[Team 07] Proj-C2 Report

Aaron Casey
amrothem@ncsu.edu

Xavier Genelin
xgeneli@ncsu.edu

Alexej Lozevski
aglozevs@ncsu.edu

I. Methodology

A. Project Overview

Terrain types pose an interesting challenge for intelligent prosthetic limbs. Comfort while walking depends on the ability to adjust gait to suit the underlying surface. To begin to address this issue, our task is to use accelerometer and gyroscope readings, from sensors attached to the legs of ambulatory subjects, to predict the type of surface the subject is walking on. There are four types of terrain/activity that are considered: walking on a solid surface (class 0), descending stairs (class 1), ascending stairs (class 2) and walking on grass (class 3).

For our initial model, we use a convolutional neural network (CNN) to classify the type of terrain from the data. The network architecture is based on a study by Zhao *et al.* applying a CNN to time series signal classification [1]. Because our X, Y, and Z for each sensor are not spatial coordinates but sensor readings, we elected to view the problem as a signal classification problem.

B. Network Architecture

Our CNN will have two convolutional layers, as well as a fully connected linear layer. The full architecture of our best model is shown in Figure 1.

The first convolutional layer will have 6 input channels, representing the X, Y, and Z values of each sensor type, with the output channels and kernel size to be determined using cross validation (See Section II). The output feature maps are fed through a ReLU activation function. Next, the activations are pooled using average pooling with a kernel size of 3 and a stride of 1 for the first layer and then a kernal size of 3 and stride of 3 for the second layer.

This output is fed into the second convolutional layer where the number of input channels is determined by the output of the first layer. The second layer's output and kernel size are also determined by cross validation. The output is also fed to a ReLU non-linear activation function, and then is average pooled with a kernel size of 3. To reduce complexity, however, the second layer's pooling uses a stride of 3.

Finally, the activation from this second layer is flattened and fed into a fully connected layer, where two linear transformations, outputting 128 and 4 units, are applied. The loss function is calculated with PyTorch's CrossEntropyLoss, which combines log-softmax and a cross-entropy function to generate class probabilities. The weights are updated after each iteration using stochastic gradient descent, with a learning rate of 0.1 and momentum factor of 0.9.

C. Preprocessing

One challenge of working with this data is the time intervals between the x and y observations are different for each subject (40hz and 10hz, respectively). We also hypothesized that the most important signal information would occur in the close vicinity of a subject's step. Therefore, we wanted to ensure that at least one step was included in each observation fed to the model.

To address both of these issues, we chose a sample window surrounding each y-time which would be passed as the x value for each y observation. We chose a window sufficiently large

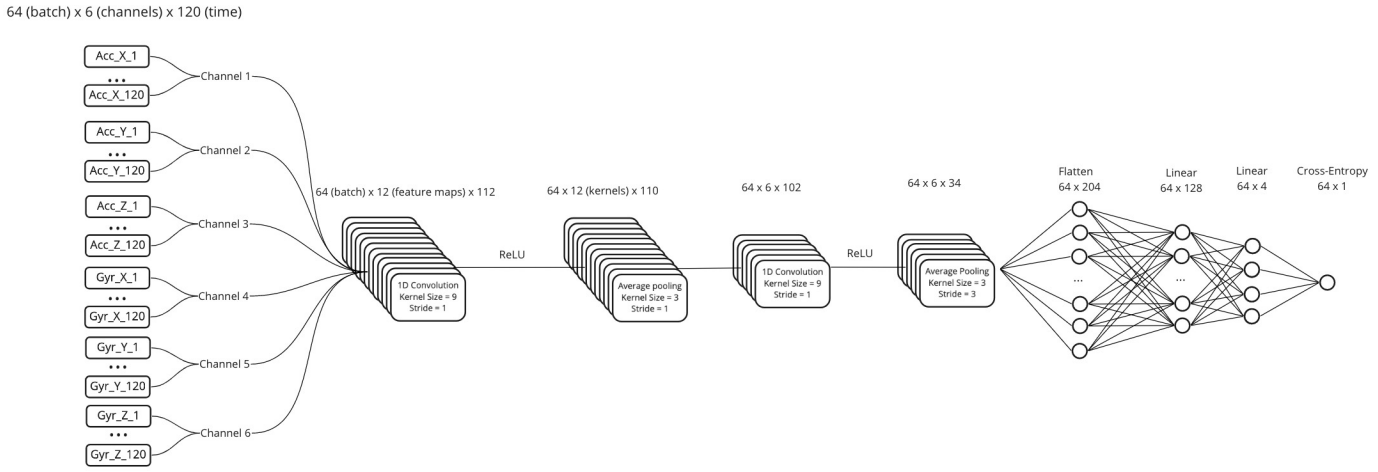

Fig. 1: Convolutional Neural Network Architecture

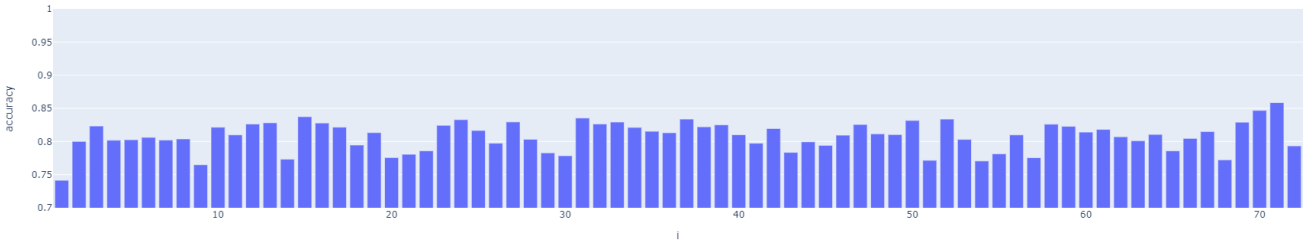

Fig. 2: Model Accuracy

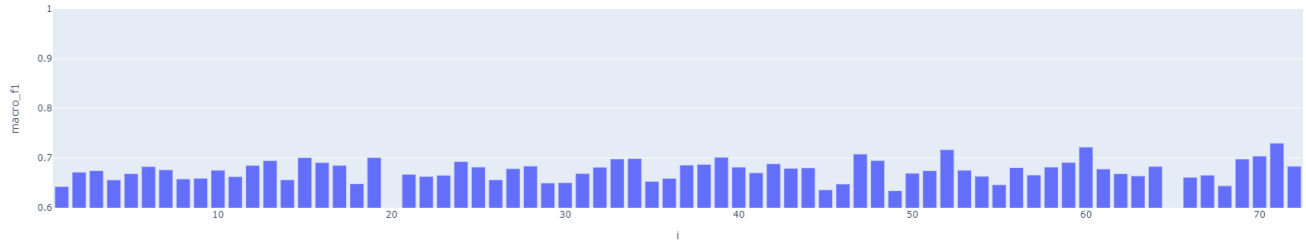

Fig. 3: Macro F1

(3 seconds) that it was very likely to capture at least one step for every observation. Of course, the size of our window also limits the precision of our model, since the x-measurements for a *transitional* y-time will include data from both the old and the new surface type. We had a window of size 8 and applied a rolling mode across the test sample y predictions. This mode will be applied to the center of the window.

II. Model Training and Selection

A. Model Training

After processing the data, we split our training data into a training set and a test set. We did an 80-20 split for the training and test sets respectively. Importantly, we elected *not* to shuffle the data before splitting, meaning that the test set consisted of the measurements from later subjects. This more closely models the hidden test data, because it means that some subjects' data is not used for training, but will be tested upon (this same methodology was applied to our validation splits later).

After this split, we load the data into tensors. The X tensors have three components: the length of the training set, the number of channels, and the number of measurements in the time window. The input contains 6 channels, representing our 2 sensors with x, y, and z components.

In order to address class imbalance in the data, data for each batch were selected using class-weighted random sampling (with replacement).

B. Model Selection

To select the hyper parameters for the model, we used 5-fold cross validation. We considered batch sizes of 32 or 64, kernel sizes of 5, 7, or 9 and output sizes of 6 or 12 for *each* convolutional layer. These candidate hyperparameter values were adopted from Zhao *et al.* [1]. Therefore, a total of 72 models were evaluated (in addition to evaluating different numbers of epochs). For each of the 72 models, we calculated the accuracy and the macro-averaged F1 score. We used the *macro-averaged* F1 to address the class imbalance in the data. When first assessing accuracy, we observed in Figure 2 that the 71st model was the most accurate, with an accuracy of 0.8586.

Next we checked the macro-averaged f1 value, which produced similar results. As we can see in Figure 3, the 71st model also has the highest macro f1 value. Given that both measures of model performance agree, we select model 71 as our model. From model 71, we identify the following hyperparameters:

Hyperparameter	Value
Batch Size	64
Conv1 Kernel Size	9
Conv1 N Filters	12
Conv2 Kernel Size	9
Conv2 N Filters	6

TABLE I: Hyperparameters of the Best Model

To determine the number of epochs to use, we looked at the learning curve for each of the 5 folds at all 16 epochs. From Figure 8 we can see that at each fold there was a large drop in the loss from the first to second epoch. Then they all slowly move towards a loss of about 0.05. Therefore,

we determined that little benefit is gained after the first few epochs, and overfitting is likely.

To confirm that this leveling off was supported in our validation data, we looked at the accuracy and macro averaged F1 for each fold (not pictured). We determined that there was not much improvement after the first few epochs, and settled upon 5 as our final epoch number. *However*, since submitting our predictions, we have determined that after averaging the scores for each fold, the model's performance was still improving after all 16 training epochs (Figure 4). Therefore, in the future, we will explore longer training times.

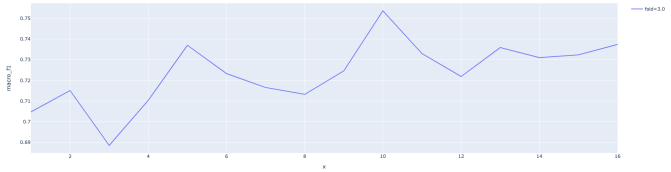

Fig. 4: Validation macro-averaged F1 score for each training epoch

C. Smoothing

After submitting our initial predictions, we observed that our predictions were more irregular than was possible in real life. For instance, one stretch of time was labeled $[0, 0, 3, 0, 0]$. However, this would imply that the subject stepped on grass for approximately .1 seconds, which is highly unlikely. This irregularity can also be observed on a macro scale. Although they depict different subjects, it can be seen that the predictions for subject 9 (Figure 6) are far less regular than the actual values for subject 1 (Figure 5).

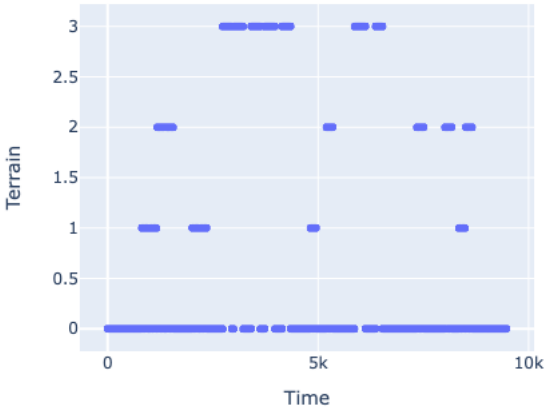

Fig. 5: Actual labels for subject 1

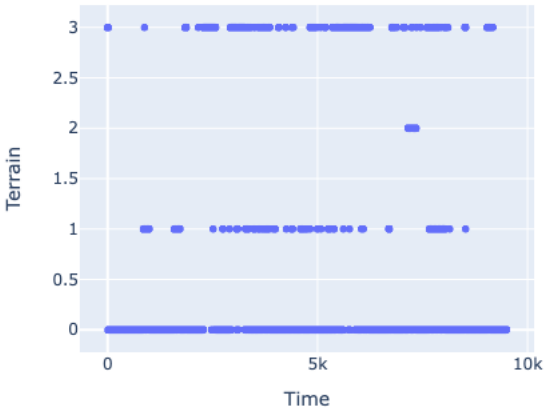

Fig. 6: Predicted labels for subject 9

To address this indiscrepancy, we applied smoothing of the predictions. We used a rolling window and applied the mode to find the most frequent label. We found that, in the range of 1-25, a window of size 8 maximize the macro F1 Score. Therefore, for our final prediction, we applied mode smoothing with window size 8.

III. Evaluation

After selecting the best model, we retrained this model for five epochs using the full training set. Evaluating our model with the full held-out test data, we obtained the results shown in Figure 7 and Table II. We observe that we achieve 88% accuracy on our test set, and a macro-averaged F1 of 0.8. However, we also observe that we had a high error rate for class 1 (ascending stairs) and class 3 (walking on grass). Specifically, the low precision and high recall means that we overpredicted class 1 (as can also be observed in Figure 7). Class 3, on the other hand, has a high precision and low recall, indicating that it was under-predicted.

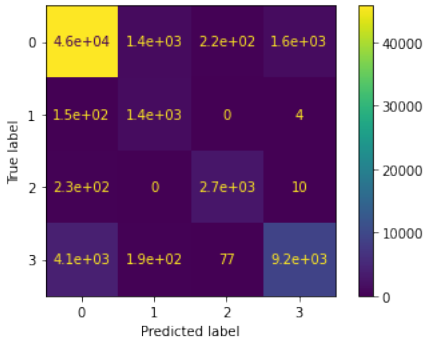

Fig. 7: Confusion Matrix for (known) Test Set Predictions

Using the model on our held out test set, we obtained the following results:

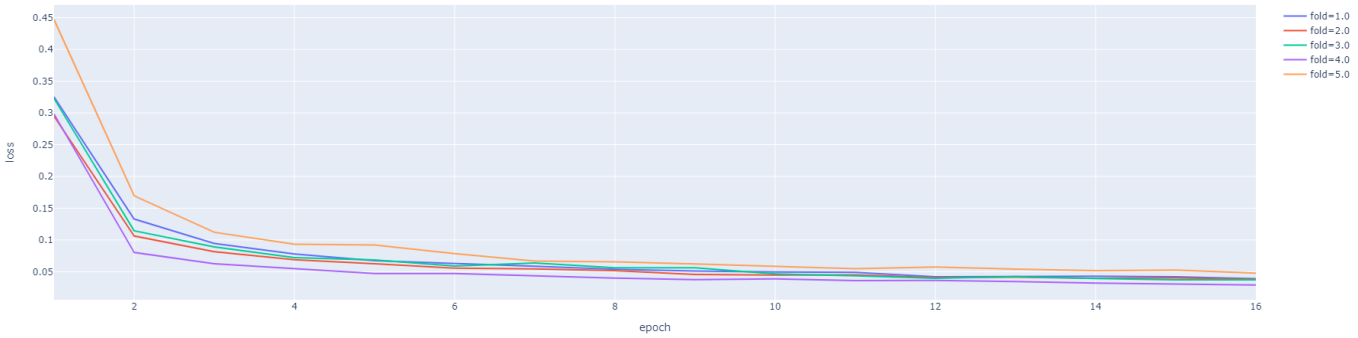

Fig. 8: Learning Curve

Class	Precision	Recall	F1-score	Support
0	0.91	0.94	0.92	48949
1	0.48	0.90	0.63	1591
2	0.90	0.92	0.91	2991
3	0.85	0.68	0.76	13552
Accuracy			0.88	67083
Macro Avg	0.79	0.86	0.80	67083
Weighted Avg	0.89	0.88	0.88	67083

TABLE II: Performance metrics for (known) Test Set Predictions

Finally, we retrained our model on *all* labeled data (including the held out test set), created predictions, and applied mode smoothing with window size 8 for the hidden data.

REFERENCES

[1] B. Zhao, H. Lu, S. Chen, J. Liu and D. Wu, "Convolutional neural networks for time series classification," in Journal of Systems Engineering and Electronics, vol. 28, no. 1, pp. 162-169, Feb. 2017, doi: 10.21629/JSEE.2017.01.18.