ASL Alphabet Classification using K-Shot Learning

Jonah Buchanan, B.S
March 8, 2024

***Abstract*—The research presented in this paper explores the capabilities of a simple Siamese Network's ability to generalize to unseen classes. Various configurations of the vanilla Siamese Network are trained on the American Sign Language (ASL) Alphabet dataset. Each trained model's performance is compared in a K-shot setting on testing data whose classes were not included in the training set. All training and testing were developed using Apple's new Machine Learning framework, MLX. Overall, this work provides a starting point for developing K-shot learning algorithms that utilize Apple's silicon chips.**

Keywords—Siamese Network, K-Shot Learning, MLX

I. Introduction

Machine Learning has been at the forefront of computer vision tasks such as image classification, object detection, and semantic segmentation. The increase in publicly available large datasets and improved computational efficiency has helped to train models that constantly push the state of the art. Supervised Learning algorithms have leveraged the ability to efficiently process the large datasets to train their models. One major downside of Supervised Learning is that it is very costly to train a model due to computational resources and time. Also, Supervised Learning models often don't generalize to new data domains and thus very specific models are trained for a variety of problems. This project will focus on an approach to train an image classification model that can generalize to image classes not included in its training set. There are two major goals for this project: 1) Investigate techniques to improve an image classification model's ability to generalize to new classes and 2) Implement the project using Apple's new machine learning framework: MLX. Learning to train models that can generalize to new classes is important because in real life scenarios we constantly get new data and need to handle it without having to retrain our model. The MLX framework is important because it should be more performant on the Apple silicon chips and will be useful for not only training models but also integrating them with applications on Apple products.

II. Background

A. Siamese Networks

Traditional image classifiers utilize a supervised learning approach to predict a class for each image passed through the model. On the other hand, the goal of a Siamese Network is to embed each image into a higher dimensional latent space such that images of the same class are closer and images of different classes are farther apart [3]. Figure 1 demonstrates how the model works. The model is trained on image pairs where the goal of the model is to predict if the images belong to the same class. The first step in the model is to embed both images into a latent space using the same model. Typically, a Convolutional Neural Net (CNN) backbone is used to extract translationally and rotationally invariant features at different scales. Then a fully connected linear layer is used to output a feature vector. If the Euclidean distance between the two feature vectors is below some margin, then the model predicts that the images belong to the same class.

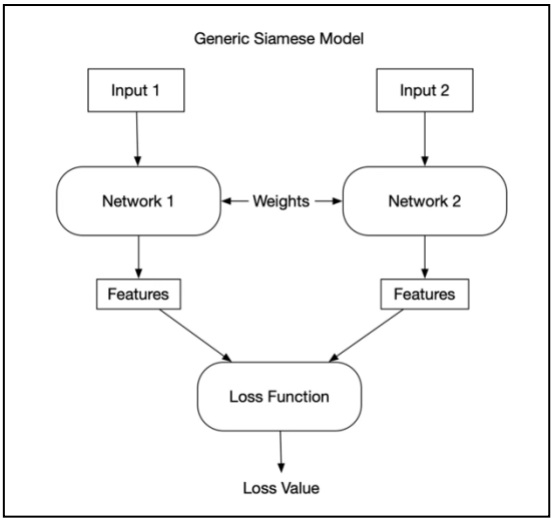

Figure 1: Siamese Network Architecture [2]

B. Contrastive Loss

$$L\left(W,\left(Y,\overrightarrow{X_1},\overrightarrow{X_2}\right)^i\right) = Y * D^2 + (1-Y) * \max(0, m-D)^2 \quad (1)$$

The Contrastive Loss described in Equation 1 is used as the loss function to calculate the gradients during backpropagation. *W* is the weights of the model, *Y* is the binary label indicating if the image pair is the same class, and *X* represents the feature vectors of the image pairs after getting passed through the model. For this project, *D* represent Euclidean Distance, but it should be noted that other distance metrics could be used. Finally, *m* represents the margin or maximum distance allowed between feature vectors that will let incorrect labels effect the loss.

The Contrastive Loss makes more sense for this problem compared to a typical Binary Cross Entropy because the Contrastive Loss rewards features that are closer together and penalizes features that are farther away. This helps the model to learn how to embed the input images in such a way that similar classes are closer together which is exactly what we want [2, 5, 6].

C. Dataset: ASL Alphabet

The dataset used in the project is the ASL Alphabet dataset from Kaggle [1]. The dataset contains 29 classes where each class has 3k images. The total dataset size is 87k images. Figure 2 displays example images from the dataset. The data provides a wide variety of lighting and angles to help the model learn the important features that distinguish each class. The data was split 70% training data, 20% testing data, and 10% validation data. In order to demonstrate the model's performance on classes it hasn't been trained on, the training and validation classes were not included during training.

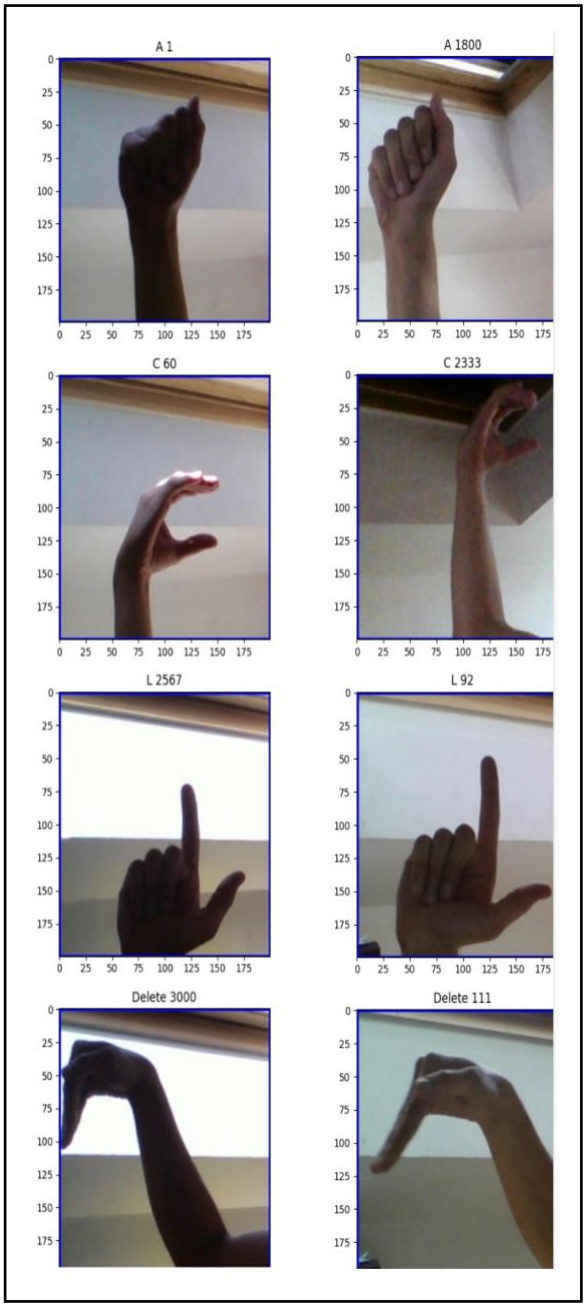

Figure 2: Example Images

A variety of data augmentation transformations were experimented with in order to prevent overfitting. The testing and validation sets only had normalization applied. The training set was trained with various combinations of the following transformations: random resized crop, horizontal flip, color jitter, gaussian blur, and normalization.

D. MLX

Apple's new Machine Learning framework, MLX, is a framework syntactically similar to PyTorch but utilizes an array structure similar to NumPy. MLX provides 2 major pieces of functionality that should improve model latency. The first is lazy computations. MLX only computes computations when necessary. This allows the framework to optimize repeated calculations. The second and biggest improvement is that MLX utilizes unified memory on the Apple silicon chips. This means that the CPU and GPU share the same memory. All other frameworks and hardware require a lot of copy operations between the CPU and GPU. With MLX, the unified memory significantly reduces a lot of the overhead copy operations [4].

III. Approach

A. Methodology

The overall approach for training the Siamese Network for image classification starts with training the backbone embedding model that performs binary classification to determine if an image pair belongs to the same class. The validation set is used during training to monitor the validation accuracy and to allow for early stopping to prevent overfitting. Then the trained model is tested on the testing set for varying K-shots. The testing set includes 6 classes not included in training but the support set used include images from all 29 classes. Each model was tested with a batch size of 1 to get an average inference speed. Also, a confusion matrix was generated for each model for each value of K tested.

B. Models

The architecture of the base backbone model tested is a simple CNN. The model has 3 layers of convolutions paired with RELU, Batch Normalization, and Max Pooling. There is an additional convolutional layer with RELU and Batch Normalization followed by a fully connected linear layer. This model has about 3 million trainable parameters. This model was trained with no additional data augmentations.

The second model trained had the same architecture of the first model but it included all of the transformations described in the data augmentation section. The third model added dropout layers to the base model to reduce overfitting and to help the model generalize to the test classes. The fourth model tested switched the order of the RELU and Batch Normalization layers. It also removed the horizontal flip data augmentation because that didn't make much sense in the context of the problem.

The hyperparameters tuned during training besides the model architecture included the learning rate, batch size, and transformation type. Due to time constraints, I was not able to perform a full grid search and only discuss the four models described above. 1 training epoch took 1 hour to iterate over the 60k training images with a batch size of 32 on a M1 chip. Thus, it took about a day to train each model.

IV. Results

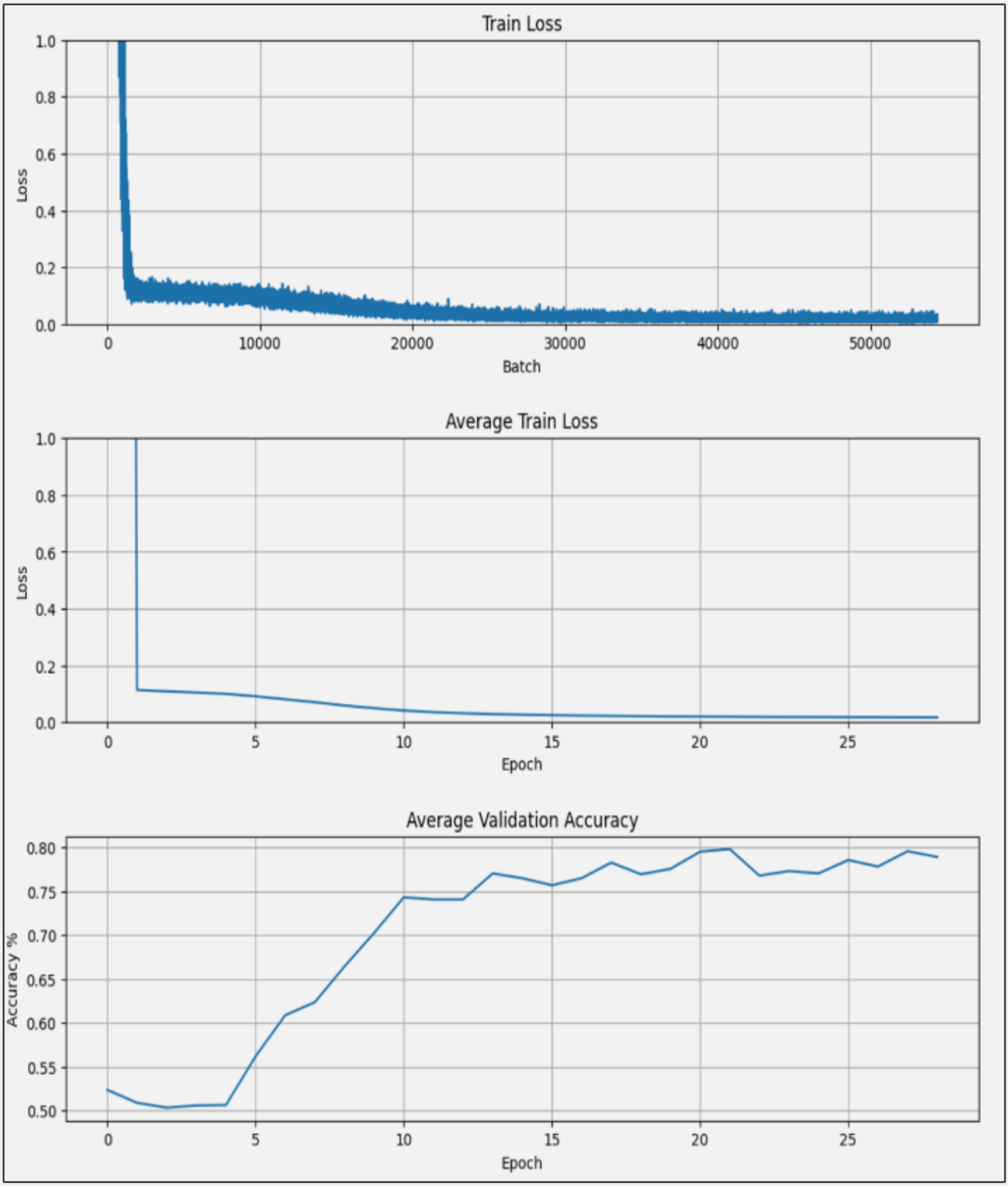

Figure 3: Model 3 Training Results

	K = 1	K = 2	K = 5	K = 10
Model 1	**47.53%**	**51.36%**	**55.25%**	64.81%
Model 2	16.51%	29.75%	31.31%	43.62%
Model 3	24.62%	44.73%	52.94%	**67.60%**
Model 4	30.24%	29.81%	47.74%	53.11%

Table 1: K-Shot Average Testing Accuracy

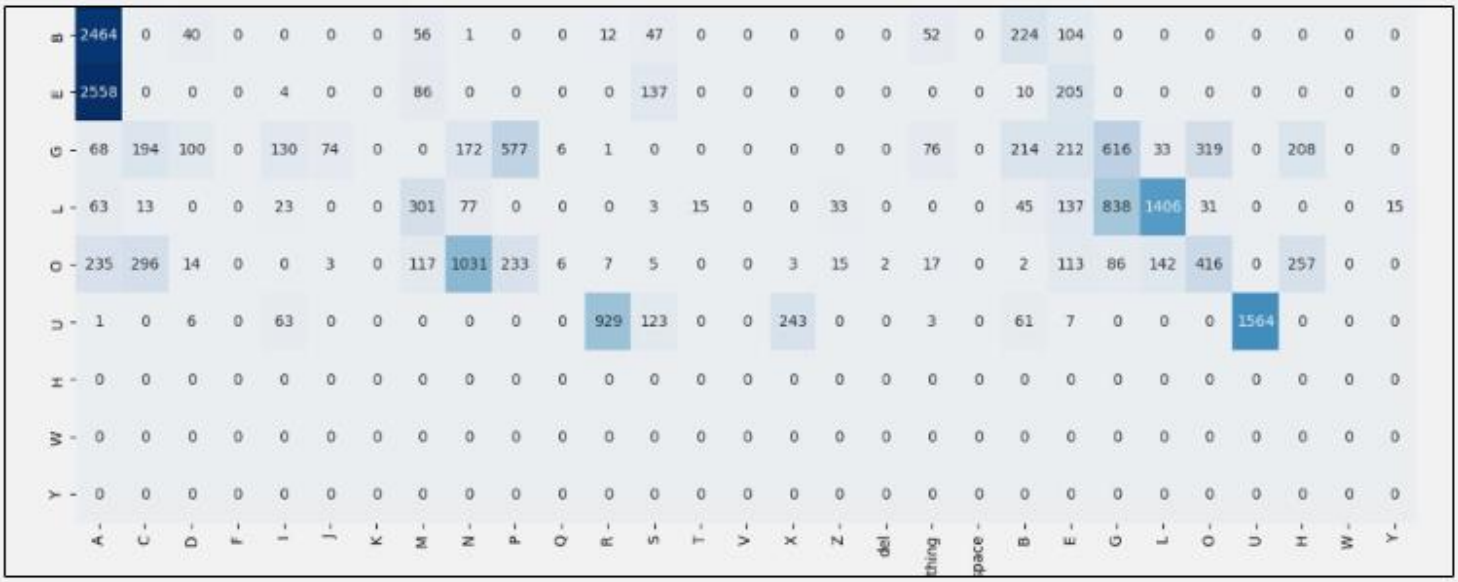

Figure 4: Model 3 Confusion Matrix K = 1

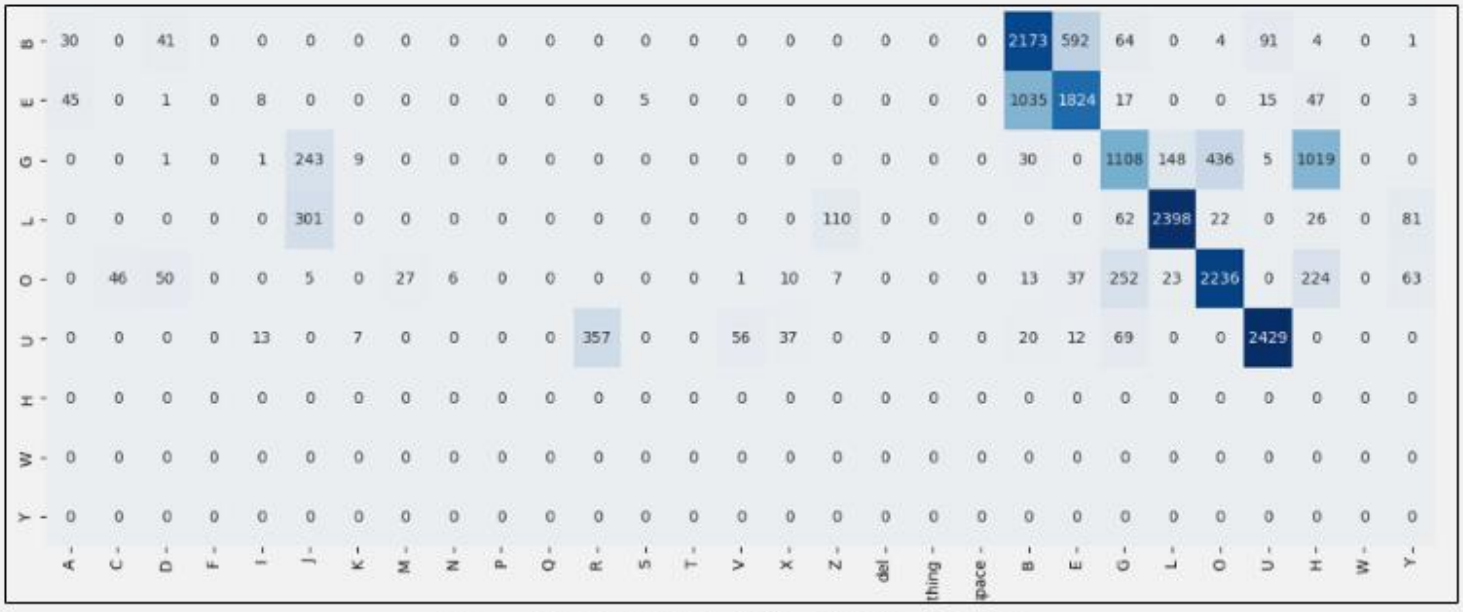

Figure 5: Model 3 Confusion Matrix: K = 10

V. Analysis

Table 1 displays the K-shot average accuracy for each model. As expected, as K increases, the model accuracy generally improves. Model 3 got worse from K = 1 to K = 2. Since the support set is chosen at random, it is possible that we get unlucky and select support images that don't get embedded close to other images of the same class. But as K increases, we increase our chances of having enough samples of the class's distribution which would improve our performance. Since the model architecture was the same for each test, the frames per second was consistently around 28. Note that since K was relatively small and that the support set feature vectors were precomputed, the FPS remained roughly the same. Also, it is important to note that as we use larger and larger values of K, we would be required to use more memory and eventually time to classify each image. My jupyter notebook crashed when I experimented with K = 100.

Another takeaway from Table 1 is that the base model with no data augmentation or dropout outperformed the other models when K <= 5. This was slightly surprising because this model didn't train to as low of a training loss as model 3. Also, without data augmentation and dropout, the model is more prone to overfitting. But model 3 outperformed model 1 when K = 10. This could show that model 1 just had better random draws compared to model 3.

Model 2-4 were attempts to reduce overfitting to help the model generalize to new classes. One key hyperparameter that I was surprised about was the horizontal flip data augmentation. I initially accidentally included the horizontal flip and removed it in model 4. But surprisingly the model seemed to perform better with it included. I believe this happened because although the horizontal flip doesn't make sense in the context of gesture recognition, it prevents the model from overfitting on the training data. The transformation was not a silver bullet as I tried training additional models, and sometimes the models did not learn anything. Just like all hyperparameter tuning, each hyperparameter is very brittle in impacting model convergence.

The model performance is definitely low but it is promising because a random guess with 29 classes has roughly a 3% chance of being correct. The model's performances range from 16% - 67% which demonstrates that the models are learning. What is even more impressive is that the testing data classes were not included in the training set. So the model was able to generalize to new classes with only a few shots of reference images.

The last takeaway I have from the tests come from Figure 4 and 5. They demonstrate the confusion matrix for model 3 when tested with K = 1 and K = 10 respectively. The test set includes the following classes: B, E, G, L, O, U. The x-axis of the confusion matrix represents the class predicted by the model. The y-axis represents the actual label for each image. As you can see, when K = 1, the model's predictions are all over the place with the most predicted class not matches the correct actual class. But when K = 10, we can see a distinct diagonal heat map indicating that the model was able to predict the correct class most of the time.

VI. Conclusion

Overall, I have demonstrated how to implement, train, and test a Siamese Network using the MLX framework for K-shot image classification. Also, I was able to demonstrate the model's ability to generalize to unseen classes. Future work includes testing different distance metrics within the loss function like cosine similarity, continued hyperparameter testing, and additional K-shot training on the unseen data instead of just testing.

References

[1] ASL Alphabet Dataset, https://www.kaggle.com/datasets/grassknoted/asl-alphabet/data

[2] B. Williams, "Contrastive Loss Explained", https://towardsdatascience.com/contrastive-loss-explaned-159f2d4a87ec

[3] G. Koch, R. Zemel, R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition," https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

[4] MLX Documentation, https://ml-explore.github.io/mlx/build/html/index.html

[5] M. Bekuzarov, "Losses explained: Contrastive Loss", https://medium.com/@maksym.bekuzarov/losses-explained-contrastive-loss-f8f57fe32246

[6] R. Hadsell, S. Chopra, Y. LeCun, "Dimensionality Reduction by Learning an Invariant Mapping," in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2, pp. 1735-1742, 2006, doi: 10.1109/CVPR.2006.100