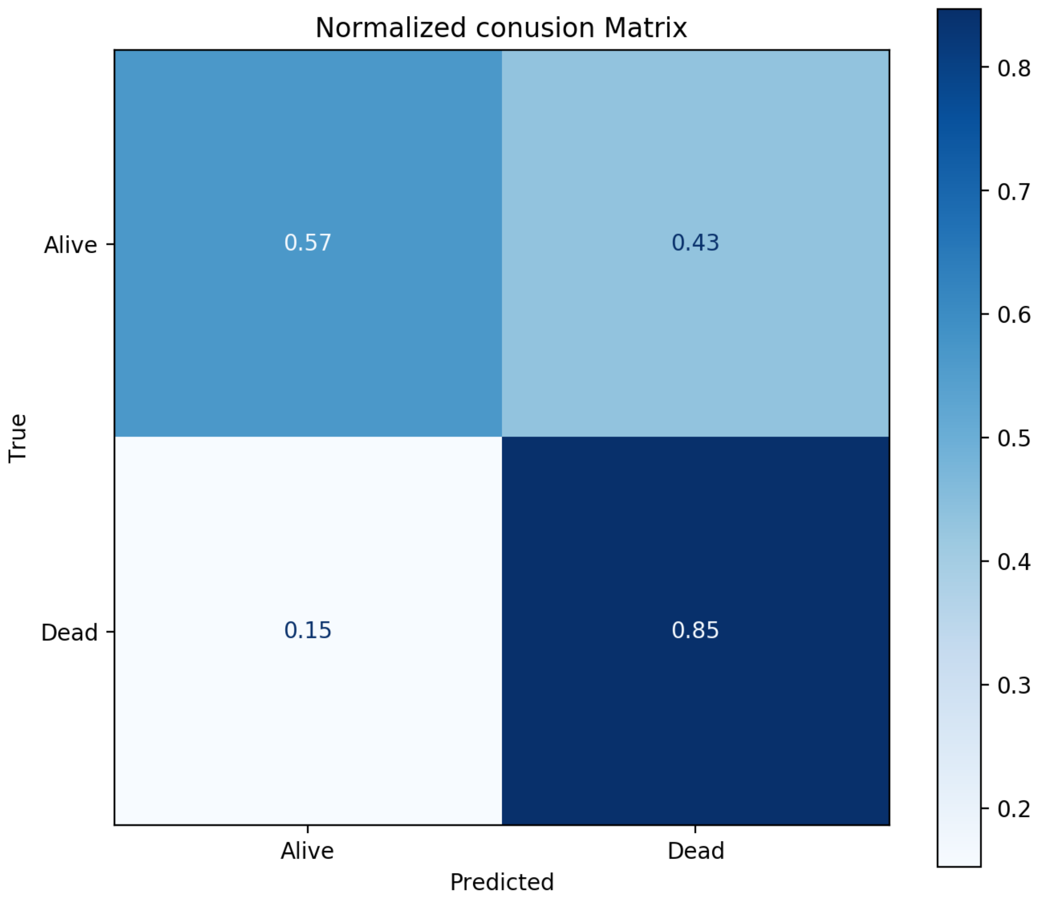

Figure 16: confusion matrix RNN var impr

### MyVariableRNN: Unimproved vs. Improved Version

**Baseline unimproved.** The original version of `MyVariableRNN` was a simple many-to-one GRU classifier:

This version assumes the last visit is the most informative and uses only 1 GRU with a small hidden size 16.

**Improved architecture.** The improved version keeps the same overall shape but the inside became more improved:

- **Stronger input projection with dropout.**
  - Before: $Linear(\rightarrow 32)$.

  - Now: $\texttt{Dropout}(0.6) \rightarrow Linear(d \rightarrow 128, bias = False) \rightarrow \texttt{Dropout}(0.0)$.

  This increases the embedding dimension from 32 to 128 and regularizes the input so the model does not overfit early visits.

- **Two GRU layers instead of one.**
  - Before: $1 \times GRU(32 \rightarrow 16)$.
  - Now: $GRU(128 \rightarrow 128)$ followed by another $GRU(128 \rightarrow 128)$.

  This increases both *depth* and *width*, letting the model capture longer temporal patterns in the visit sequences.

- **Proper masking for variable-length sequences.**
  - After unpacking with `pad_packed_sequence`, the improved code builds a mask so that padded positions are explicitly identified.

  This directly targets the homework requirement to handle variable-length EHR sequences.

- **Time-wise transformation before pooling.**
  - After the second GRU is unrolled, the model applies

    $$out = \tanh(Linear(128 \rightarrow 128)(GRUoutput))$$

    This lets the model reshape temporal features before attention pooling.

**Discussion about plots.** When plotting the learning curves (training vs. validation loss/accuracy), the unimproved shows:

- faster convergence but lower ceiling (small GRU, no attention),
- stronger dependence on the last visit.

The improved model:

- trains a bit slower (more parameters, two GRUs, attention),
- but reaches a higher validation accuracy / lower validation loss,
- and shows less overfitting thanks to the dropout layers and masking.

In the confusion matrix, the improved model tends to reduce errors on patients with longer or irregular visit histories, which is exactly where attention + masking provide extra signal.