[Team 11] Proj-C: Final Report

Nathan Kohen (nrkohen), Trenton Wallis (tcwallis), Caleb Wheeler (cbwheele)

I. Methodology

Recent advancements in prosthetic limbs have reduced the size of the processing units required for controlling them to a reasonable level to attach them to the wearer. However, the movements these prosthetic limbs need to make vary greatly depending on the terrain the wearer is traversing. In order to determine which type of terrain the user is traversing, one of the solutions is to attach an Inertial Measurement Unit (IMU) to the user's leg which records and evaluates acceleration and gyroscope data in the x, y, and z axes. For this project, we have created a neural network that can take in this data and predict what type of activity and terrain a user is moving over. These were classified into four types: walking or standing on solid ground, walking down stairs, walking up stairs, and walking on grass.

Our training data set consisted of x, y, and z measurements for both a gyroscope and accelerometer. These measurements were collected for eight participants across multiple sessions per subject. Each session was approximately twenty minutes long, and the number of sessions each subject created data for varied anywhere between one and eight sessions per subject. The gyroscope and accelerometer data were sampled at a rate of 40 hertz, so there were 2,400 samples per minute in each session. The type of activity was determined by watching a video of the subject during the trail that was shot at 10 hertz, so correspondingly, the y values were labelled in .1 second increments. We were told to assume that the y labels were accurate to approximately two frames of video, or about 200ms. The labelled data we were given for training had a much higher representation of '0' labels than any other label, which can be seen in Fig. 1 [2].

The structure of the neural network is based around a Convolutional Neural Network that can classify traditional images. The architecture is composed of two convolutional layers, one that pads the data with zeros, the other that does not. These are followed by three fully-connnected layers. Each layer in the architecture has a corresponding relu activation function. There is no optimization performed on the model, and there is no regularization in the form of normalization or dropout techniques, as we were able to attain decent accuracy without implementing these techniques, and we had no evidence to suggest they may have helped regardless. [2]

II. Model Training and Selection

In lieu of data augmentation, we created a custom methodology for splitting up the data into samples that can be fed into the convolutional neural network. This special pre-processing creates an "image" of all of the data 1.5 seconds before each sample, padded with zeros at the start of a session. Once this special pre-processing is applied, there is an "image" that corresponds to the data we want to process for each sample that corresponds to x_time.

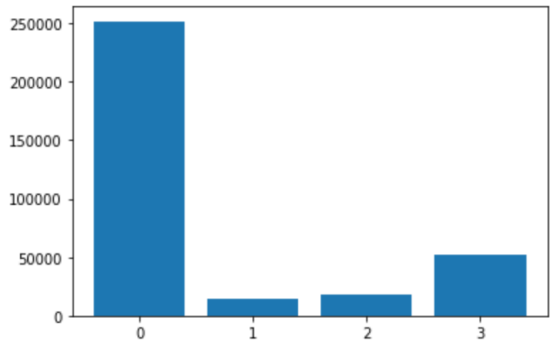

Fig. 1. Number of labels of each type.

A. Model Training

The biggest challenge for splitting the data up into training, validation, and test sets was that we did not want to have data samples overlap with each other. Therefore, we went through the data for every session and divided the data into "sets" which contained the number of samples that were taken over a 20 second interval. We added buffers between each of these sets that were 1.5 seconds long in order to be sure that the data did not overlap.

Once these sets were created for every session, we used a randomization function (specified by a seed so that the same sets would be chosen during different training attempts) to split the sets up into training sets, validation sets, and testing sets. Once the sets were created and split up, we looped through the sets to extract only the "images" corresponding to each type. This process is summarized in Fig. 2. Finally, before feeding the images marked for training into the model, we randomized all of the images so that the model wasn't biased by the ordering of data in batches that arose from the way we split them up into sets.

B. Model Selection

The model was selected as a baseline for training on the convolutional neural network friendly input data. Because of the temporal data that is encoded within each image being trained on the CNN, the architecture is quite simple. The architecture first sends the input through a convolutional layer

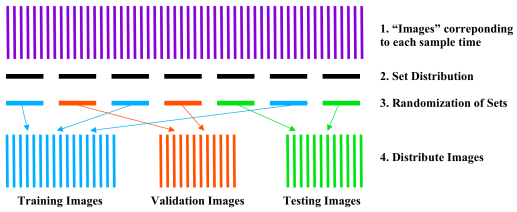

Fig. 2. Process for splitting images.

that extracts 128 features using a 30 x 6 kernel without padding the data with zeros. The purpose of the padding is to not disrupt the model from drawing conclusions between rows in the image corresponding to the 6 sensor values at any given measurement time, but since the kernel is 6 wide, the image size is not reduced in that dimension. There is no pooling involved between layers for this same reason. The network then transforms the data with a fully connected layer stepping from 3968 features in the flattened data to 128 before stepping down to 64, 32, and finally 4. Each of the CONV/FC layers have a relu activation function applied as well. The convolutional layers have a 2-D dropout applied at a rate of 0.5, and the fully-connected layers have a dropout of 0.2 applied. The original prediction was made with only 1,200 images, but this greatly improved model was trained on 1,044,800 images. This increase in data fixed the over-fitting issue that plagued the initial predictions.

III. Model Evaluation

Our model was trained over 8 epochs, and achieved an accuracy of 96% using the model with the lowest validation loss. Over the 8 epochs a batch size of 20 was used, with a total training sample size of 1,044,800 images. For our test metrics we were able to get 97% accuracy for 0 labels, 97% accuracy for 1 labels, 93% accuracy for 2 labels, and 92% accuracy for 3 labels. These results are summarized in Fig. 3 Our evaluation used 20,800 images.

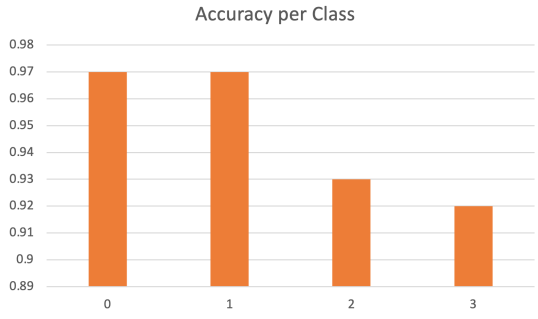

Fig. 3. Accuracy Per Class

Based on Fig. 4, our training loss decreases over the first five epochs, but then goes up higher for epochs 6-8. Our validation loss reaches a minimum on the 5th epoch, so that is the model that was chosen. Based on our model training, we usually train our best model in 1 to 5 epochs based on validation loss.

As shown in Fig. 5, the accuracy of our training set is increasing rapidly, and this makes sense, as the training loss is also decreasing on average. It is also encouraging that the validation accuracy is almost constant, as the validation loss is also constant in Fig. 4.

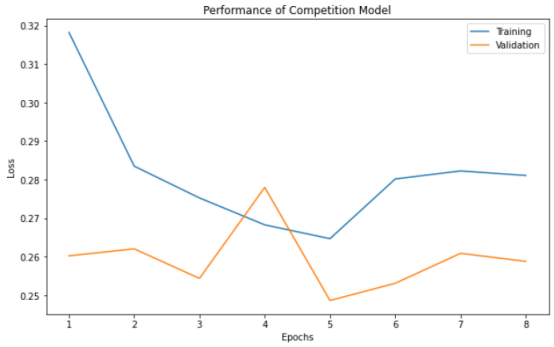

Fig. 4. Loss Performance of Model.

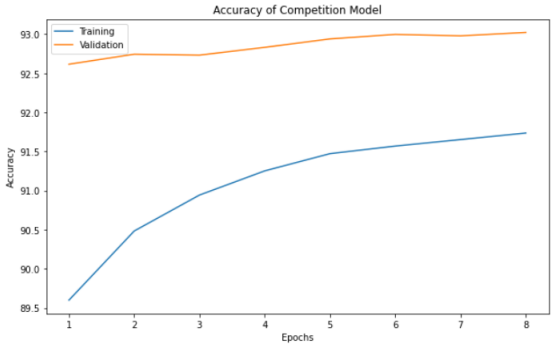

Fig. 5. Accuracy Performance of Model.

The average F1 Score for our original model was 0.322. As previously mentioned we believed there was a bug in our code causing our F1 Score to be lower than we thought it should be. With some debugging we found we were accidentally rotating our images of the data causing our validation and test to look good (since it is testing on the rotated images), but our actual score would be low since it would be tested against non-rotated images. Once this bug was fixed we tried training and testing our new model and received much higher F1 Scores. Our best competition result for our model we received an F1 Score of 0.85, which is a significant improvement over 0.322. Using some sessions for evaluation, we were able to obtain F1 Scores upwards of .91, this may indicate that our model robustness could improve if we augmented the images with noise or increased our data set with more volatile data. Calculated over all 29 sessions, the average F1 Score for each class was 0.965 for class 0, 0.821 for class 1, 0.875 for class 2, and 0.860 for class 3. This is visualized in Fig. 6. We also evaluated if our model is over or under guessing each label. As shown in Fig. 7, our model tends to under guess for the label 0, over guess for the label 1, over guess for the label

2, and under guess for the label 3. Ideally we would want our model to guess each label 100% correctly, with a "guess" score of 1. Guessing was calculated using $guess\_score = count\_of\_guessed\_label\_x / count\_of\_correct\_label\_x$.

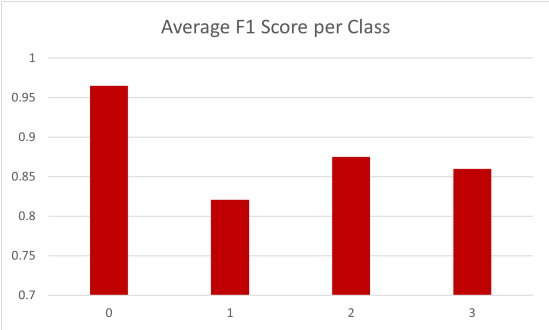

Fig. 6. Average F1 Score per class

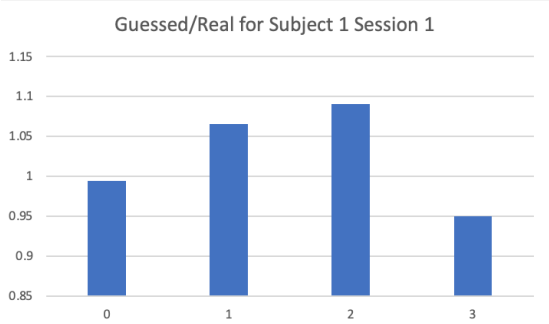

Fig. 7. Guessed vs. Real

IV. Conclusion

Overall, we are very happy with our performance in the competition project. It was very disheartening when our F1 score was so low at first, but after we found the bug we were able to achieve decent results. It was also very rewarding to work on a project with such a tangible application.

V. Acknowledgement

Our group would like to heartily thank our Professor Dr. Edgar Lobaton, as well as our teaching assistants Hangjin Liu and Sanjana Banerjee for offering assistance and knowledge throughout the semester. This class has been extremely interesting and insightful and will provide us a great launchpad for our future in the machine learning space.

References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting,"p. 30, https://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

[2] E. Lobaton. (2022). Terrain Identification from Time Series Data. Raleigh, NC: NCSU. https://docs.google.com/document/d/1DA-0_IYSno_OREDATejhvXCsjscPQaDK/edit?usp=sharing&ouid=101042411330286391908&rtpof=true&sd=true.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," presented at the International conference on machine learning, 2015, pp. 448–456. https://arxiv.org/ftp/arxiv/papers/1512/1512.00242.pdf