Final Project Report

Sign Language Classification

Machine Learning 2 – Professor Amir Jafari

Tina Nguyen

I. Overview

For the final project, our group decided to work on classifying images of sign language characters. The broad objective is to be able to translate sign language to text or speech to assist the disabled community.

We obtained the dataset from Kaggle. Our objective is experimenting with different frameworks. Since Keras and Pytorch are the two frameworks we have had exposure to before, my partner experimented on both. Tensorflow 2.0 is new and required a little more studying, so I volunteered to take on this for learning purposes.

II. Procedures

My first task was uploading the data to the cloud. It seems that the GW's AWS account does not allow students to create a S3 bucket, so it took me a lot of time to figure out how to upload the data to the cloud. I tried to upload the data to Google Drive and made it available to public, but it didn't seem to work 100% of the time. Therefore, I ended up having to create an AWS Educate account and created a S3 bucket in that account. Using wget to get the data from this S3 bucket now worked for both my partner and me.

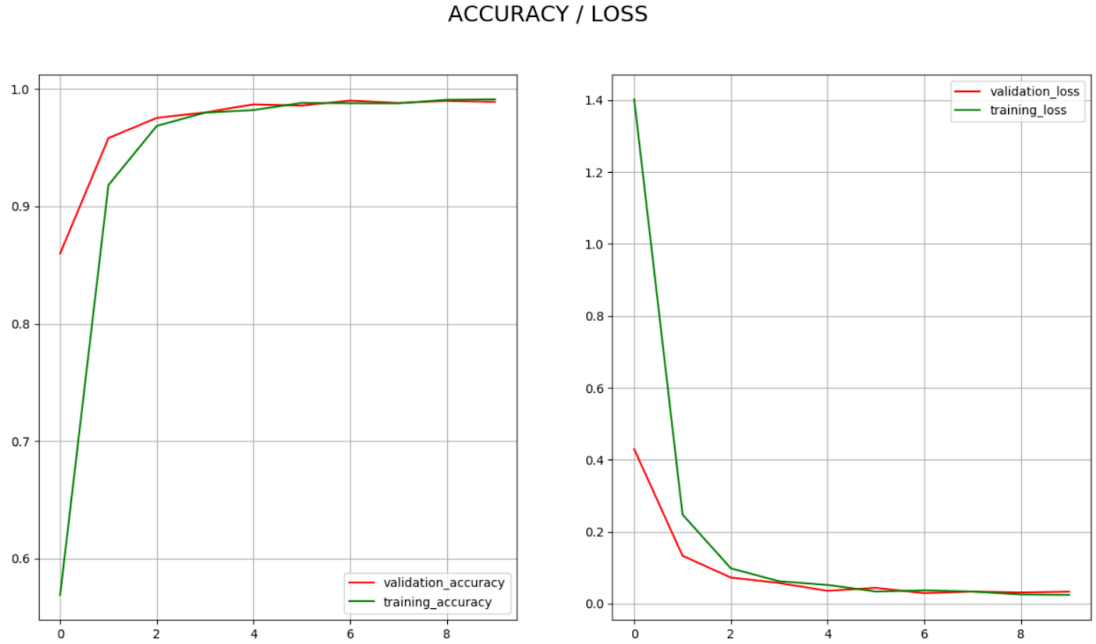
I was also in charge of building a deep learning network in Tensorflow 2.0. Our data contains two csv files for training and testing sets. Although working on different frameworks, but for the purpose of being consistent, my partner and I agreed on splitting the training data in training and validation sets and hold out the testing set to test the model. We both preprocessed the data on our own but still followed the same rules. I extracted the label from the dataset, normalized the pixel values, and split the data into training and validation sets with a ratio of 7:3. After that, I reshaped the data and created a training dataset including features and labels and a validation dataset using Dataset.from_tensor_slices().

I experimented with both Adam and SGD, but I chose to use Adam because it was faster and provided better accuracy than SGD by about 10%. I chose Sparse Categorical Entropy instead of regular Categorical Entropy because I didn't do one hot encoding the target but only converted them to tensor. Since the targets are integers, ranging from 0 to 24, using Sparse Categorical Entropy makes sense.

At first, I experimented with one layered convolutional network. The first network only contained a Conv2D and MaxPool2D with ReLu as activation function. This network returned 92% accuracy after training 10 epochs on the training data. I saved the parameters of this model and loaded it to test on the hold out data. However, the accuracy on the hold out test data was only 71%. I suspected that the model was not complex enough, and I also needed to control for overfitting. The next model had two layers; each contains Conv2D, Batchnorm, and MaxPool2d with ReLu and softmax as activation functions. I also added dropout of 0.5 to control for overfitting. This network showed better performance than the previous network.

Since the data missing images and labels for character J and Z, I tried to generate several blank images with pixel values of 0 for these two labels, but it didn't improve the results.

III. Results



Using a two-layered network, the accuracy and loss significantly increased/ decreased after the first epoch. After that, they fluctuated a little bit but remained consistently around 99% accuracy and 0.01 loss on both training and validation data. Using the same trained parameters on the hold out data resulted in the accuracy of 92.39%. Since the data is balanced, our group decided we don't need to use other metrics like F1 or Cohen-Kappa.

IV. Summary and Conclusions

Overall, I feel like everything I have learned in the class came together in this project. In the first exam using Keras, I only created a network using Sequential API. I learned to create model subclassing using Pytorch in the second exam. For this project, I built a network using model subclassing in Tensorflow 2.0 whose syntax is somewhat very similar Keras. All the parameters used in this project was mainly picked based on past experiences. Future work could focus on using grid search cv to select the best parameters.

V. Code Percentage

01-train_tensorflow_2.0.py: (56-10)/(56+32)*100 = 52.27%

02-predict_tensorflow_2.0.py: (23-18)/(23+12)*100 = 22%

VI. References

https://www.kaggle.com/avinashlalith/cnn-with-tensorflow-2-0
https://github.com/amir-jafari/Deep-Learning/blob/master/Tenflow_2/CNN/1_ImageClassification/example_MNIST.py
Tensorflow 2.0 documentations