Mice’s sleep stages classification with MLP

Authors: Edwin Haeffner, Arthur Junod

First expermiment

For our first experiment, we have to classify just two state of the mices, awake or asleep.

We started by concatenating the two `.csv` data files we receive so that we have only one array to manipulate. Then we normalized the data by putting the data to -1 for the asleep state and 1 for the awake state.

We then split the data into 3 to use it with shuffled k-fold validation.

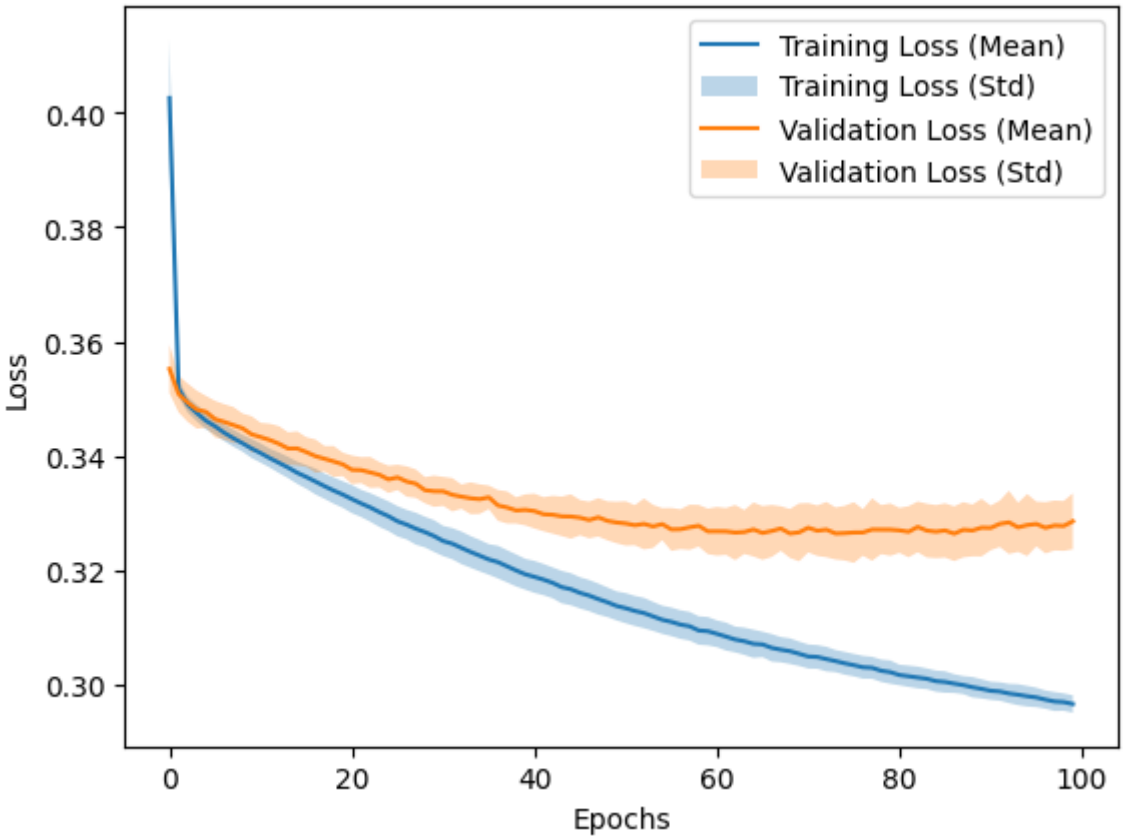
Our mlp uses 25 input neurons, 2 hidden layers of 32 neurons each and 1 output neuron. Their activation function is *tanh*.

```
mlp = keras.Sequential([
    layers.Input(25, ),
    layers.Dense(32, activation="tanh"),
    layers.Dense(32, activation="tanh"),
    layers.Dense(1, activation="tanh")
  ])
```

Its hyperparameters are : 0.001 for *learning rate* and 0.8 *momentum*. The loss function is *MSE*.

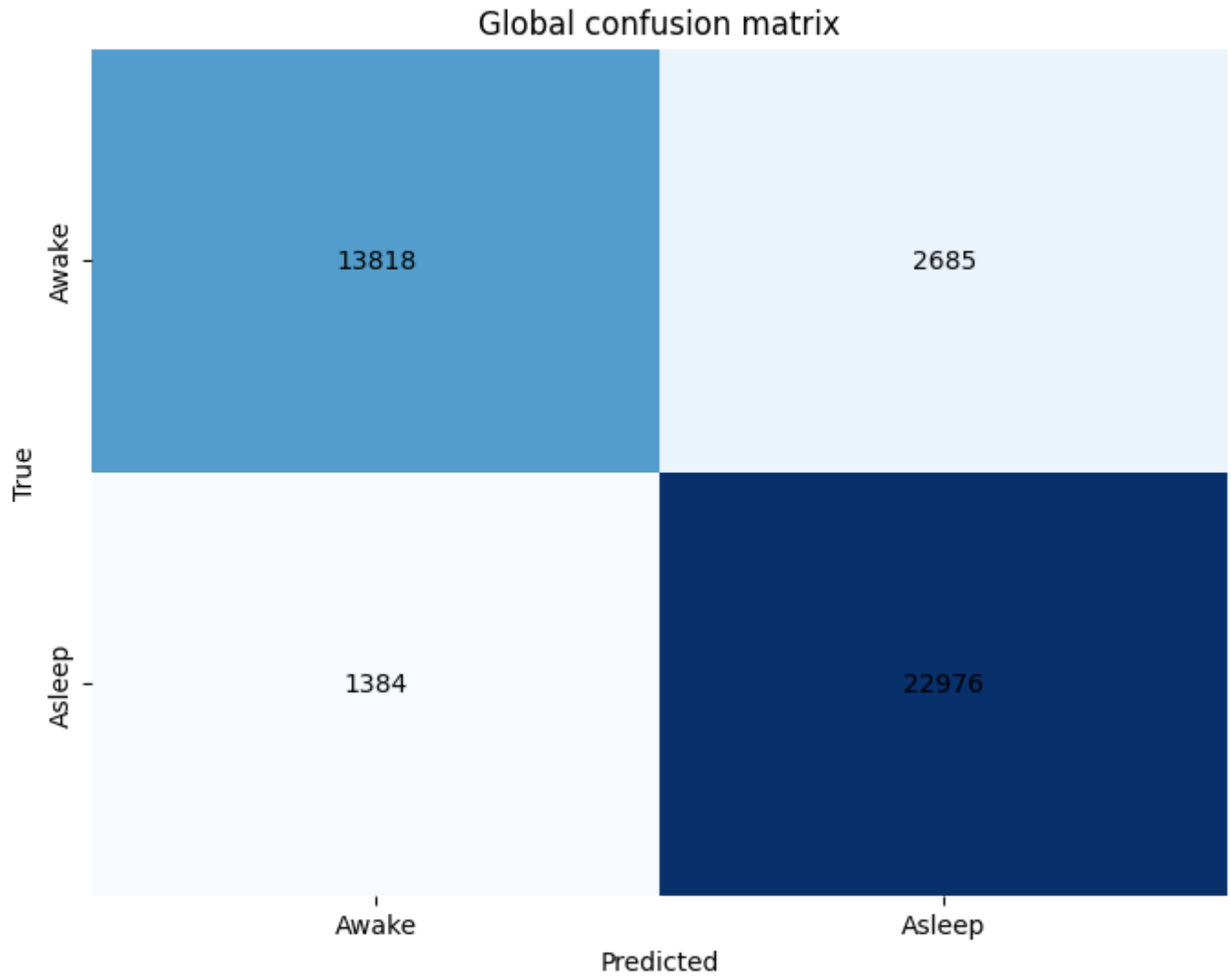
```
mlp.compile(
      optimizer=keras.optimizers.SGD(learning_rate=0.001, momentum=0.8),
      loss="mse",
  )
```

With 100 epochs we get :

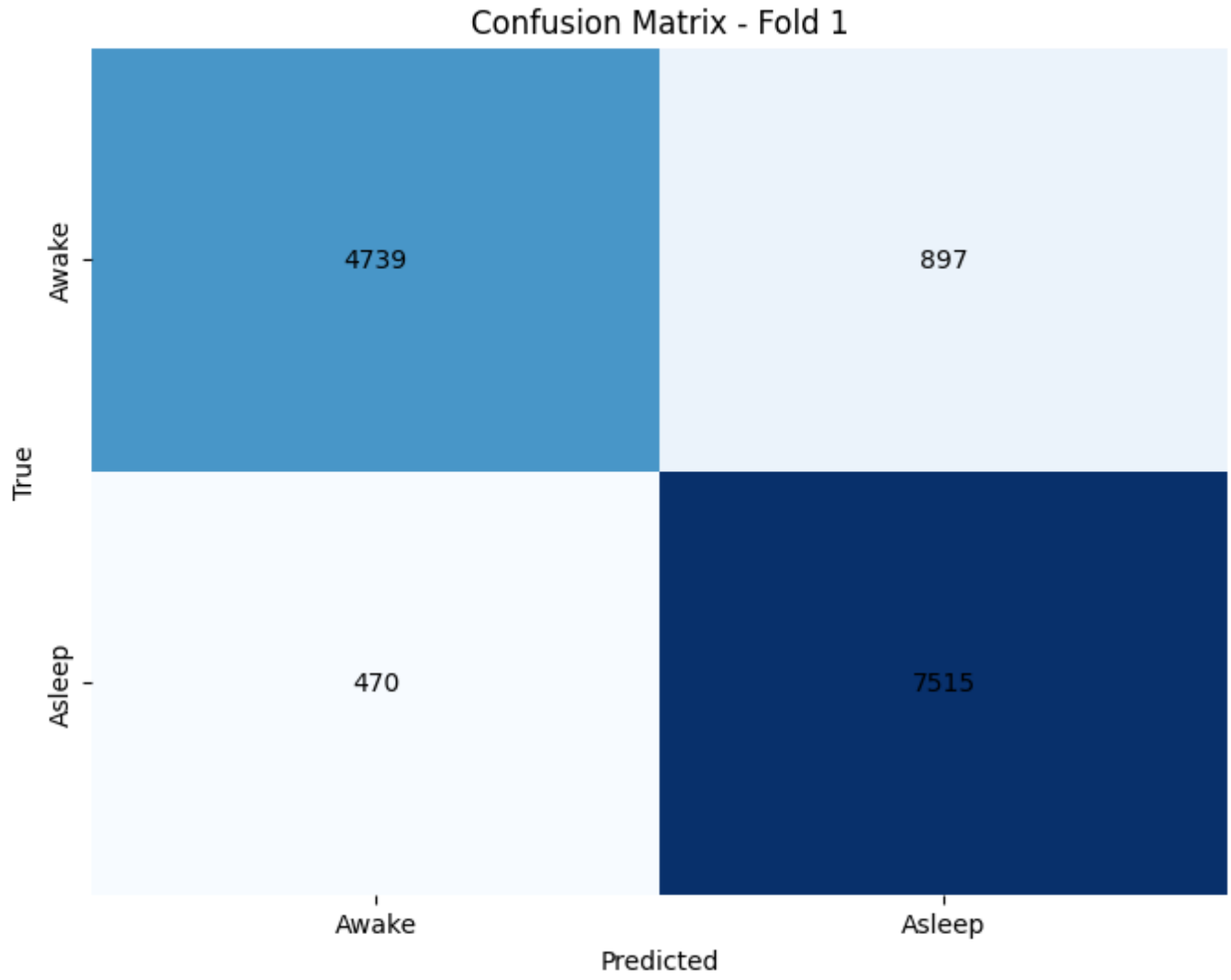


We see that our model is overfitting because the validation line stay around 0.33 of loss while the training loss gets lower and lower each epoch reaching around 0.3 at 100.

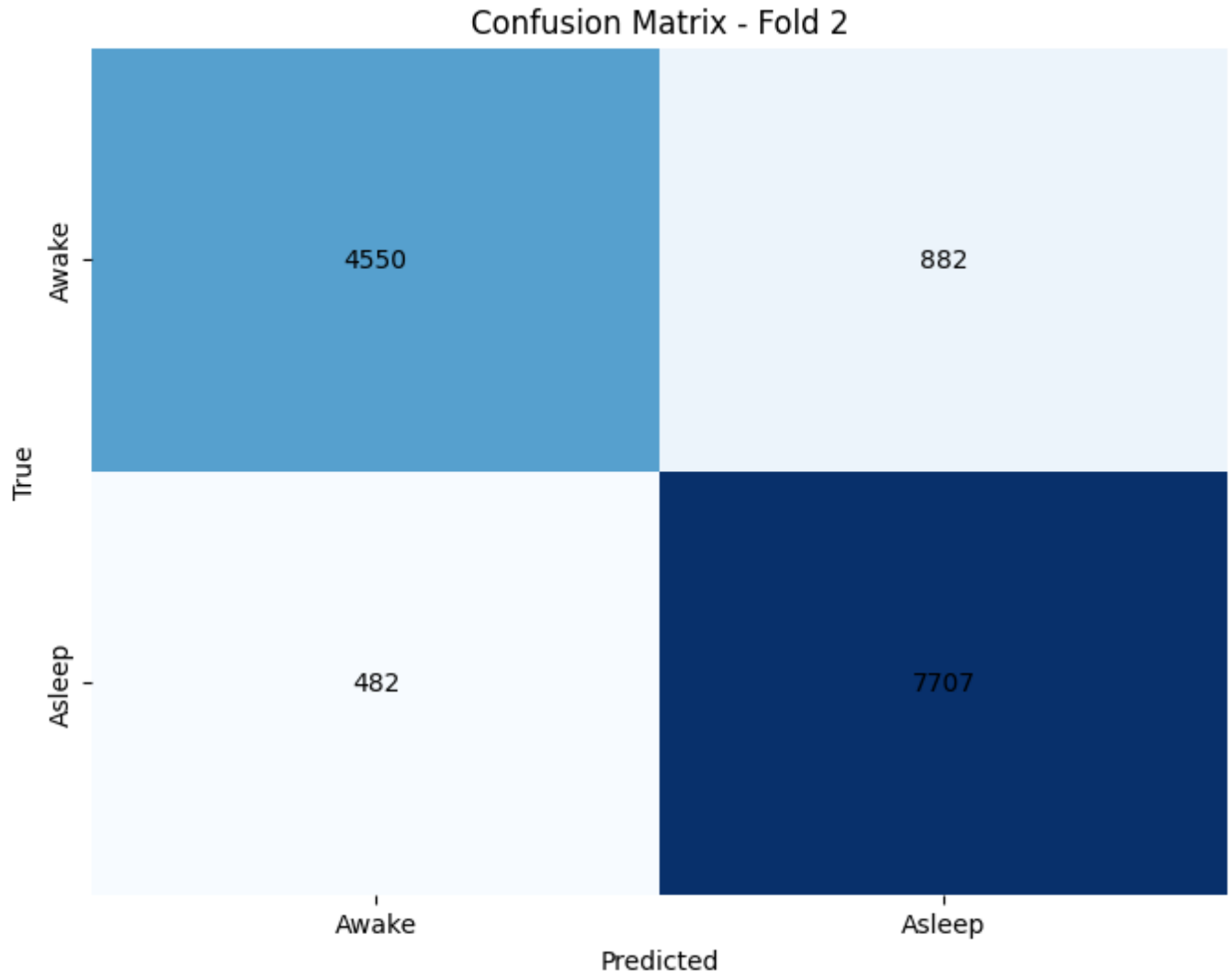
It is further confirmed with the confusions matrixes :



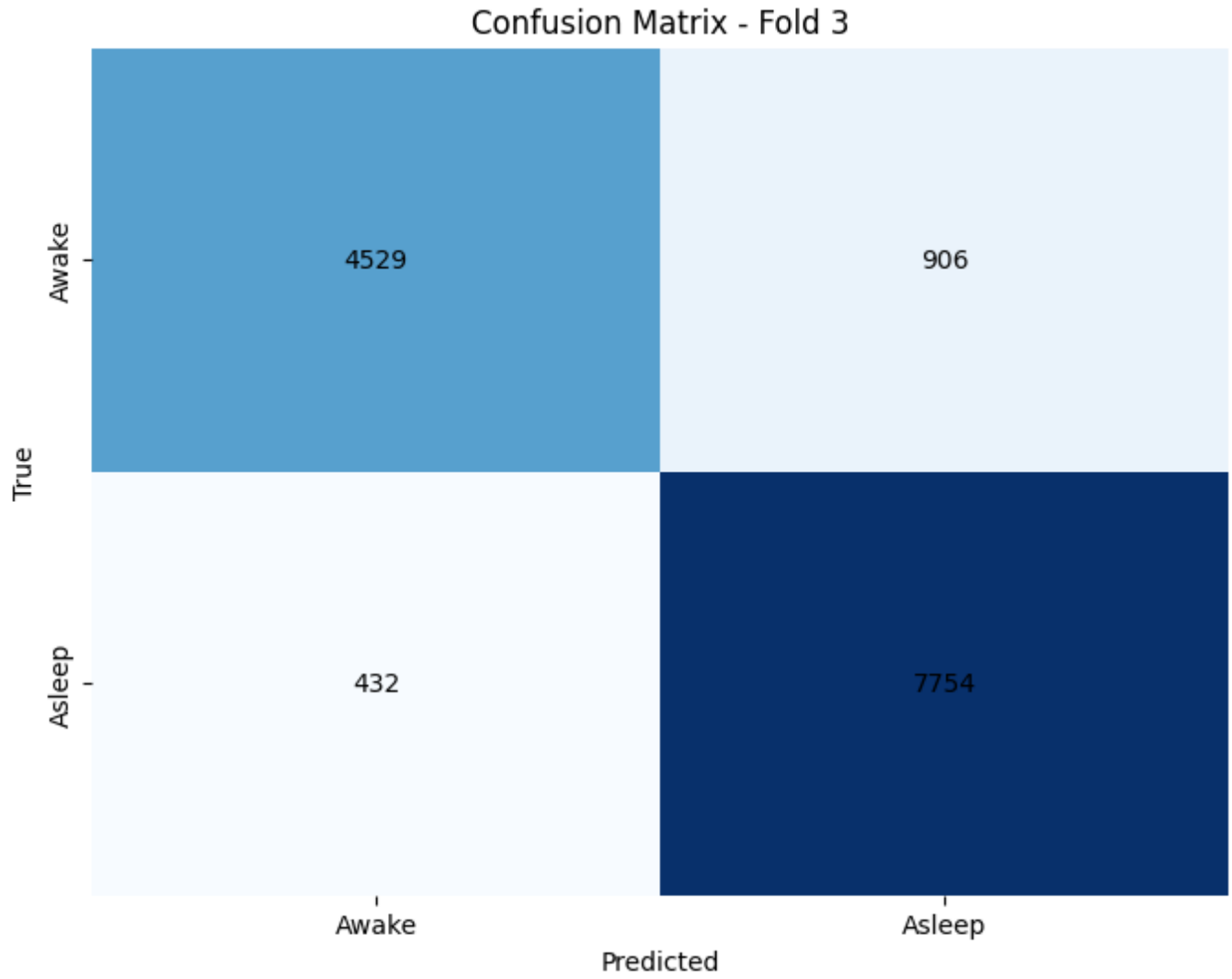
Mean F1 Score across all folds: 0.918636257236205



F1 Score - Fold 1: 0.9166310910532416



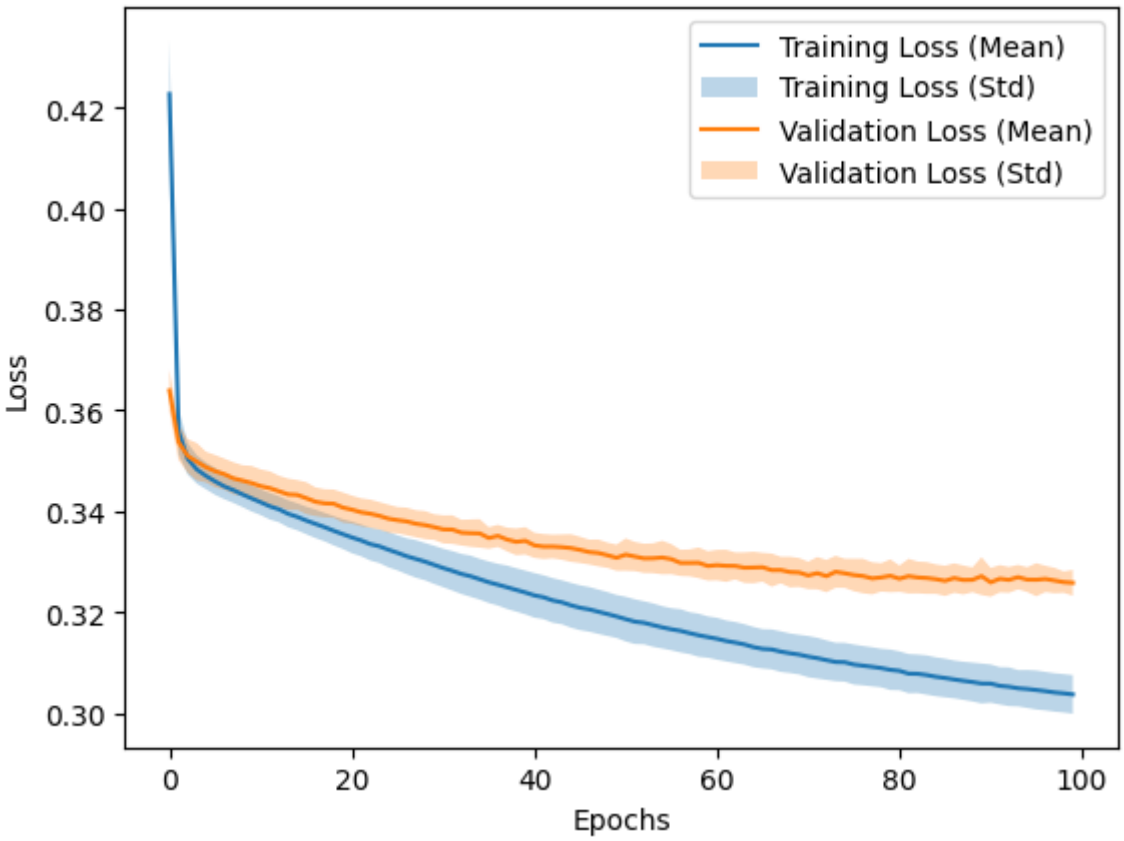
F1 Score - Fold 2: 0.9187030635355822



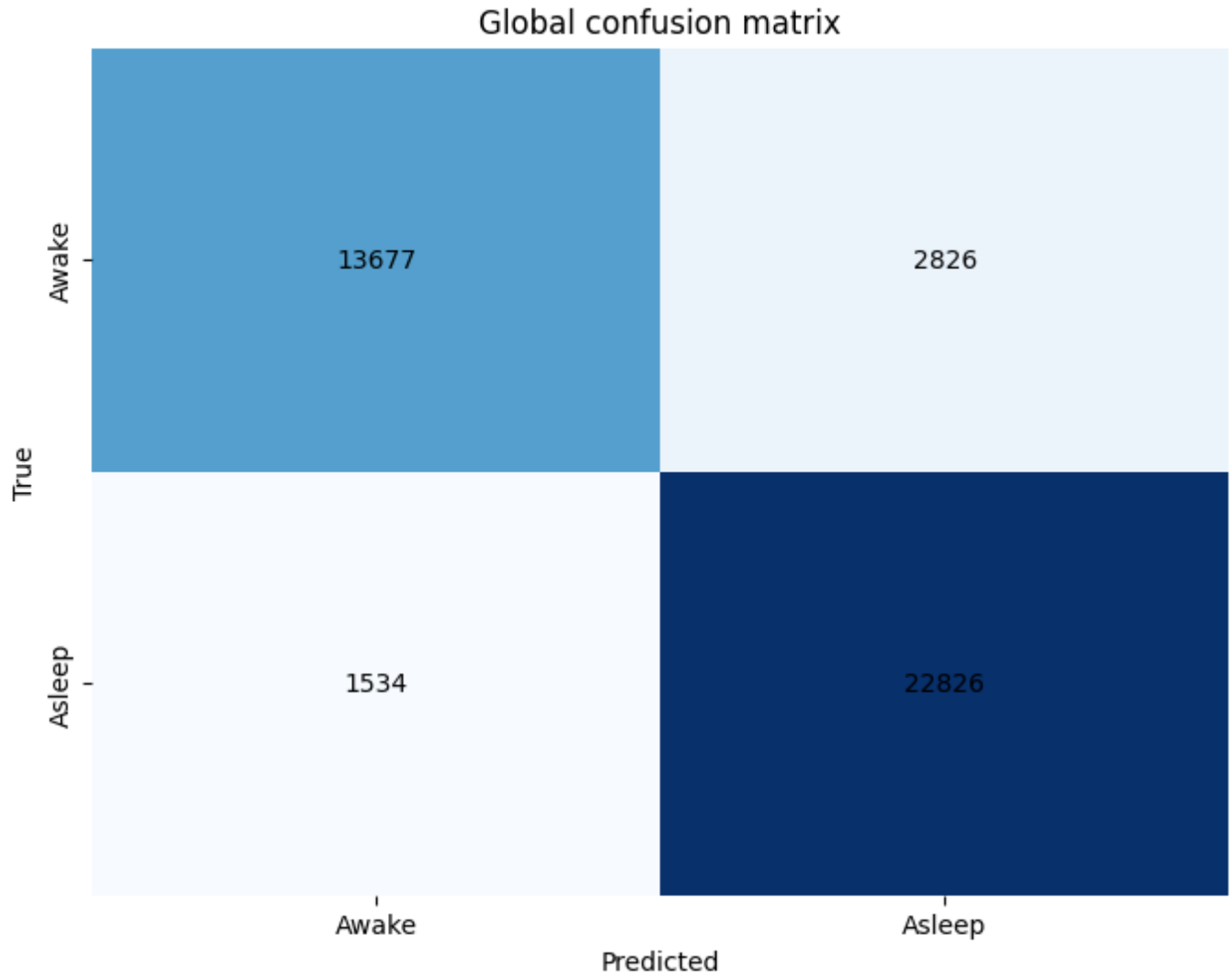
F1 Score - Fold 3: 0.920574617119791

First attempt to reduce overfitting: Less neurons

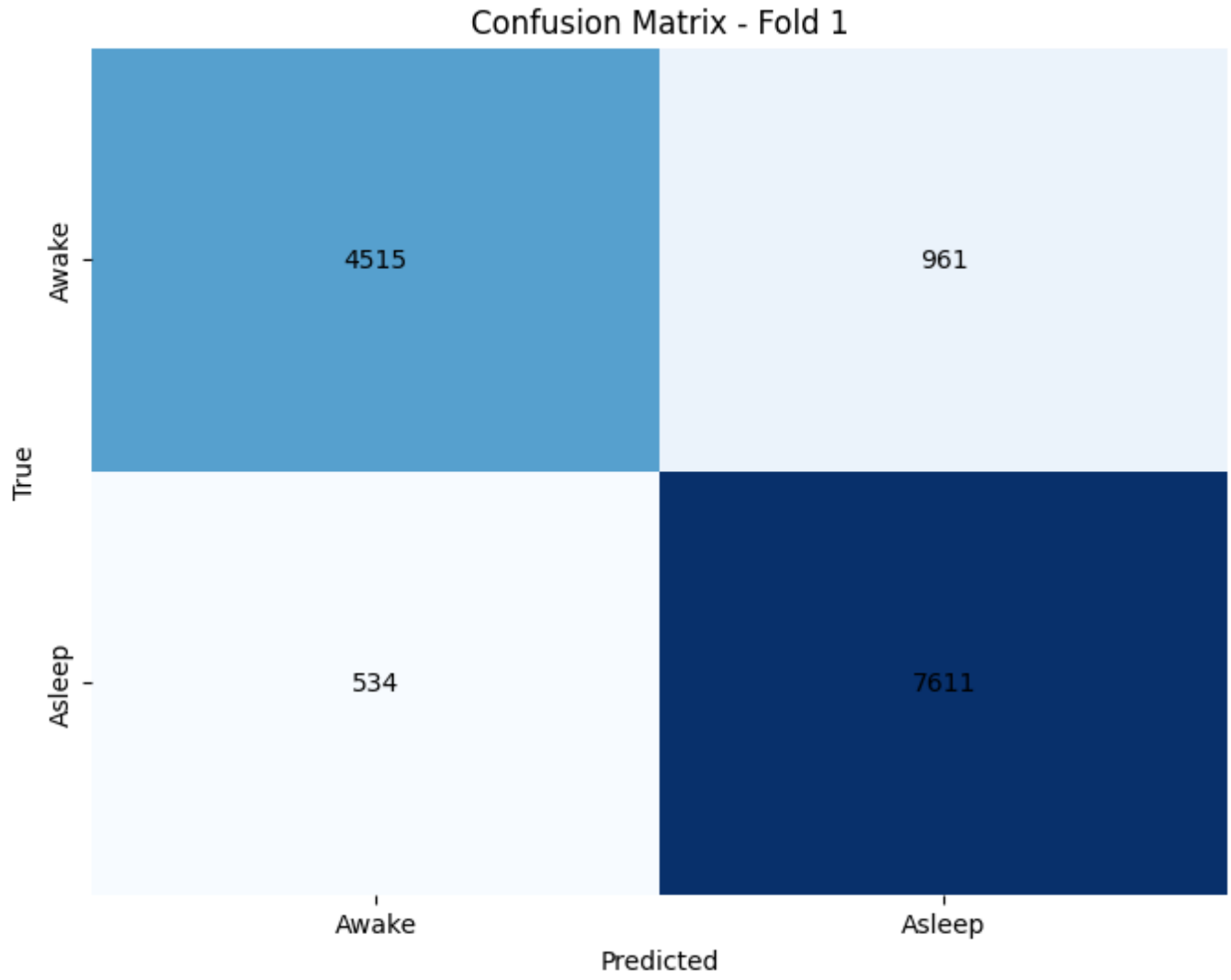
Reducing the number of hidden neurons to 22:



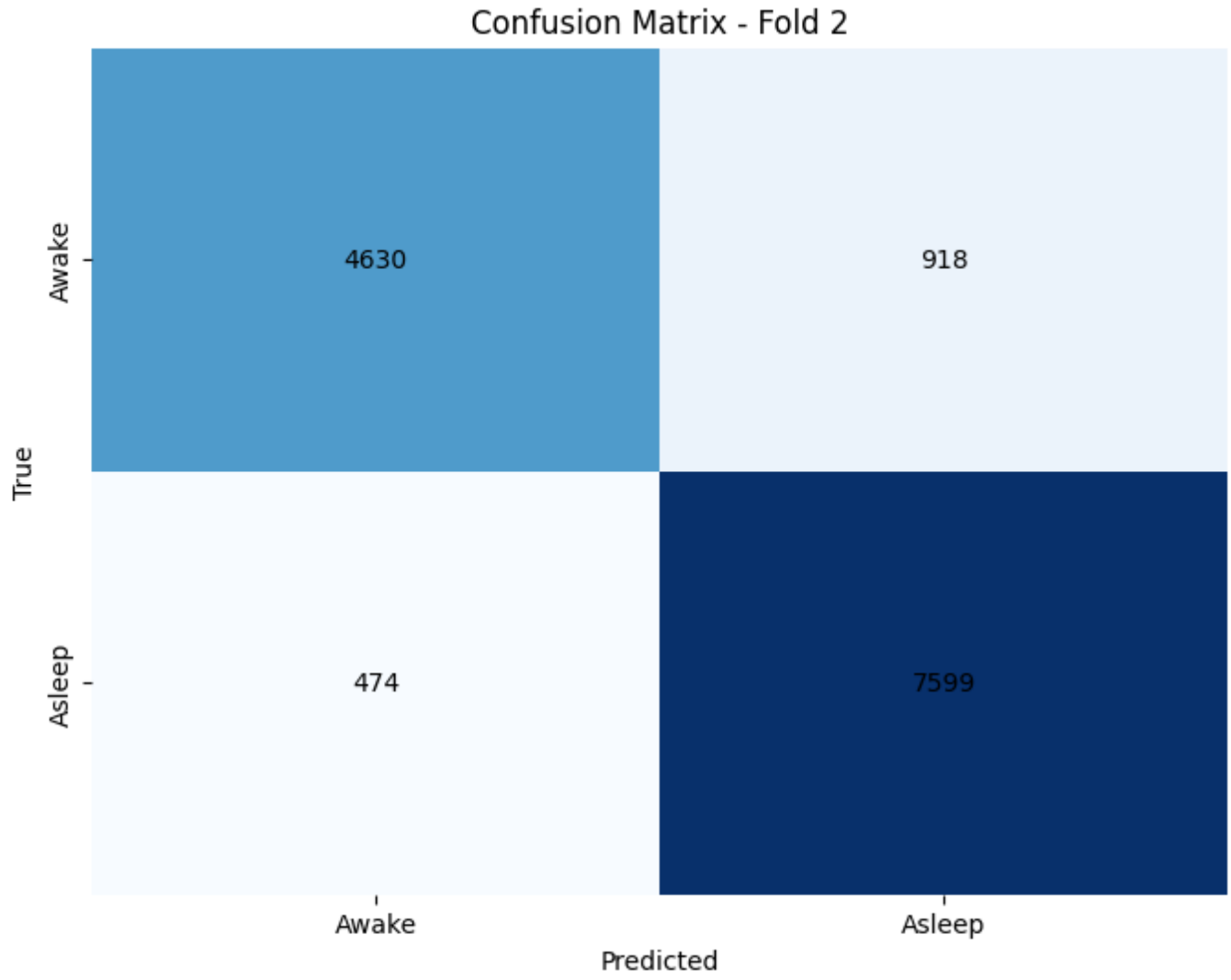
And the confusions matrixes



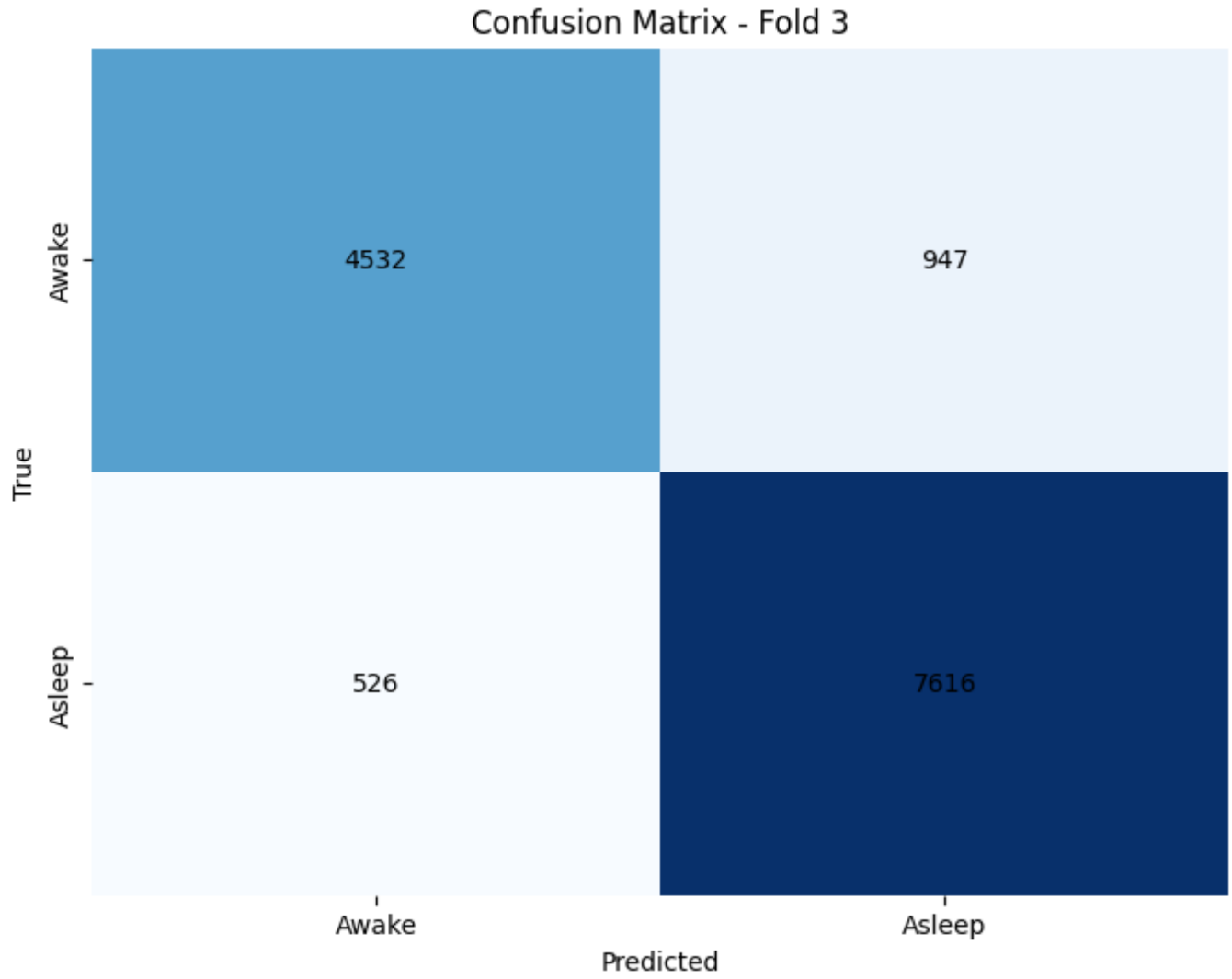
Mean F1 Score across all folds: 0.9128289728185744



F1 Score - Fold 1: 0.9105700783633427



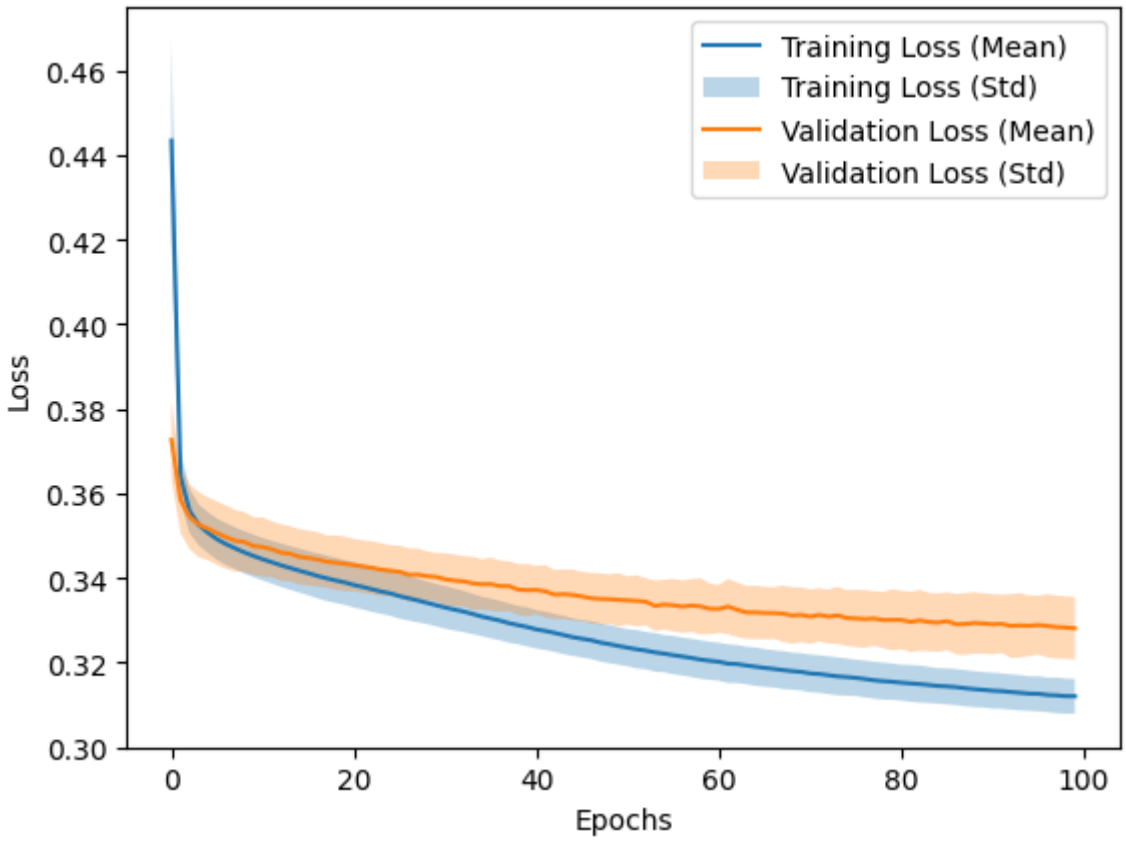
F1 Score - Fold 2: 0.9160940325497288



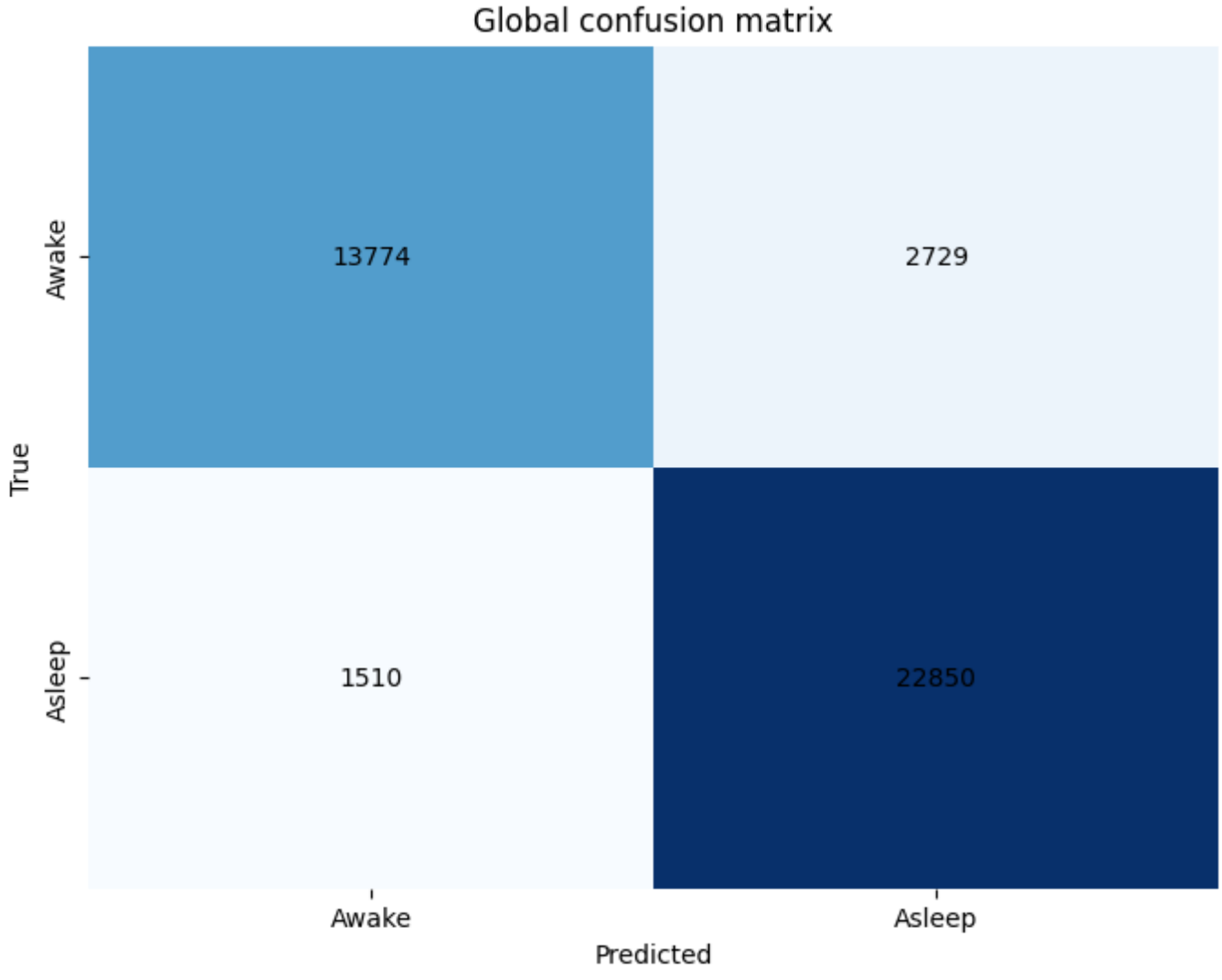
F1 Score - Fold 3: 0.9118228075426519

On the plot we can see that while the model performs worse than when we used 32 neurons, the difference between the validation and the training is smaller so it overfits less.

We can try with even less neurons (12):

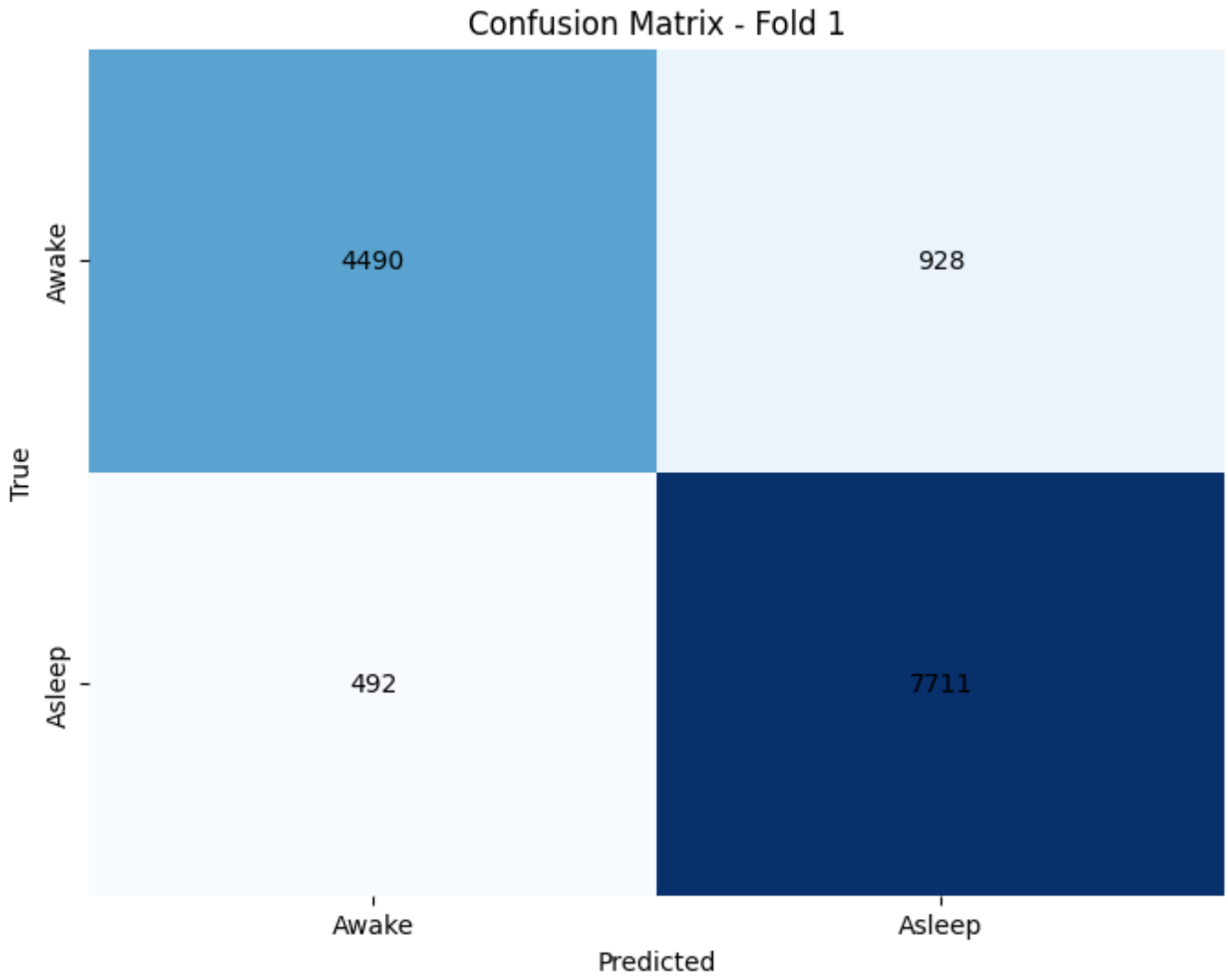


And confusions matrixes

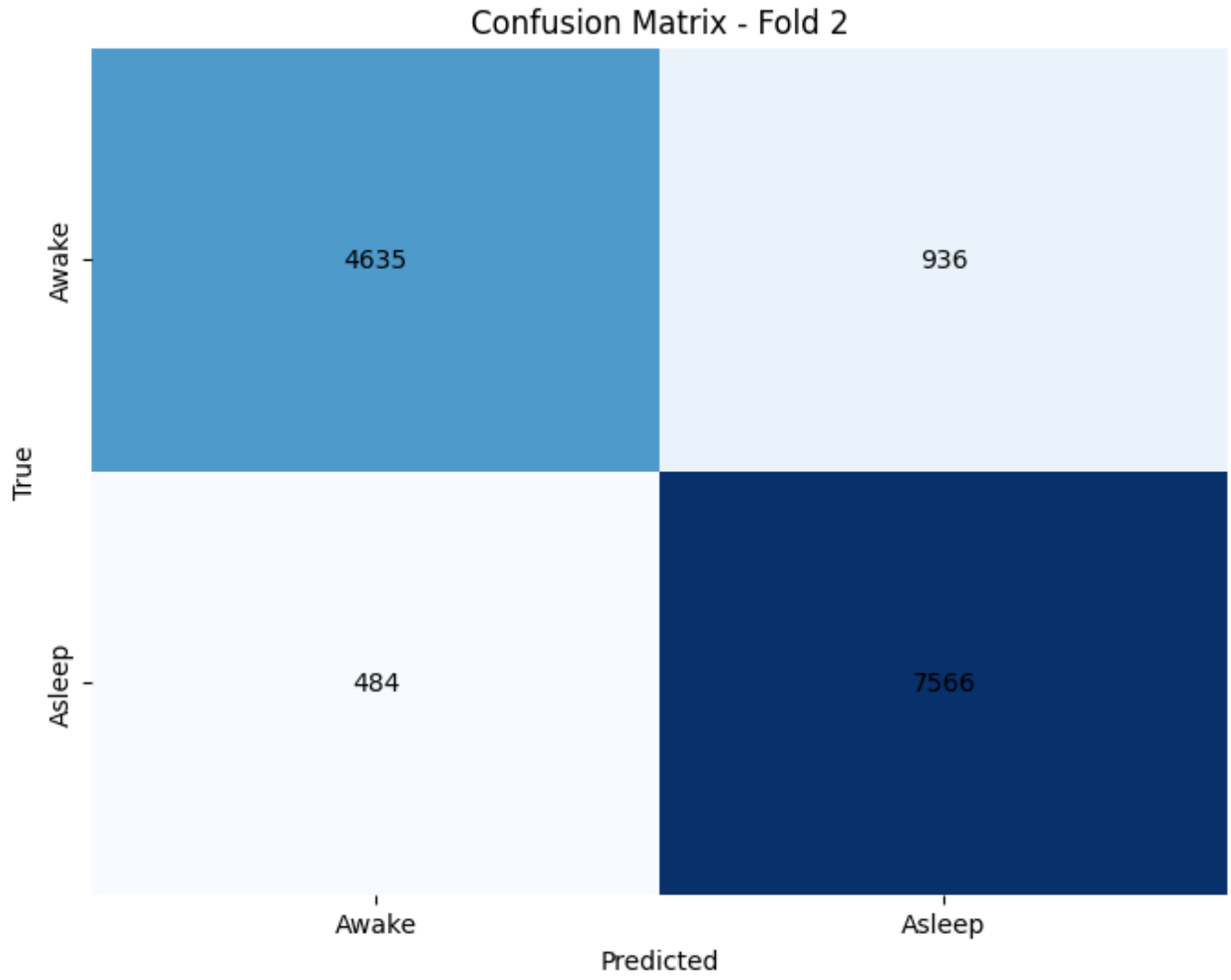


/

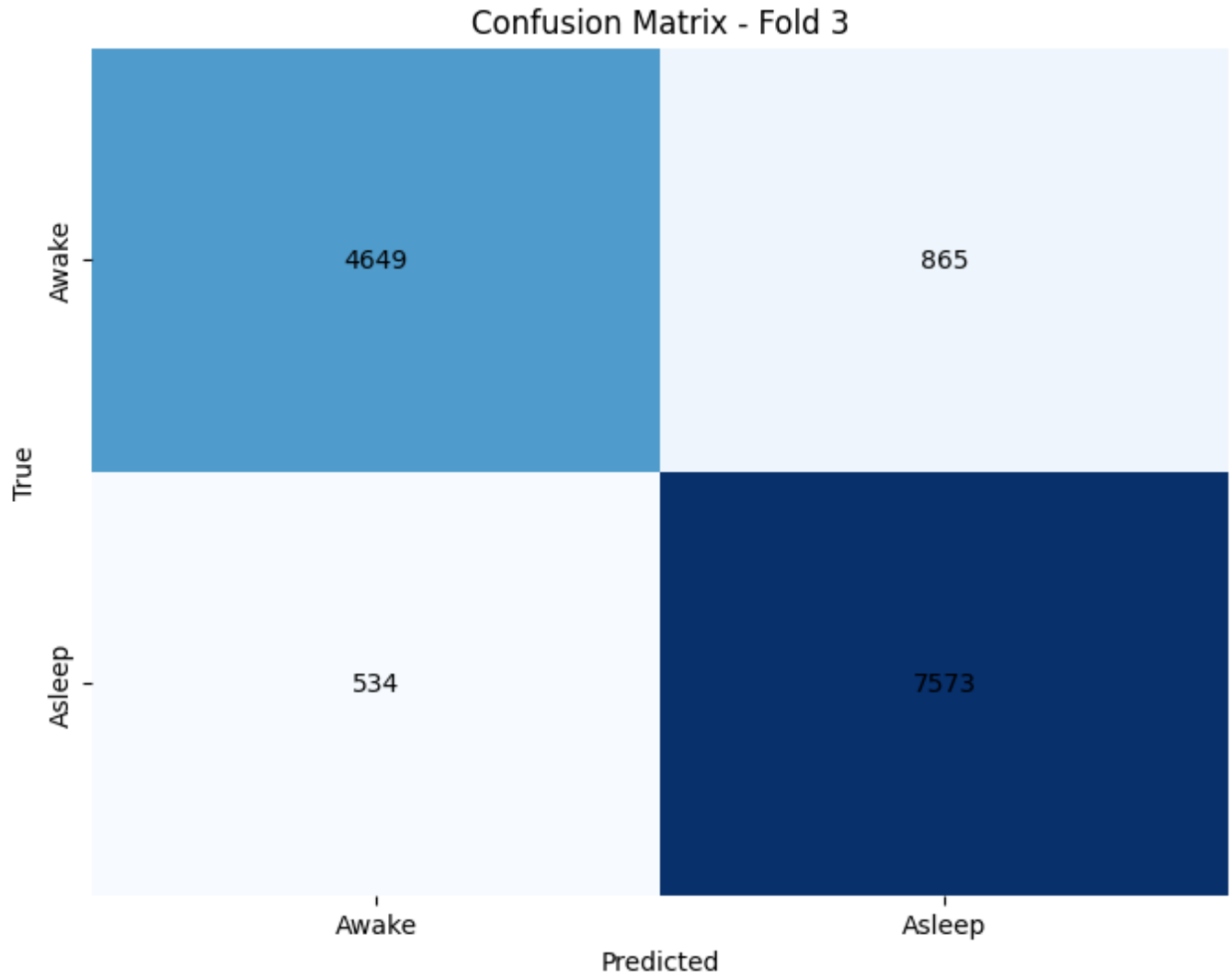
Mean F1 Score across all folds: 0.9151131560524602



F1 Score - Fold 1: 0.9156869730435816



F1 Score - Fold 2: 0.9142097631706138



F1 Score - Fold 3: 0.9154427319431853

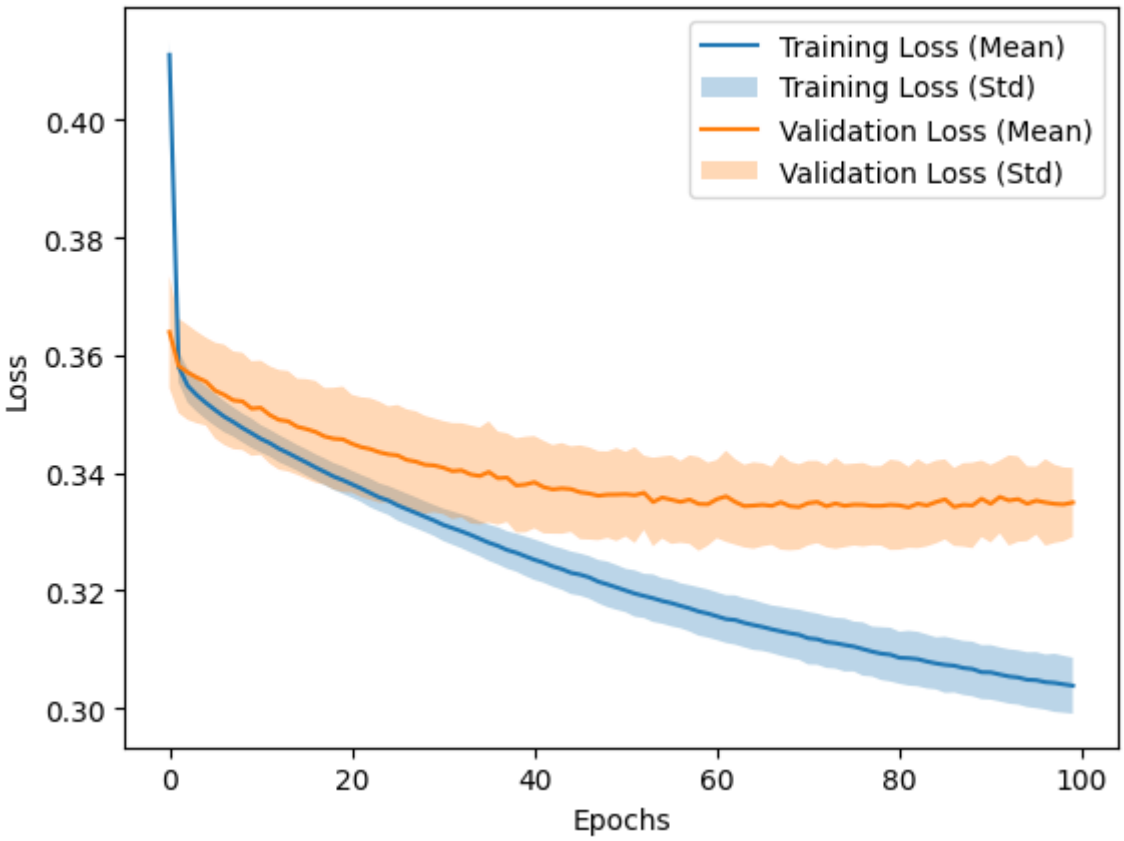
We can see that, once again, the difference between the validation and the training is even smaller but the model performs worse once again.

Second attempt to reduce overfitting: L2 regularization

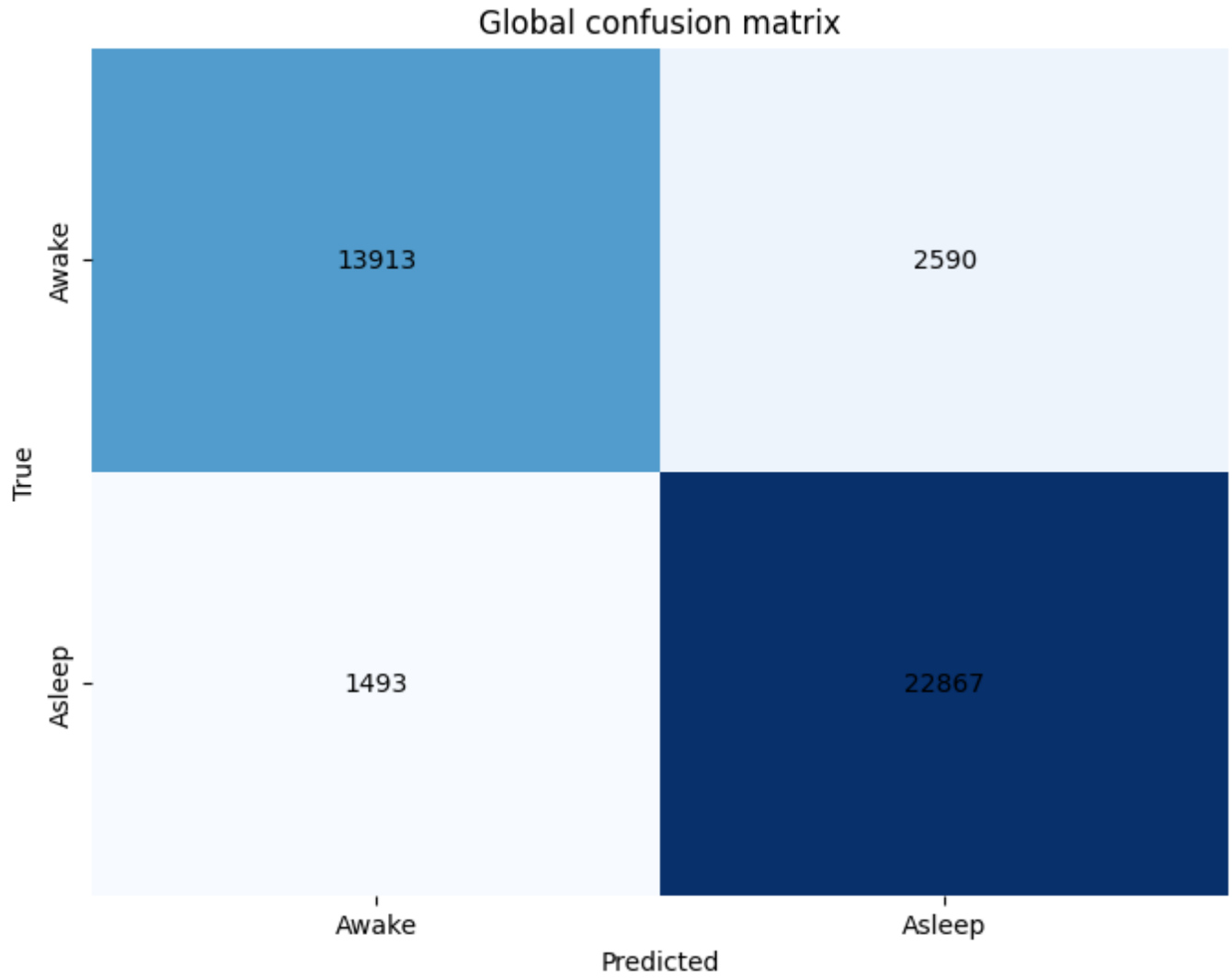
Another way to reduce overfitting is to use the L2 regularization. To use it with keras you have to add these lines:

```
from keras import regulizers

mlp = keras.Sequential([
    layers.Input(25, ),
    layers.Dense(32, kernel_regularizer=regularizers.l2(0.0001),
activation="tanh"),
    layers.Dense(32, kernel_regularizer=regularizers.l2(0.0001),
activation="tanh"),
    layers.Dense(1, activation="tanh")
  ])
```

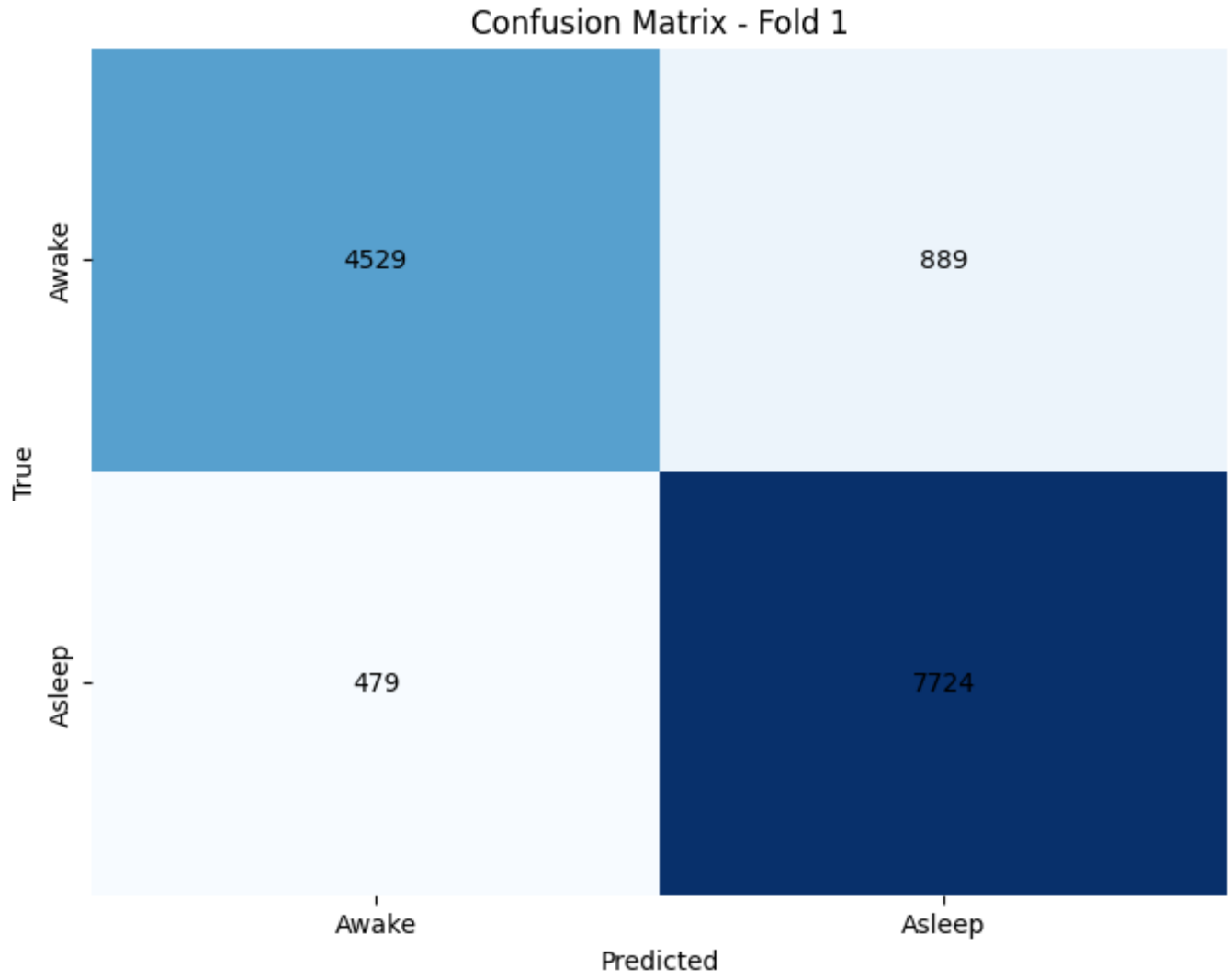
So we go back to our 32 neurons and 100 epochs but this time with L2 regularization



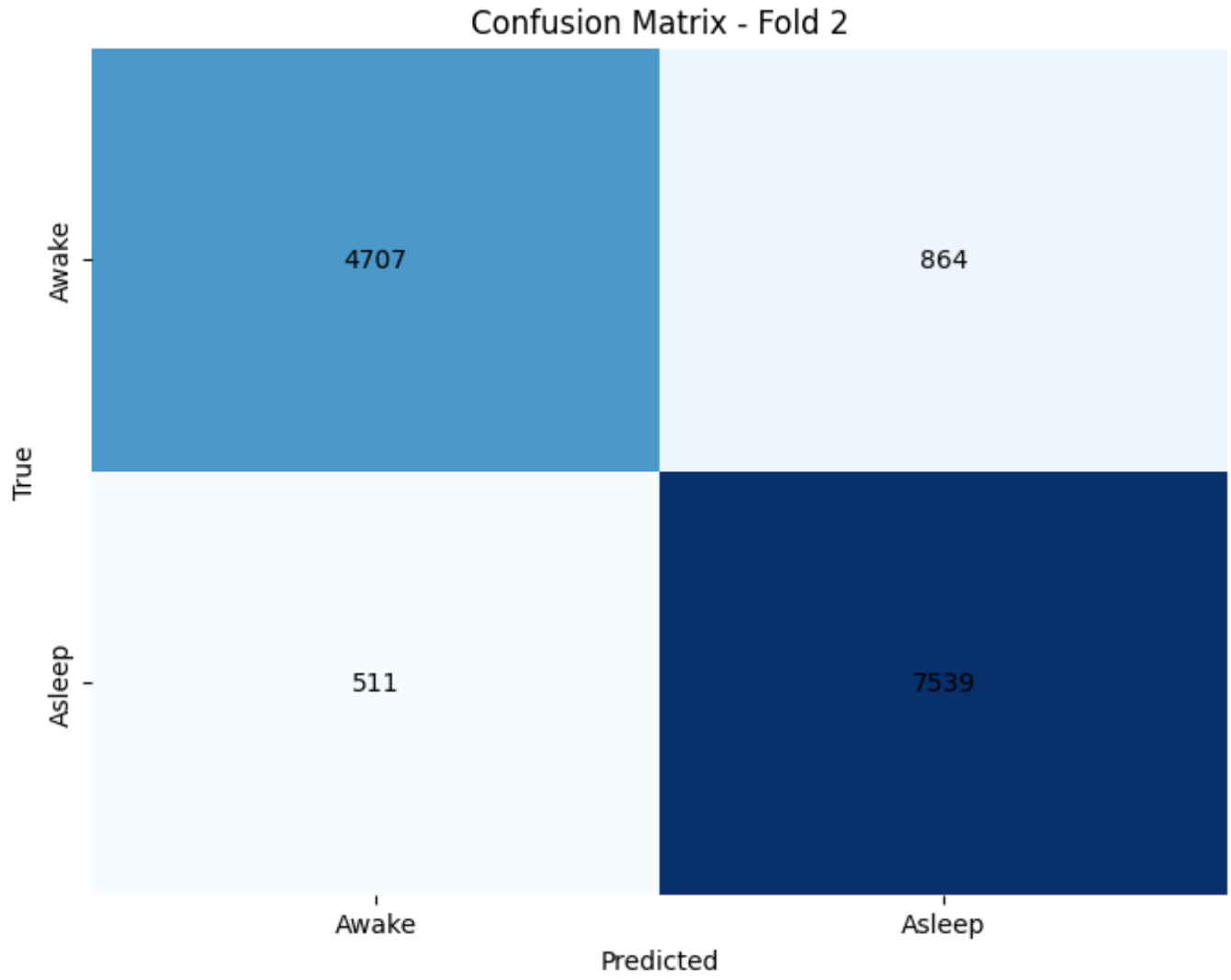
And we see that it, sadly, didn't really allowed us to find a better model.



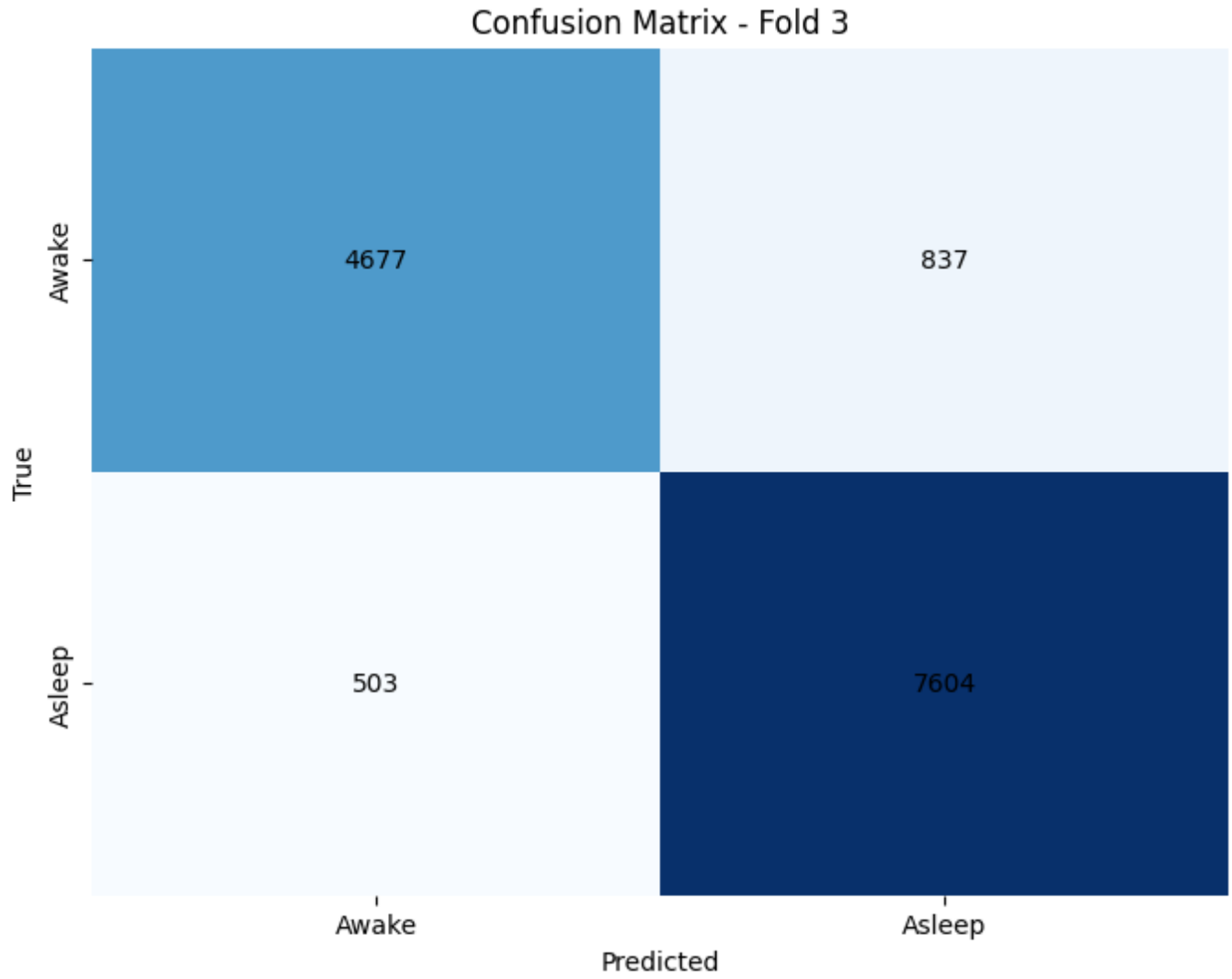
Mean F1 Score across all folds: 0.9180336558628343



F1 Score - Fold 1: 0.9186489058039962



F1 Score - Fold 2: 0.916428614842278



F1 Score - Fold 3: 0.9190234469422288

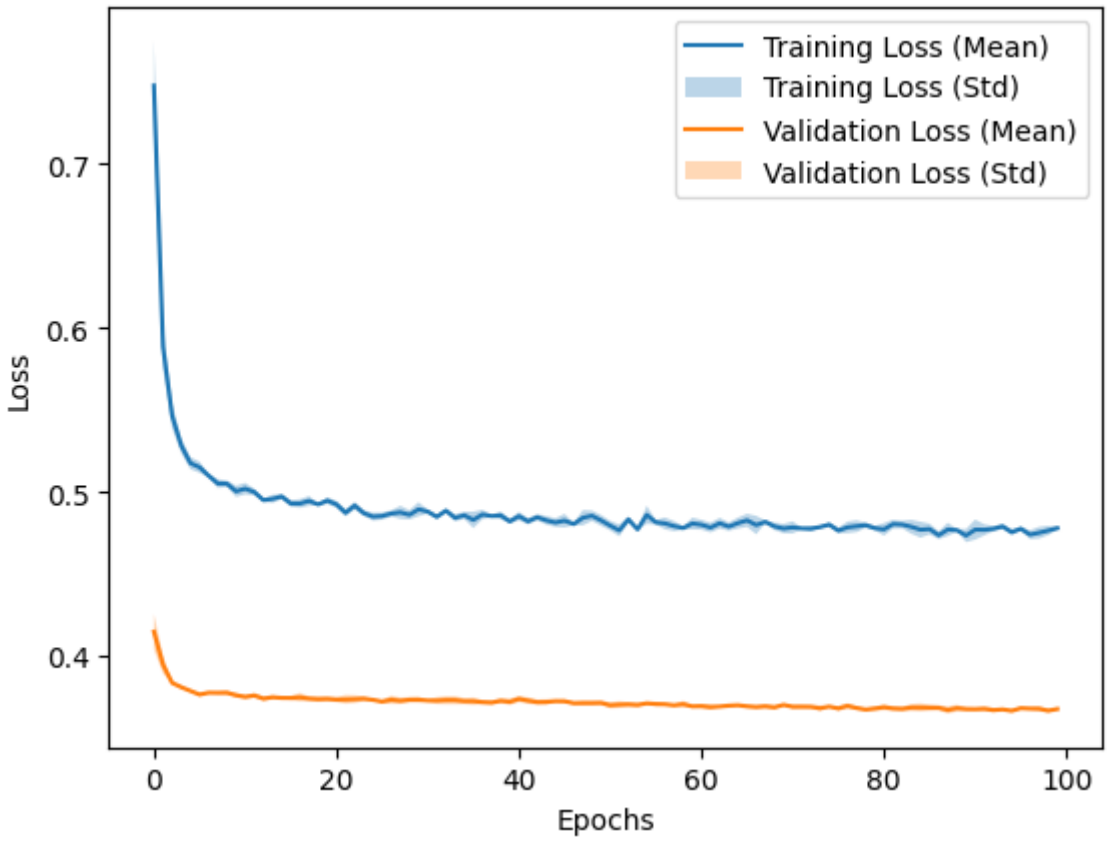
Third attempt to reduce overfitting: Dropout regularization

We also found another way to reduce overfitting by using Dropout regularization which are layers (in keras) that ignore random neurons.

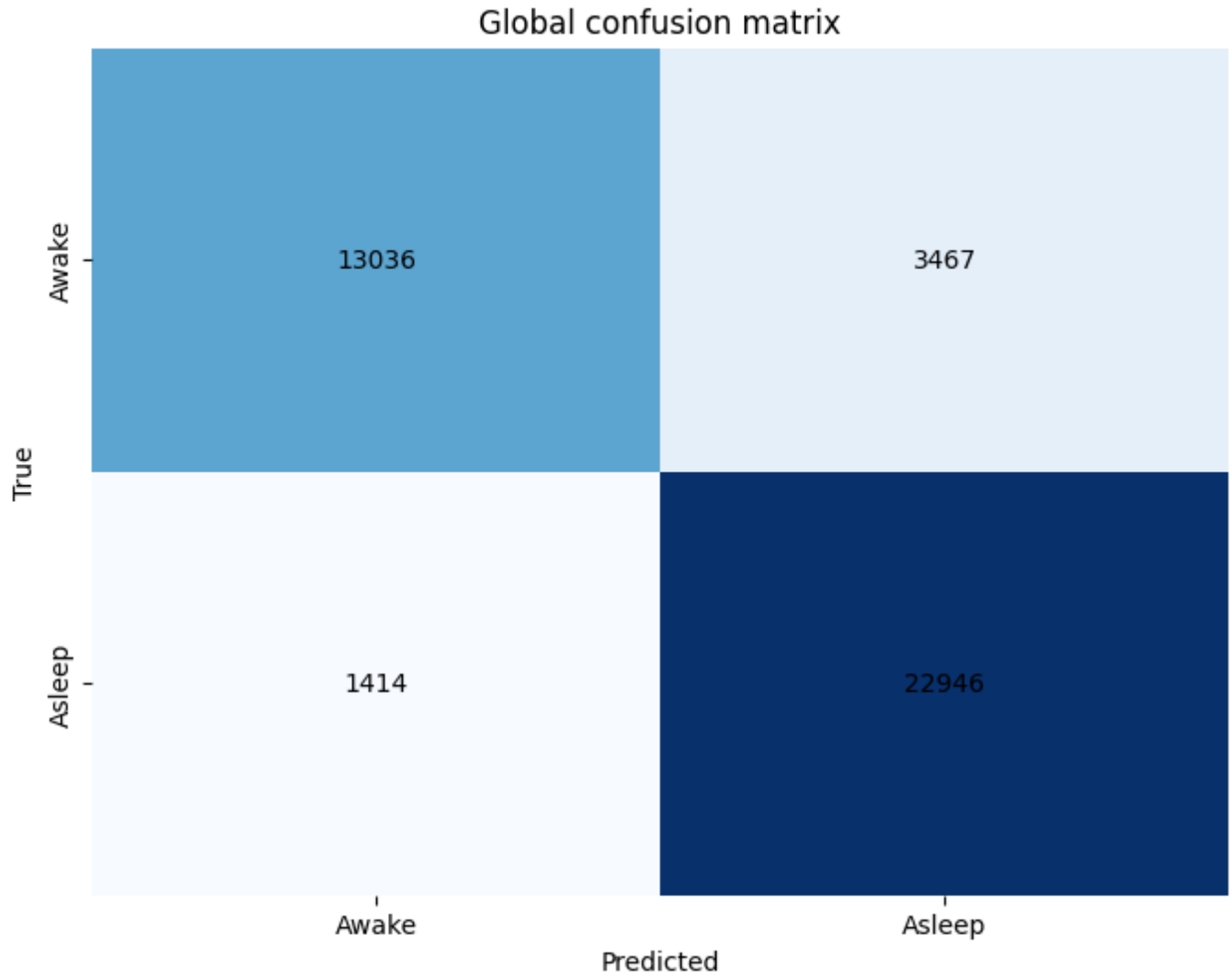
We just have to add `layers.Dropout(0.5)` between our `layers.Dense(...)`.

```
mlp = keras.Sequential([
    layers.Input(25, ),
    layers.Dropout(0.5),
    layers.Dense(32, kernel_regularizer=regularizers.l2(0.0001),
activation="tanh"),
    layers.Dropout(0.5),
    layers.Dense(32, kernel_regularizer=regularizers.l2(0.0001),
activation="tanh"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="tanh")
  ])
```

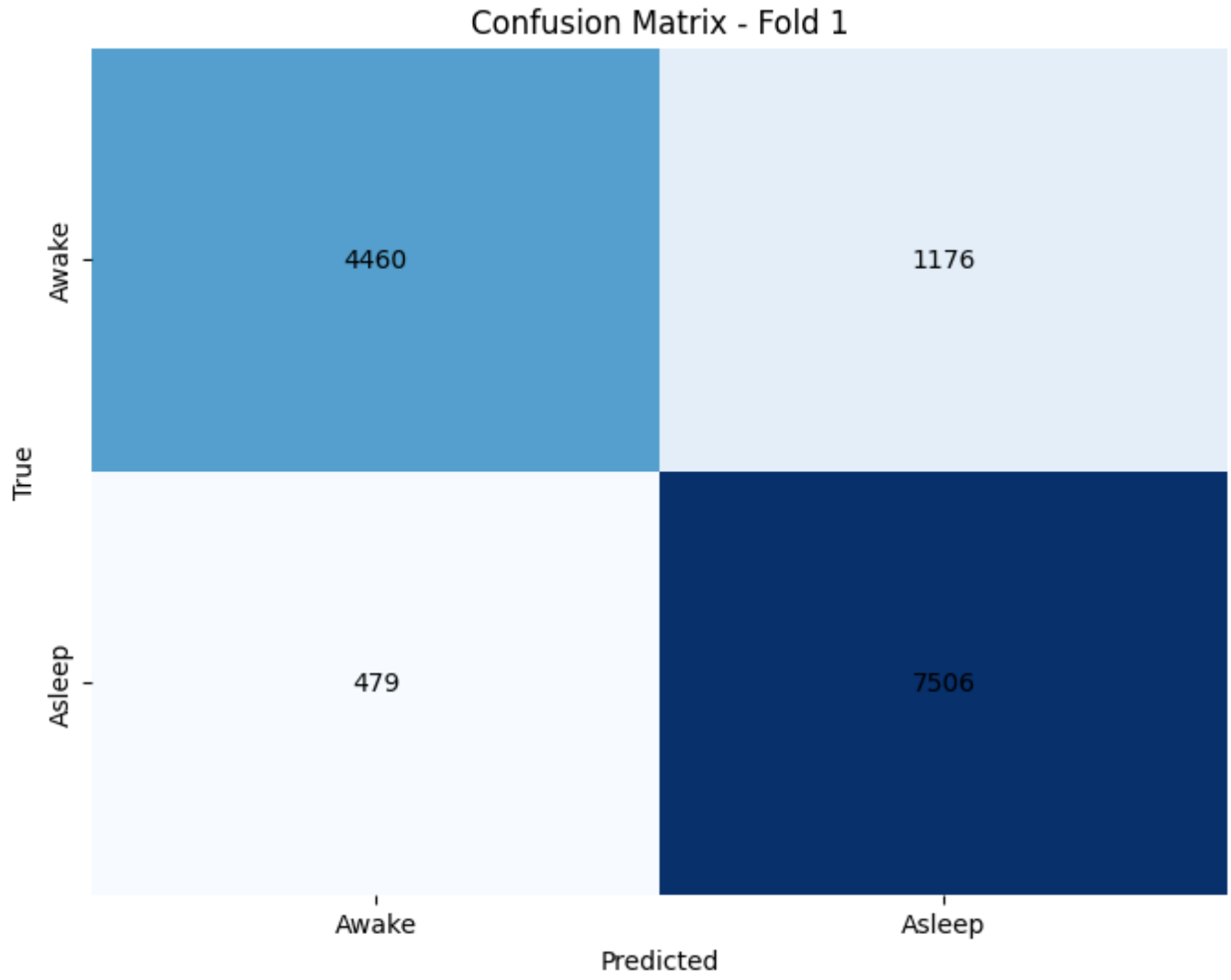
It didn't help us in our case. We can see below that the model performs significantly worse.



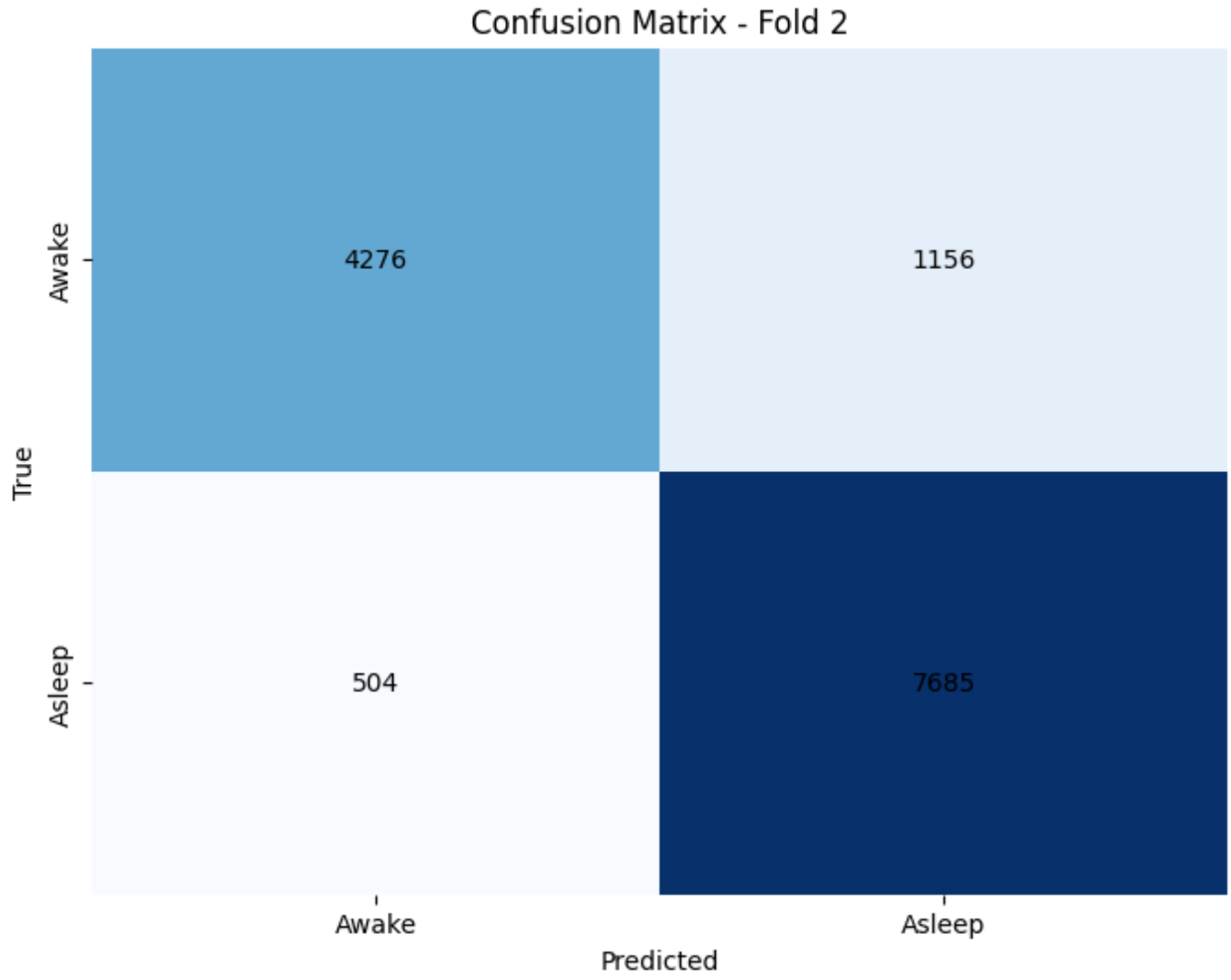
Confusion matrixes



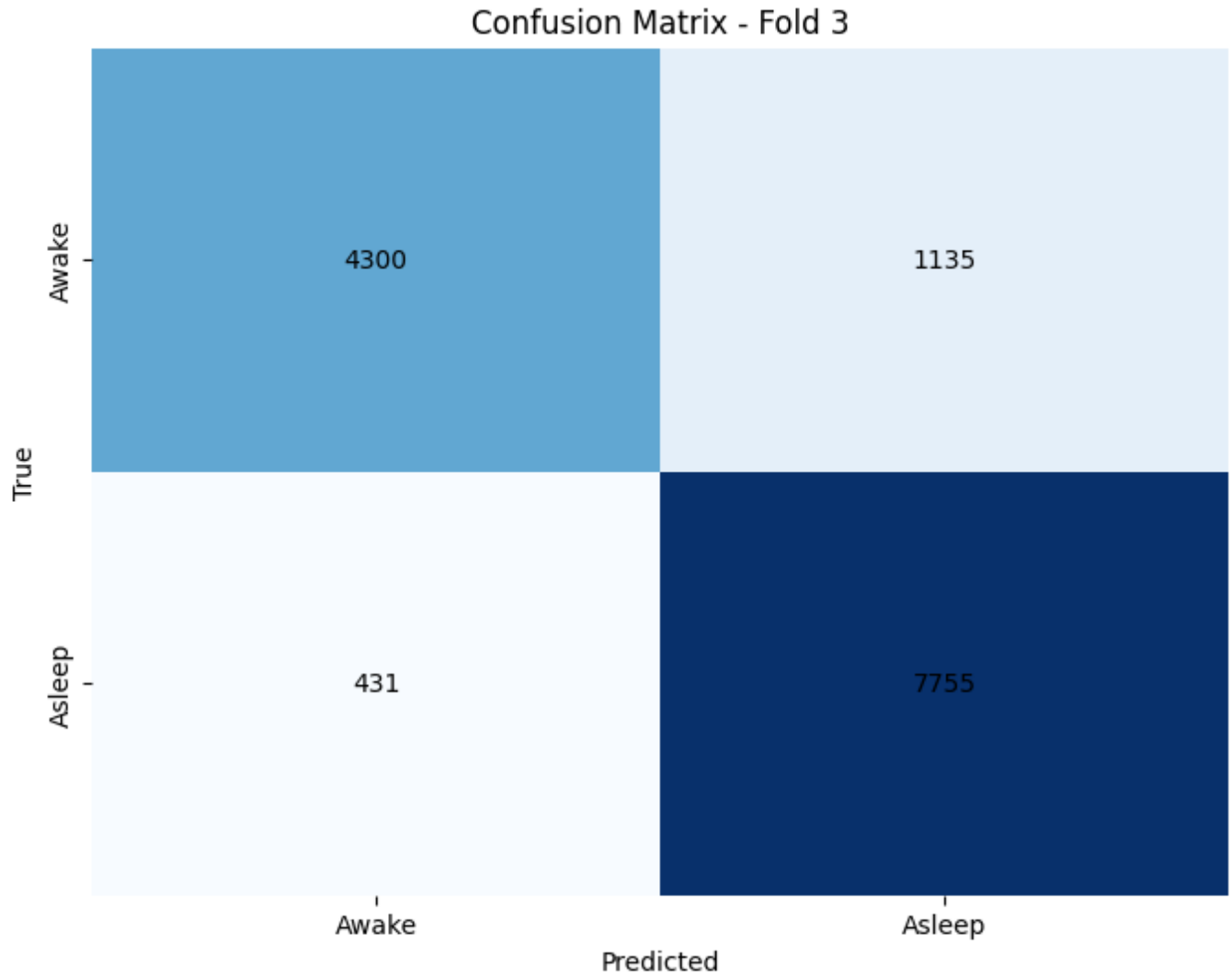
Mean F1 Score across all folds: 0.9038397606822816



F1 Score - Fold 1: 0.9007019859602807



F1 Score - Fold 2: 0.9025249559600705



F1 Score - Fold 3: 0.9082923401264934

Conclusion

The best model we had was the first one with 32 neurons and 100 epochs because, while it had overfitting, it had the best F1 score and was the best performing one.

Second experiment: awake / n-rem / rem

What we changed

We changed this :

```
output_data[output_data == 'w'] = 2.
output_data[output_data == 'n'] = 1.
output_data[output_data == 'r'] = 0.
```

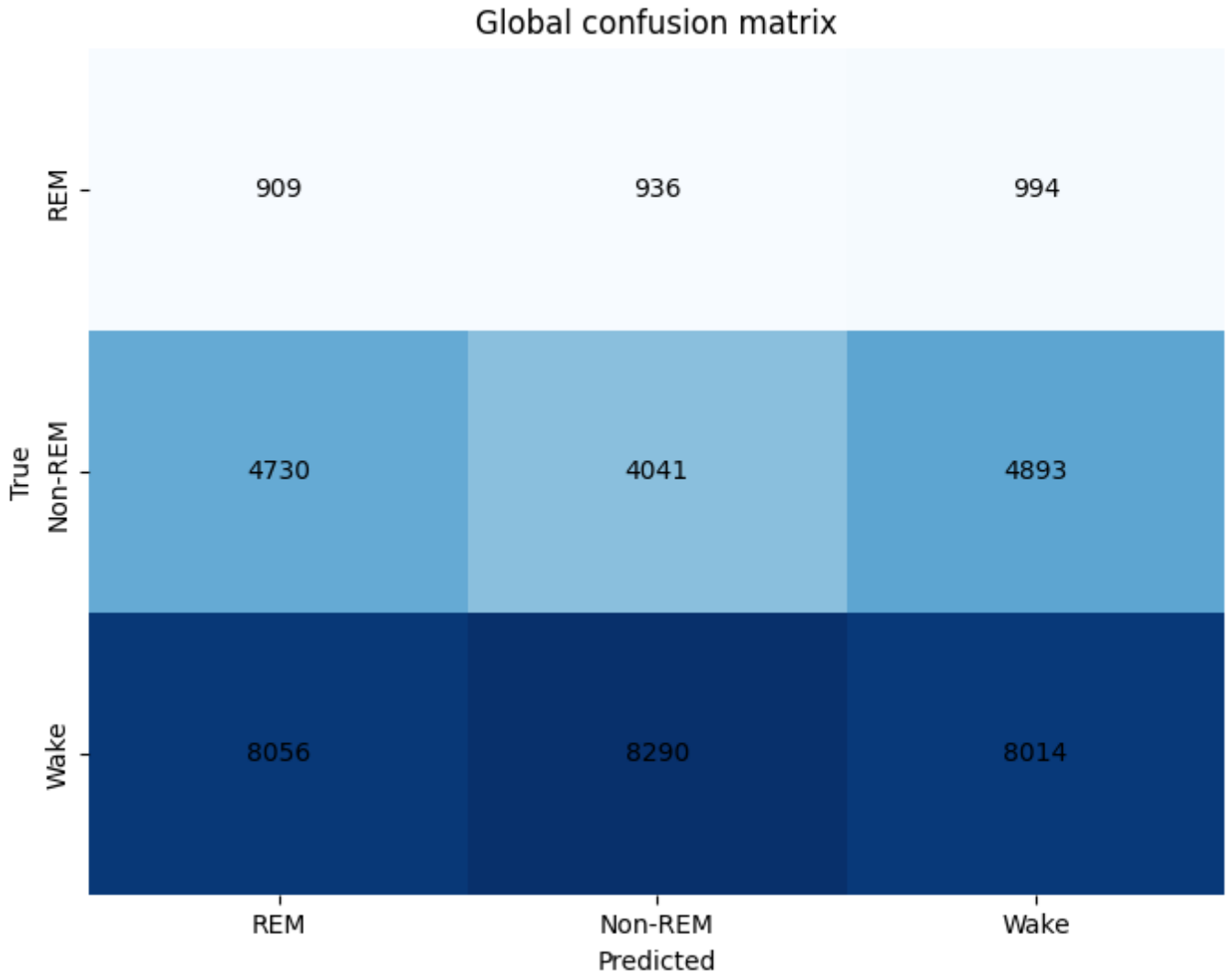
to be able to classify three different classes (instead of merging the rem and non-rem into the asleep state)

and also changed the loss function:

`loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True)`

which is apparently used for multi-class classification

When we used the MSE function, we got this...



We also had to change how the confusion matrix looked like to be able to display all the classes.

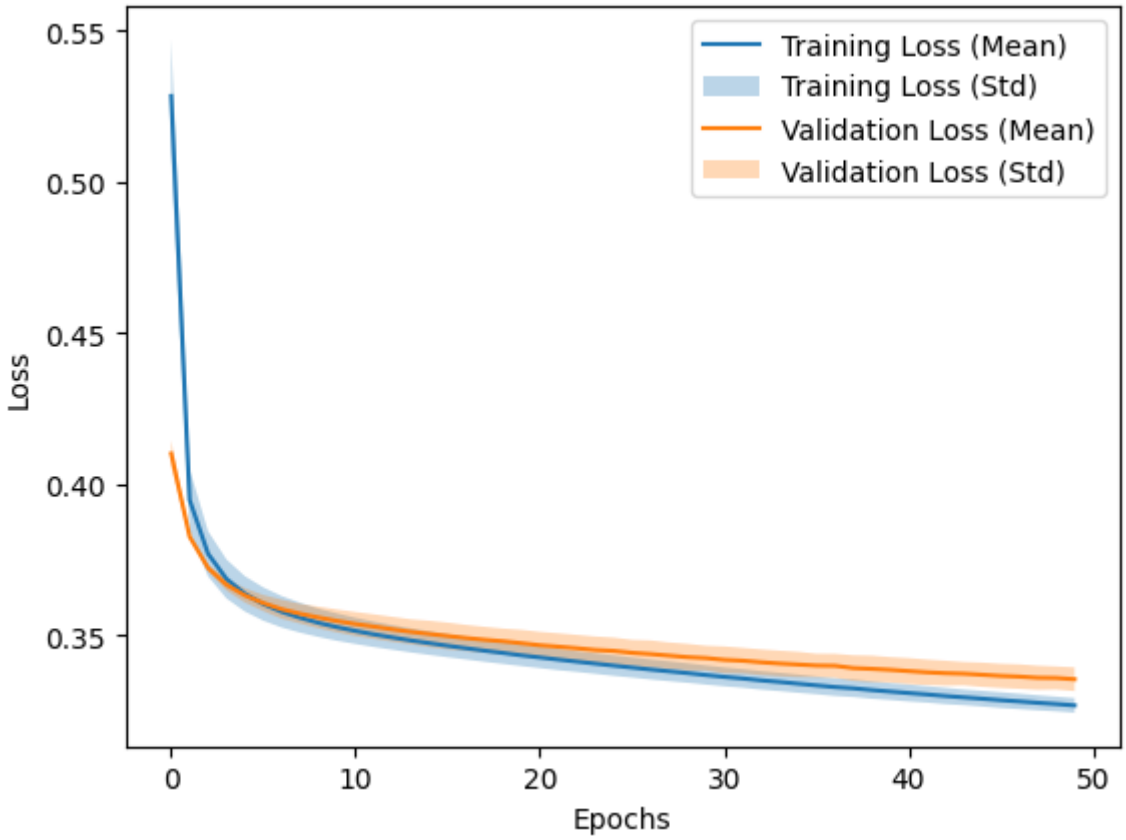
Testing

With this model :

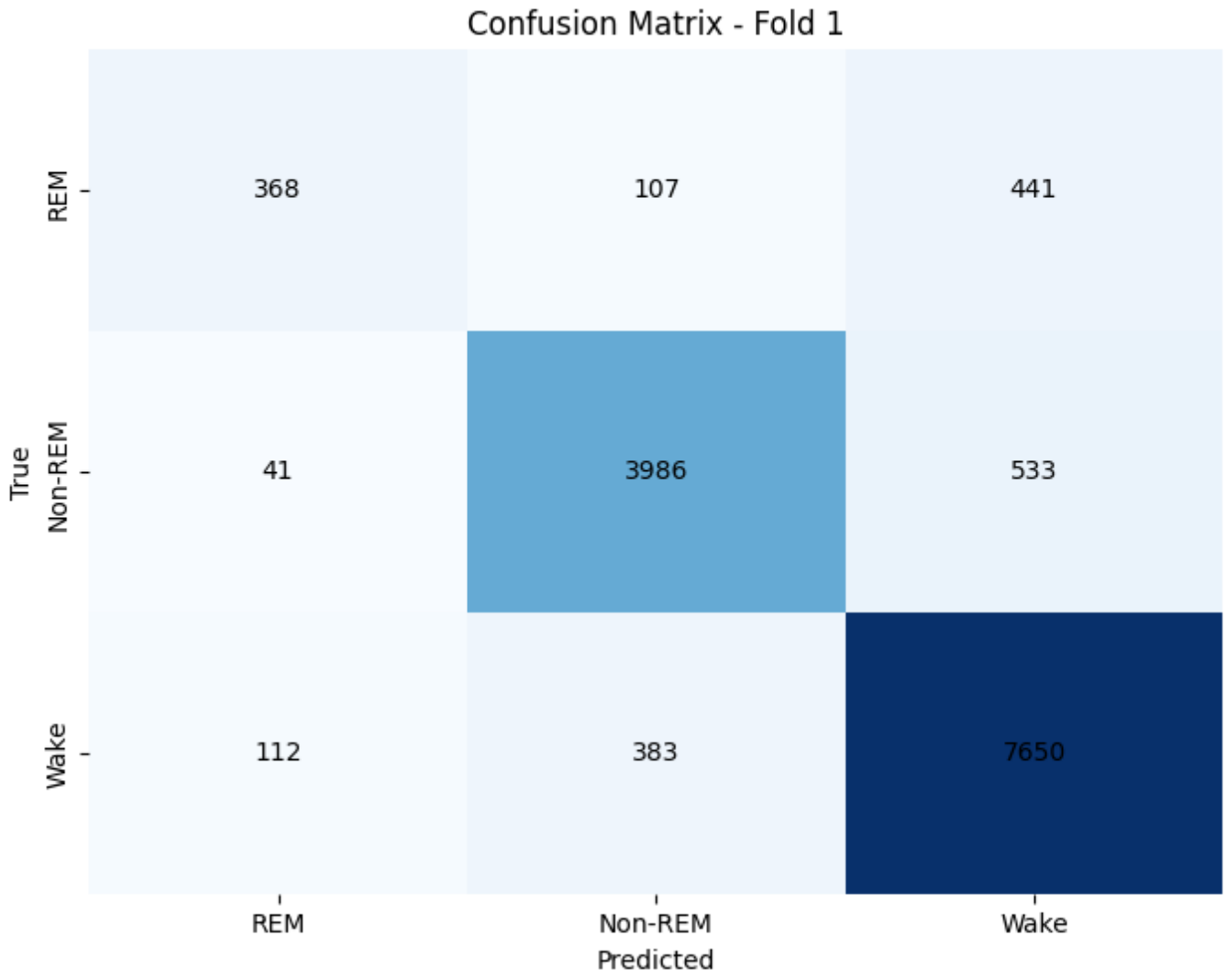
```
def create_model():
    mlp = keras.Sequential([
        layers.Input(shape=(25,)),
        layers.Dense(32, activation="tanh"),
        layers.Dense(32, activation="tanh"),
        layers.Dense(3)
    ])

    mlp.compile(
        optimizer=keras.optimizers.SGD(learning_rate=0.001, momentum=0.8),
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=["accuracy"]
    )
```

```
    return mlp
```

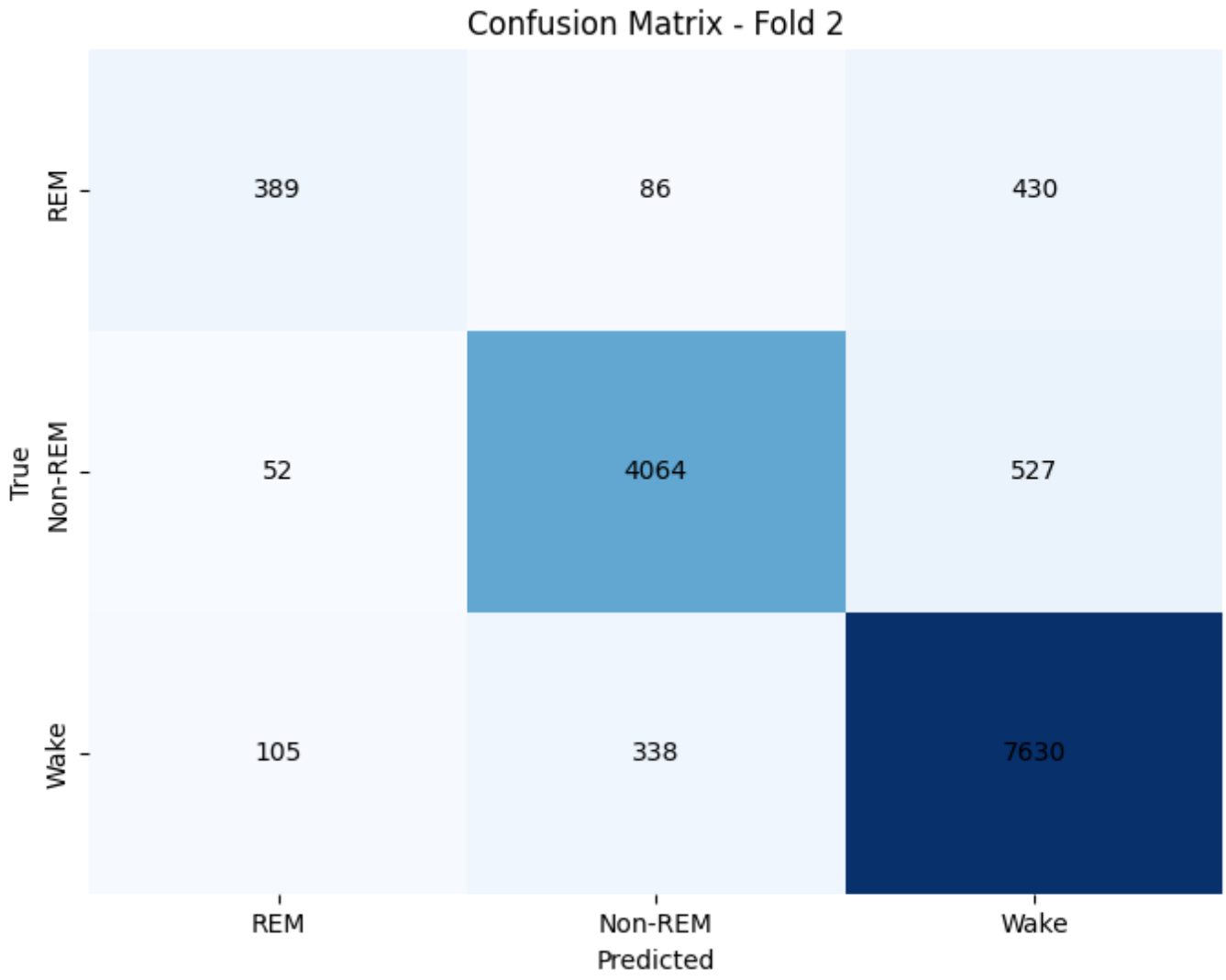
We get these confusion matrices and training loss:



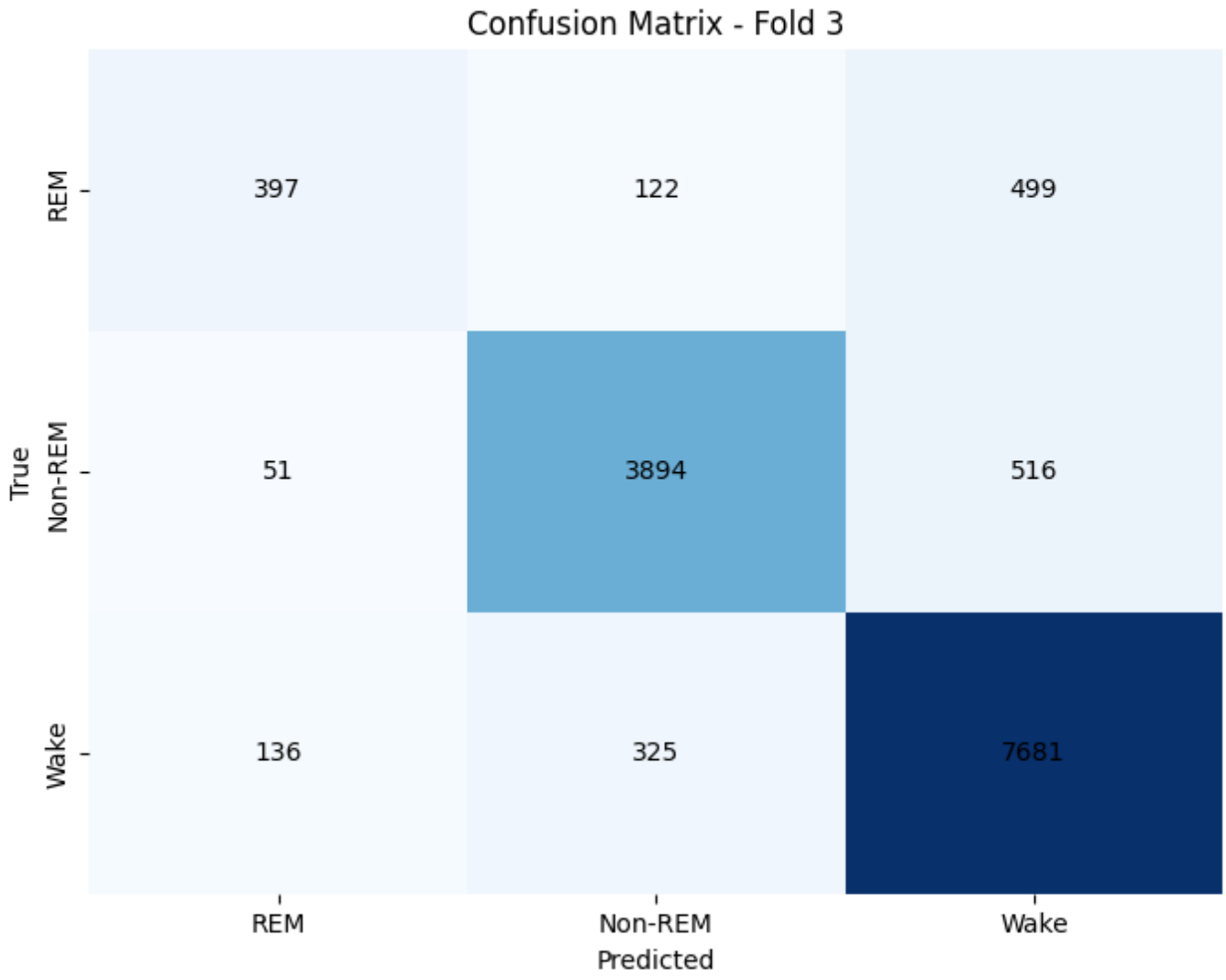
F1 Score - Fold 1: 0.7689416028678527



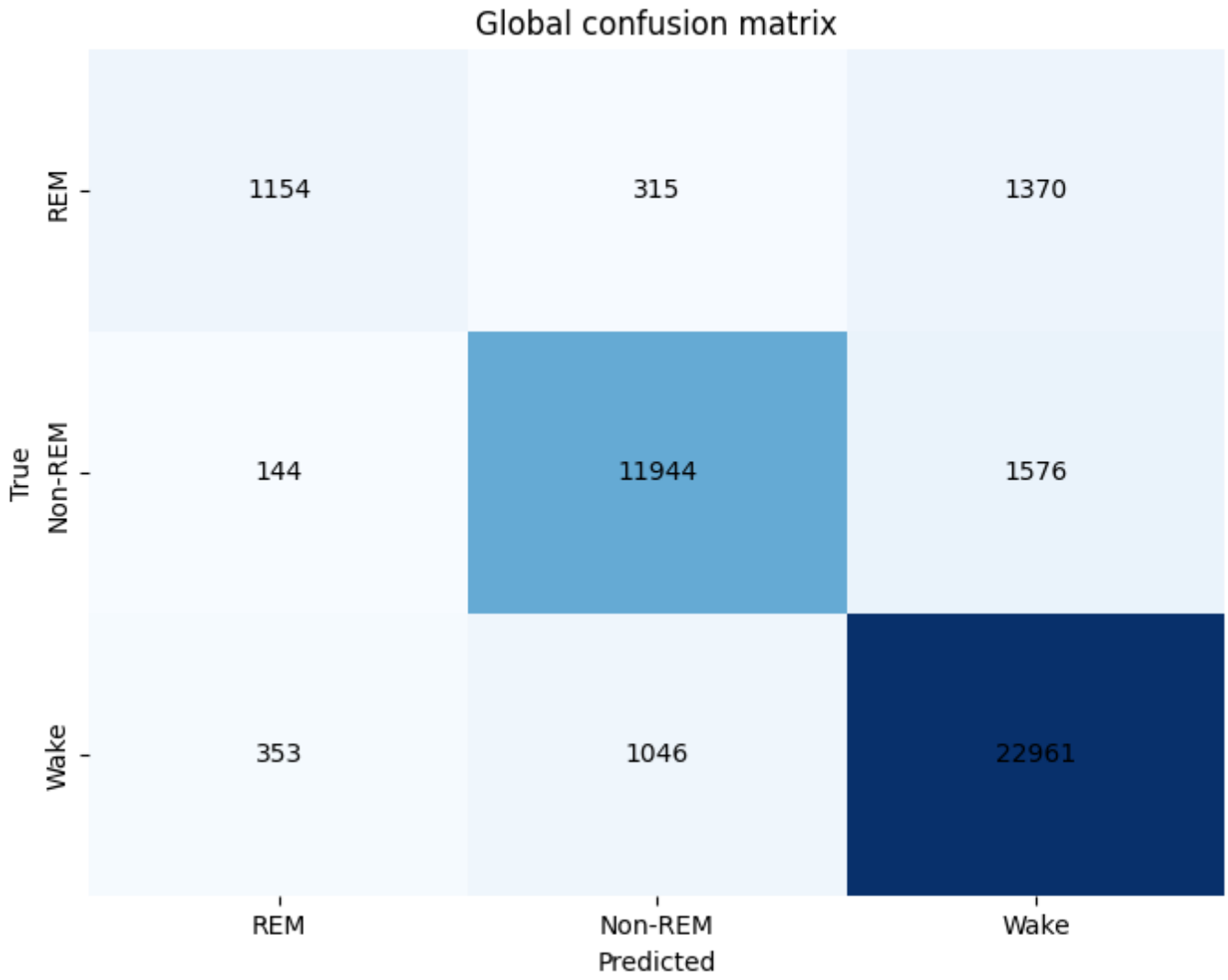
F1 Score - Fold 2: 0.7807675830180094



F1 Score - Fold 3: 0.7642568348041964



Mean F1 Score across all folds: 0.7713220068966863



Let's analyse what we got

As we can see in the global confusion matrix, the lack of data in REM means that there's a clear imbalance in the data, and trying to classify between REM and non-REM is quite hard.

What does our data look like ?

We need to make sure that our data is indeed imbalanced.

After using a function to check it, we get this output :

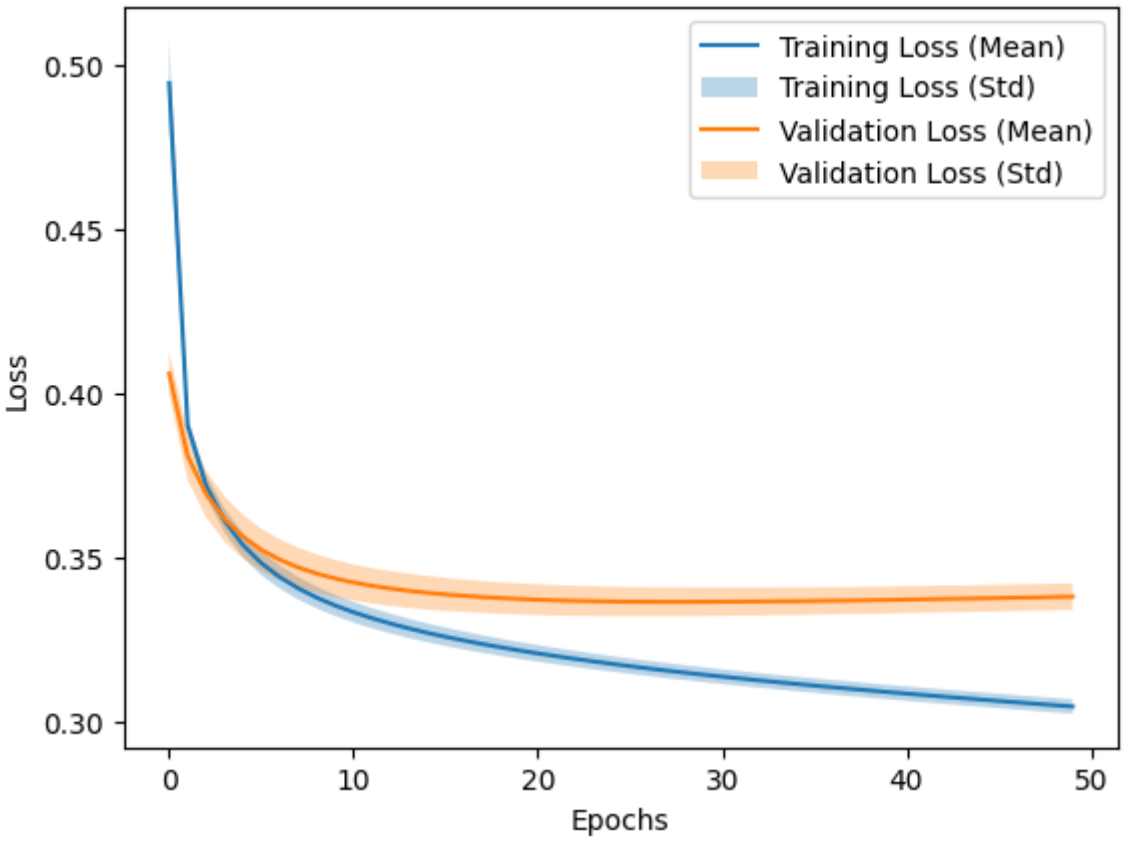
```
The character 'n' appears 13664 times.
The character 'r' appears 2839 times.
The character 'w' appears 24360 times.
```

And we were right, the data is indeed imbalanced...

We tried with those settings:

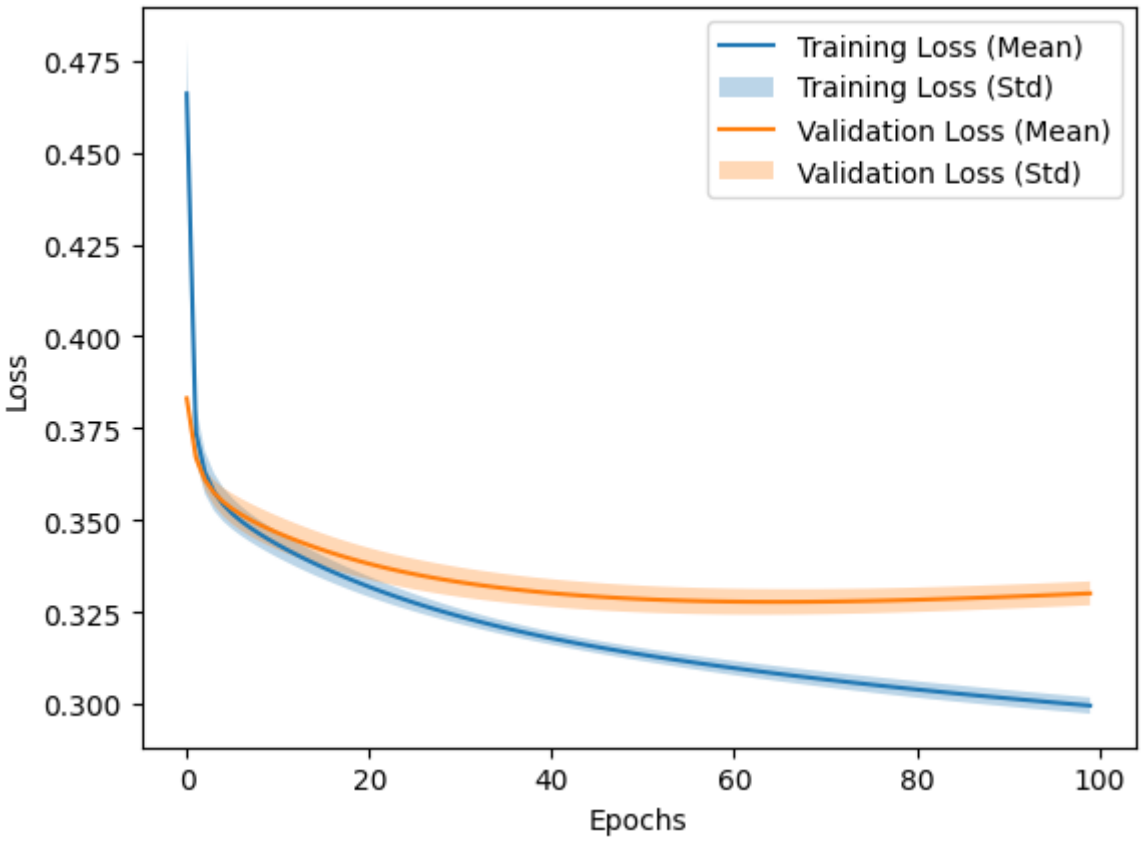
```
def create_model():
    mlp = keras.Sequential([
        layers.Input(shape=(25,)),
        layers.Dense(64,kernel_regularizer=regularizers.l2(0.0001),
activation="relu"),
        layers.Dense(64,kernel_regularizer=regularizers.l2(0.0001),
activation="relu"),
        layers.Dense(3)
    ])

    mlp.compile(
        optimizer=keras.optimizers.SGD(learning_rate=0.001, momentum=0.9),
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=["accuracy"]
    )
```

and got this:



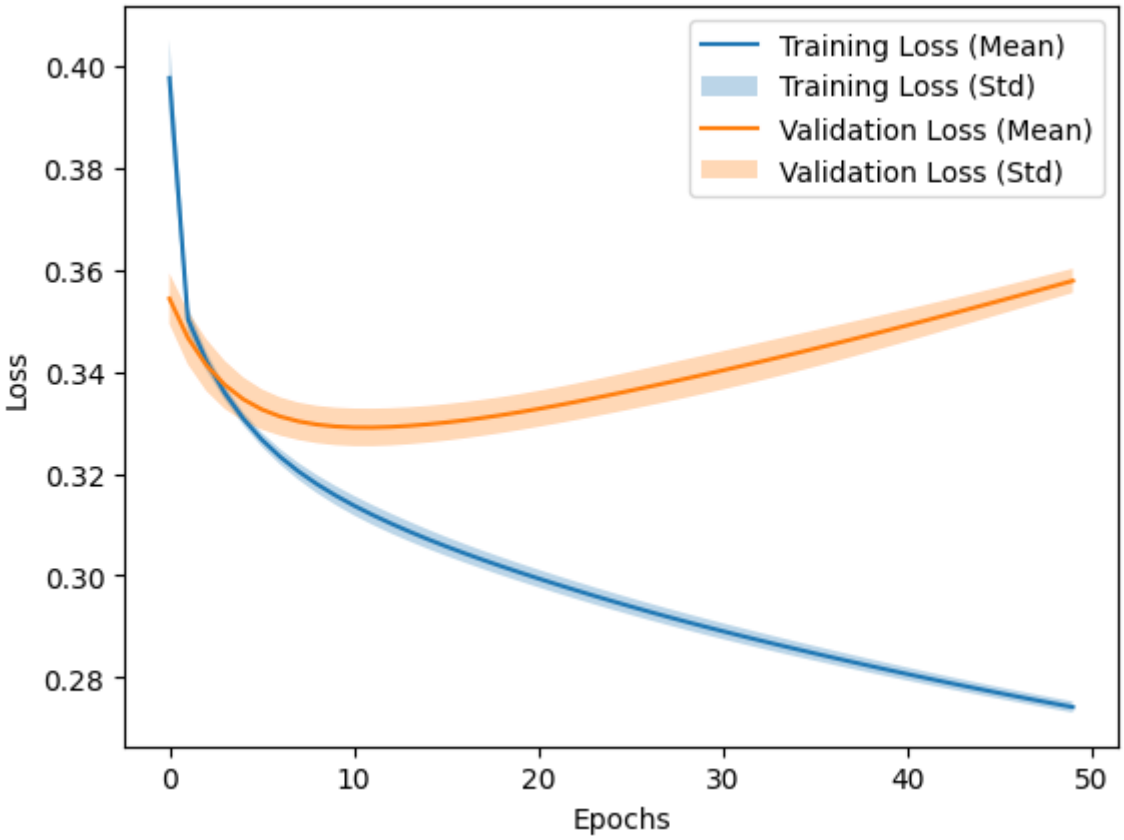
Which seems worse than our first try.

We then tried to do 100 epochs for the first settings and got this :

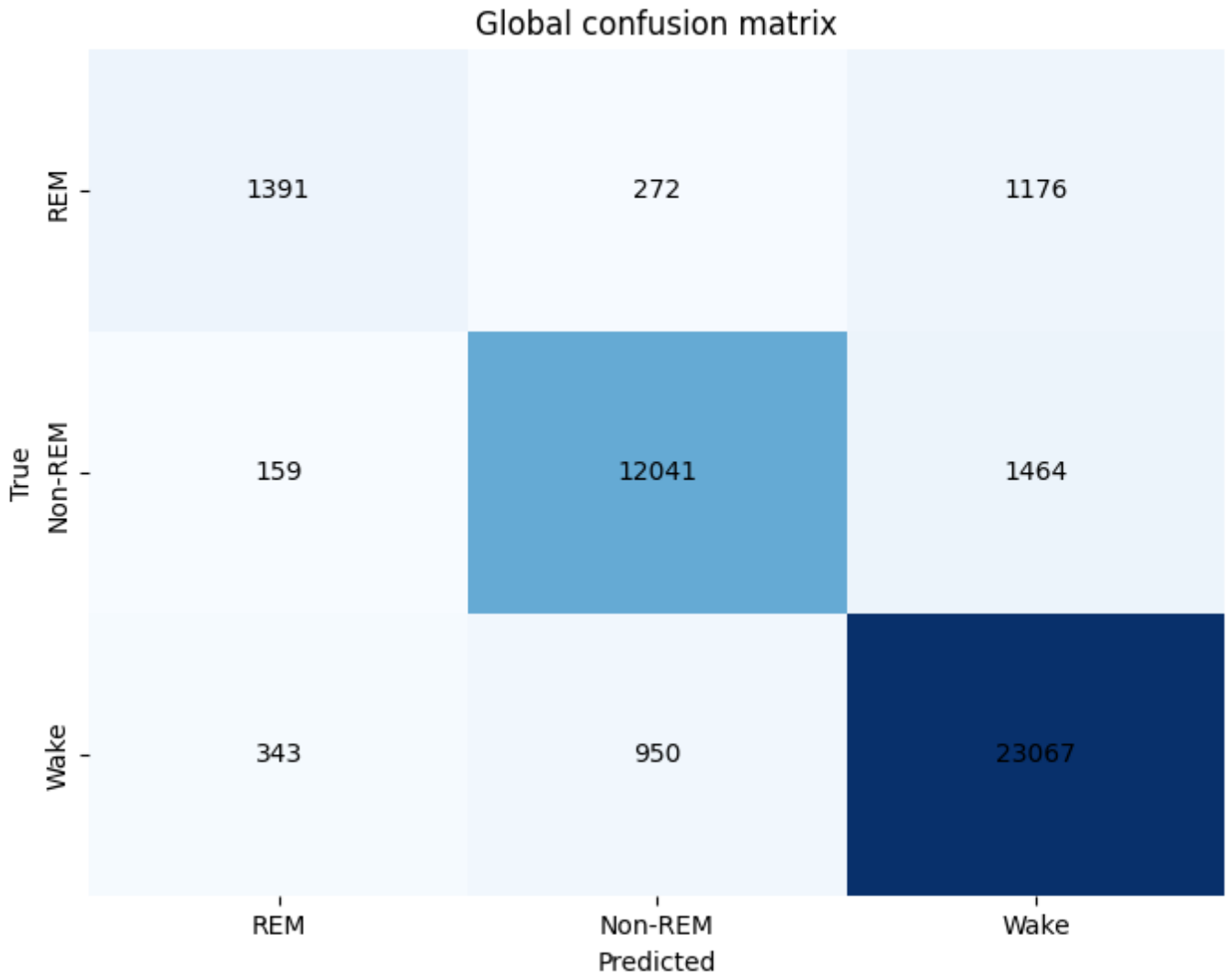


We can see that 50 epochs was the best as it seems that the loss is worsening slightly after that mark...

We also tried changing the gradient descent algorithm, instead of SGD we used "Adam" which gave us this :



It may look bad but with this algorithm, we got our best F1 score yet : 0.8110825591003598



We could maybe try to do an early stop at 12 epochs to get the best loss value possible as it cleary shows overfitting.

With 12 epochs we only get this F1 score : 0.7944308460160157

We tried with 20 but getting the same result...

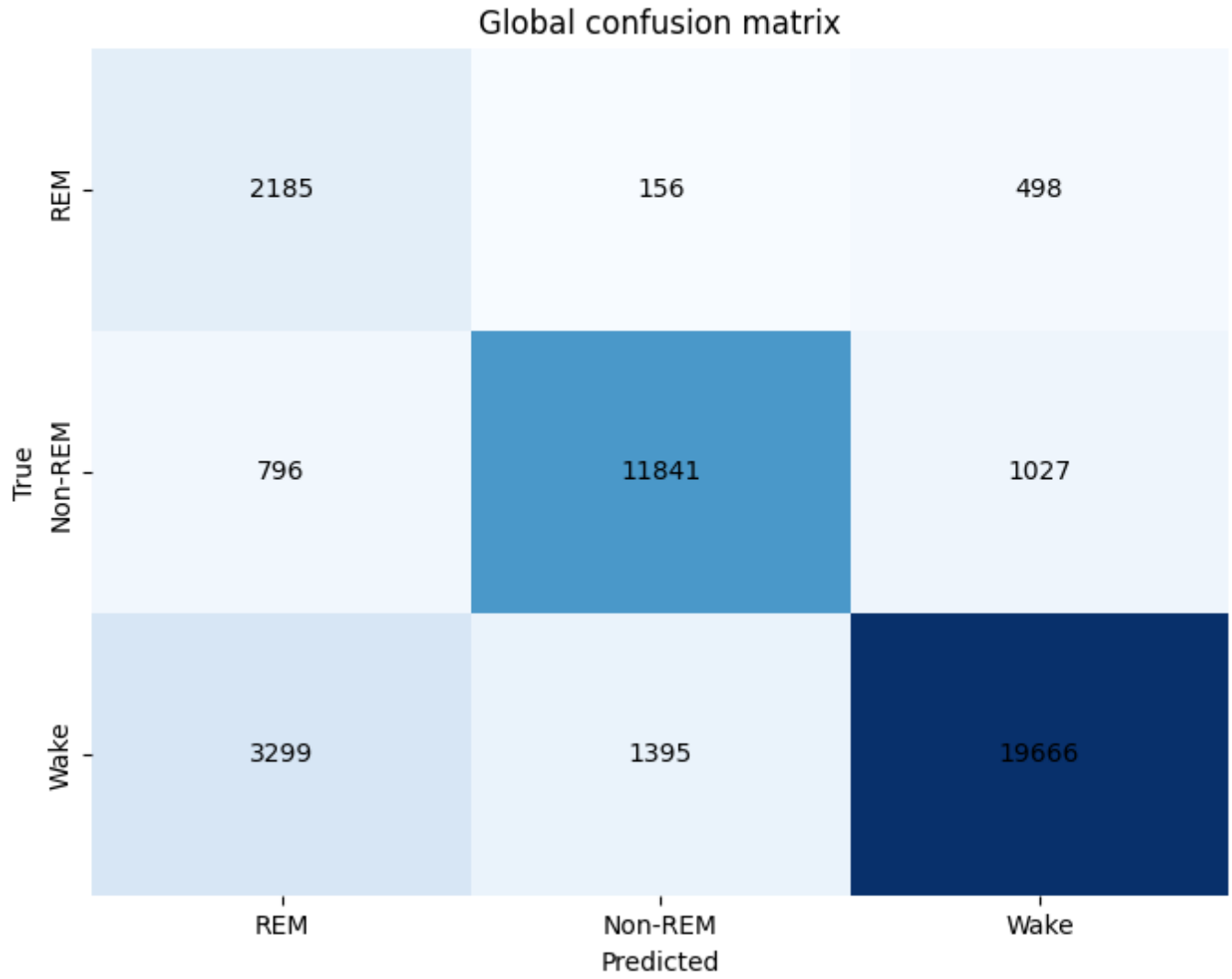
Even if the loss is worse, we're just going to use 50 epochs as it gives the best F1 score !

Doing it once more with 100 epochs gave the best F1 score yet with a value of 0.8223322635925401 !!

We will stop our testing here.

Trying to deal with the data imbalance

We tried using weight that would calculate the amount of time each state would appear in the data and create approriate weights linked to them but in general, even if we got more REM values detected, the overall F1 score was worse by ~ .05



Mean F1 Score across all folds: 0.7394647478711557

We tried with other weights but it doesn't seem to have significal difference or it just made it worse

We also tried data normalization by trying different things we found on internet, undersampling, trying to duplicate the REM values, and so on but it didn't seem to change much of anything, maybe we didn't use the right tools? It's hard to tell...

Conclusion

Here is our configuration :

```
mlp = keras.Sequential([
        layers.Input(shape=(25,)),
        layers.Dense(32, activation="relu"),
        layers.Dense(32, activation="relu"),
        layers.Dense(3)
    ])

    mlp.compile(
    optimizer=keras.optimizers.Adam(
        learning_rate=0.001,
        beta_1=0.9,
```

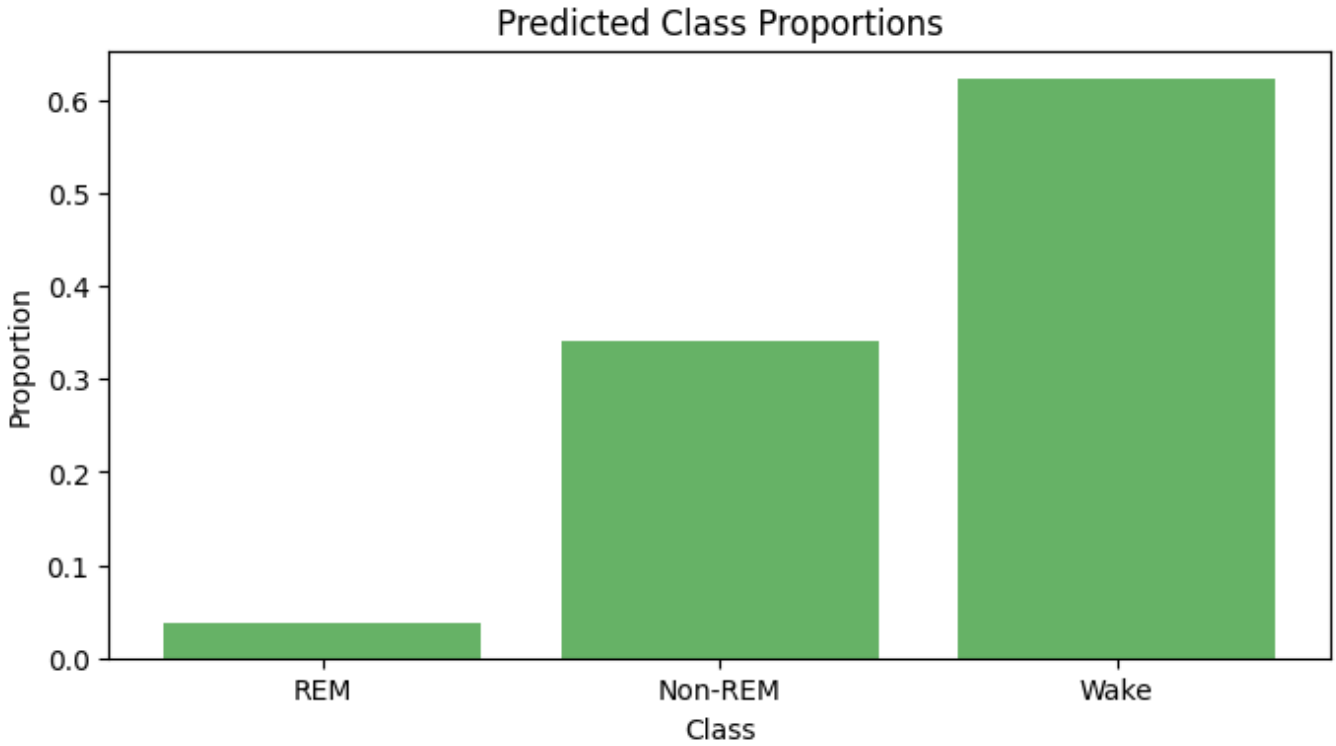
```
        beta_2=0.999,
        epsilon=1e-07,
        amsgrad=False
    ),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"]
    )
```

It didn't seem to make much difference between tanh and relu and by trial and error, 2 hidden layers with 32 neurons seemed to be the best.

Furthermore, Adam gave us much better result than SGD.

In the end, we got these results for the competition data set :

REM: 2995 Non-REM: 27263 Wake: 49851



We sadly couldn't find a way to get better results with our limited knowledge in python and in those libraries.