Image Classification for American Sign Language

Author: Vishnu Manathattai

A&O C111 Final Report

Introduction

The purpose of this project is to explore the ability for machines to autonomously detect and decipher American Sign Language hand signage in the form of images (either through a video feed or pictures through a camera). Specifically, the problem that is being addressed by this project is the fact that majority of people do not know sign language, and this can pose as a potential communication barrier that should be solved, especially considering that the barriers between most languages are already solved (with the help of online translators, for example, google translate) but the same widespread application for American Sign Language is not widely used or really accessible.

To overcome this barrier, we employ Machine Learning techniques, specifically, in being able to take visual data (a human hand) and classify it into the alphabetic letter that it is signing. The dataset of choice for this task is the ASL Alphabet Image Dataset, which contains images of different hands signing the alphabet in ASL from different camera angles and lighting, along with its respective letter label. This therefore allows us to proceed using a supervised training technique, and with a clever loss system, can optimize the model to pair the image with its respective class.

My approach to solving this ASL classification problem was to employ the use of Convolutional layers, specifically in the usage of a Residual Neural Network, a model that combines the strengths of traditional CNNs with skip connections, tackling the vanishing gradient problem that is common in computer vision. In the end, I was able to train the model to 99% accuracy in classifying the signs, successfully proving the strength of Resnet in solving this image classification problem.

Dataset

The dataset as mentioned before is the ASL Alphabet Image Data, built by Akash Nagaraj, a researcher at the Brain Science Institute at Brown University. Before diving into the preprocessing of this set, it is valuable to understand the structure of the data. The images themselves are JPG images. The dimensions of these images are 200 x 200 x 3: 200 pixels in width and height, and 3 channels (Red, Green and Blue magnitudes). This 3 channel image representation is the widely accepted standard for image storage. The images themselves depict various signed letters (A-Z, the del character, space character, and nothing). For the “nothing” label the image simply does not have a hand in frame. As mentioned before, in order for a more general, adaptable dataset, it pictures different hands, with different distances from the camera, in different lighting. The structure of the dataset itself has these images in their respective class folders {‘A’, ‘B’, etc.} and there are 2998 such images per folder/class. With 29 total classes, this makes for a total train dataset of 86,942.

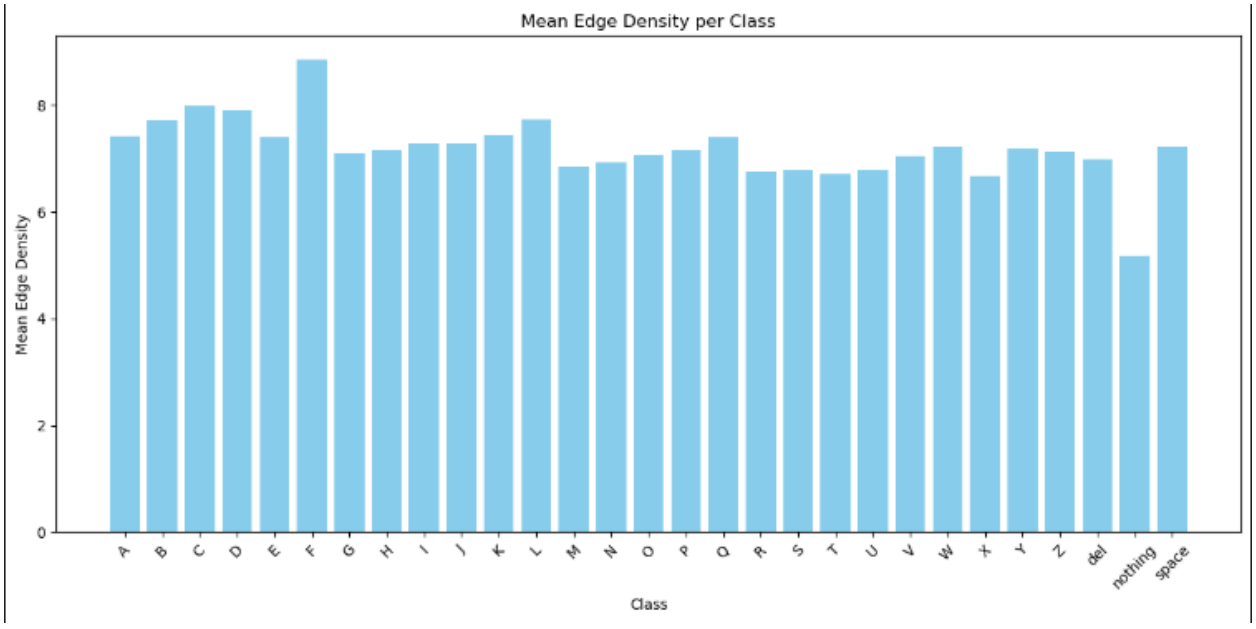
6-image sample of the dataset with their respective labels above

Onto the preprocessing for this dataset. The first consideration I undertook was the fact that Google Collab was unfit for storing all the data. Secondly, training 80K images in a reasonable amount of time is a difficult task when doing Machine Learning on CPU, as it does not implement sufficient parallelism or have specialized tensor cores. Therefore, I decided to train it on my local desktop, with my Nvidia RTX 3070 Ti GPU. In order to do this, I downloaded the data locally, then set up an environment with all the necessary packages and CUDA support to begin training on my GPU. This included installing a compatible distribution of Pytorch, and cuda drivers before I began testing moving tensors from CPU to GPU. With the environment configured, it was now onto processing the data. To do this, I wrote a Pytorch Dataset (which is essentially a class with a getitem, len, and init function defined). In the constructor for the class, I wrote a script to parse all the class directories, and append the tuple of imagepath, class label to a list, essentially giving a way to access all the data. Then, in the getitem method (which is what we use to retrieve a single sample), I used pytorch's read_image function that takes an image path and reads it into a tensor (a 3 x 200 x 200 matrix… the same dimensions from before). I then perform the operation of resizing the tensor to 256x256, and then normalizing the integer RGB values, which typically range from 0 to 255, to a float between 0 and 1. The purpose of this is by normalizing to between 0 and 1, we ensure very consistent input to the model, and therefore

the process of updating weights or computing gradients are done more smoothly, without being subject to large numerical gaps.

Let's briefly discuss how we took the semantic encodings of the classes (the letters) and changed it to numerical data. To do this, I did a simple one-hot encoding of the classes, giving the directories of the classes a number based on their alphabetical order, and therefore the label of these images in the dataset are represented by a number between 0-28 in the code rather than 'A' or 'space' characters.

For visualizing statistics about the dataset, there isn't too much one can do with statistics for image data, but one thing I implemented is using Canny edge detector to parse through a subset of the data, accumulating edge density to compare between classes.

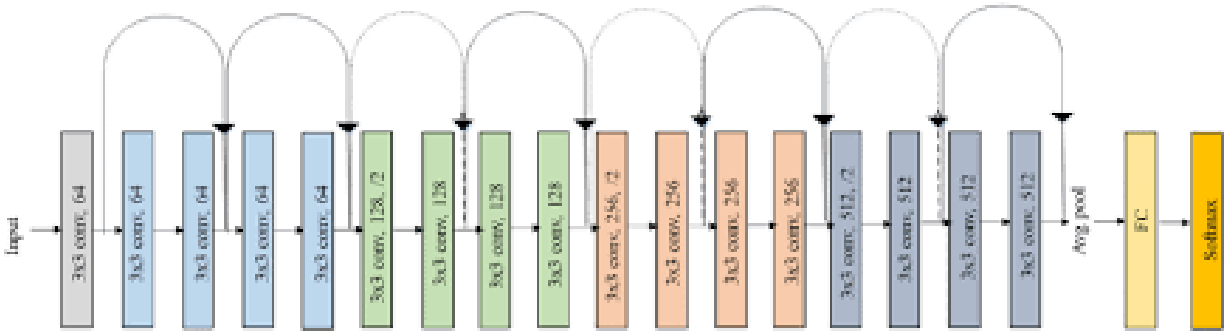


Mean Edge Densities Visualized for ASL Classification

The last part of the dataset is actually loading in batches to the training loop, and doing this on GPU. The way I set up the data loader, we only load the images into RAM when we call getitem (not when we instantiate the dataset, as at that point, it only has the paths)... making it quite convenient during training time. For loading batches, we utilize a Pytorch Dataloader object, taking in the custom Dataset object we made, and allowing us to call upon batches of 32 images. Then using the convenient pytorch .cuda() function, I can map the batch of 32 from CPU memory to GPU memory, and simply garbage collect it once the gradients have been updated for the batch, ensuring that despite the large memory overhead of the dataset, it is no problem when training.

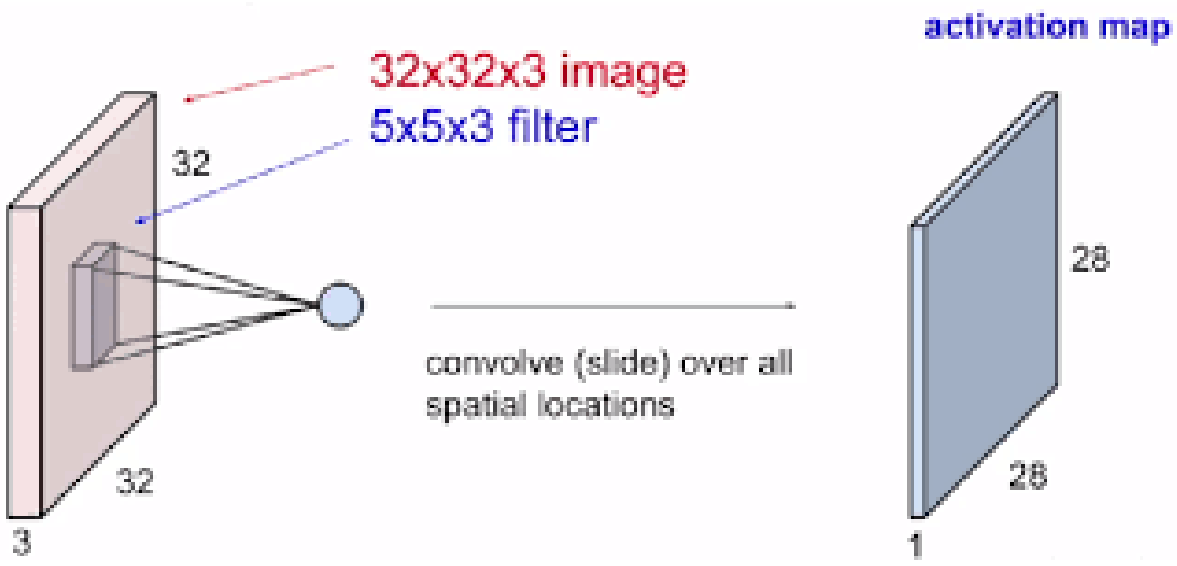
Model

The model used to fit the dataset and ultimately solve this problem is Resnet-18, an 18 layer neural network that gets its name from the use of “Residual Blocks”. These blocks are essentially 3 convolution layers that have a “skip connection” in the output. What this means is that the output of the residual block is not only the convolution output (after normalization and activation), but also summed with the input to the residual layer, allowing data to “bypass” or skip the convolution layers and be transmitted to further blocks.



Architecture of Resnet-18

It is also useful to give a description of the theory of convolution in discussing the model architecture. Convolution is simply an operation performed on the input of the previous layer, that preserves spatial information. Essentially, in fully connected (non-convolutional) architectures, if we had a 32x32x3 image for example, it would be flattened to a 3072 x 1 vector and multiplied by a weight matrix. However, with convolution, we specify a smaller filter, and slide it over the 32 x 32 x 3 image, computing element-wise dot products as we go.

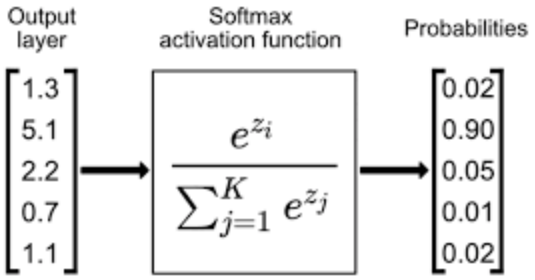


Visual Representation of Convolutional Filter

In the image above, with a 5x5x3 filter, and a stride of 1 (meaning we move the filter one pixel either horizontally or vertically when sliding across the image). We get (32 - 5)/ 1 + 1 = 28 width for our output activation map. By stacking convolutional filters and feeding the activation map of one to the input of the other, it organizes a hierarchy such that the first convolution layers focus on low-level tasks like detecting light and edges, while the later layers will take the edge data and extract higher-level insights about the semantics of the image.

Back to the Resnet architecture, by organizing these convolutional layers in the aforementioned residual blocks, we are both able to transmit these high-level extracted features to the later blocks, along with the low-level features from before, allowing later layers to have a global scope of the combination, allowing for high performance on image classification tasks. With more robust information, the Neural Network is able to differentiate between the classes of hand signs very well. One modification I made to Resnet is to change the final classification layer to output 29 classes (for our dataset) and so it outputs logits (unnormalized probability scores) for all classes. To determine what class we are actually predicting, we simply take the index of the max of these classes.

The last thing to talk about in this section is the loss and how we actually improve the model during training. The loss of choice is Cross-Entropy Loss, which essentially applies a softmax on the raw logits output of Resnet, and then tries to force the softmax output of the target class to 1.



Softmax Function Example

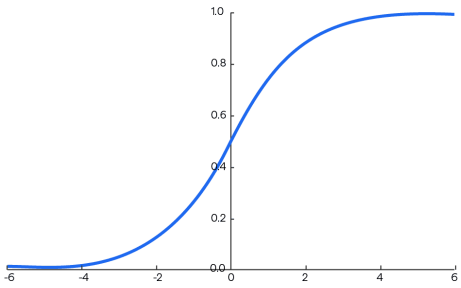
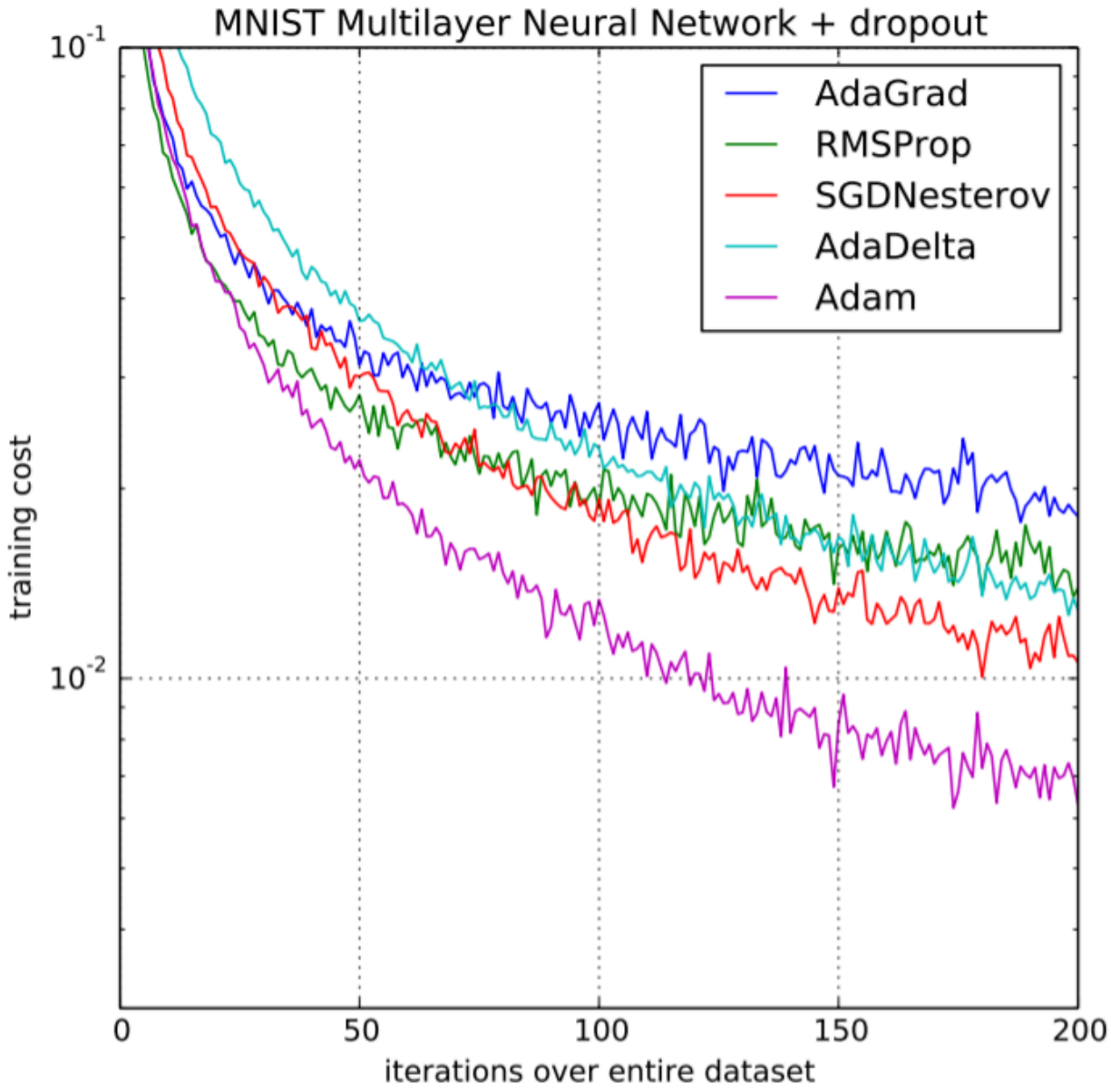


Illustration of Softmax Curve

What we mean by “forcing” the output of the target class to 1 is that the gradients will be computed with respect to this softmax function, and then when updating model parameters, we will take a step in the direction that will minimize the loss function (which will happen when we predict classes closer to the target). To take these “steps” towards optimal parameters, we utilize the Adam optimizer. The benefit of Adam vs something like gradient descent is that adam has an adaptive learning rate, preferring to learn faster in the initial stages and slower when we approach finer-grained tuning of the parameters, overall offering much faster convergence.



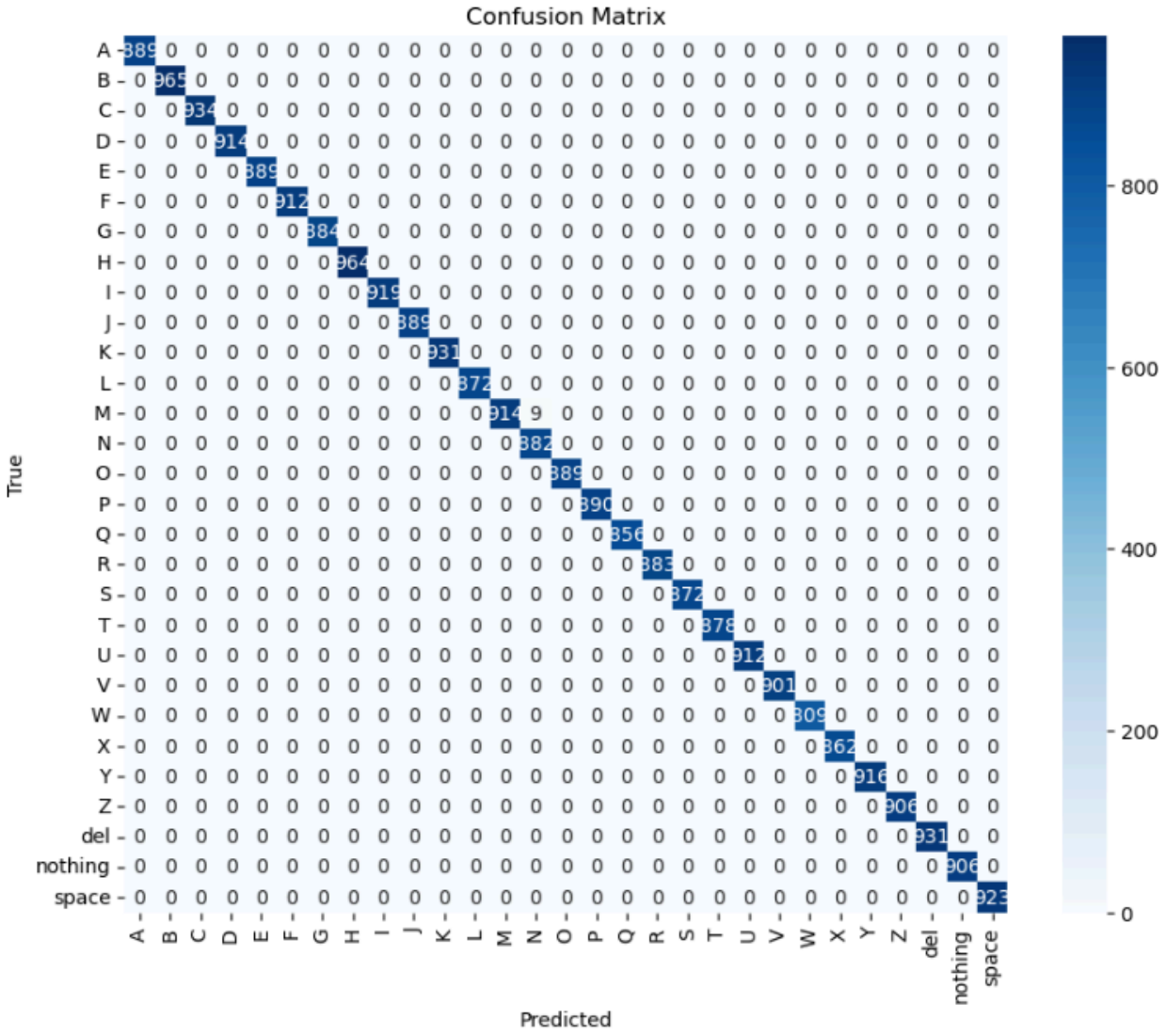
Adam Performance vs Other State of the Art Optimizers

Results

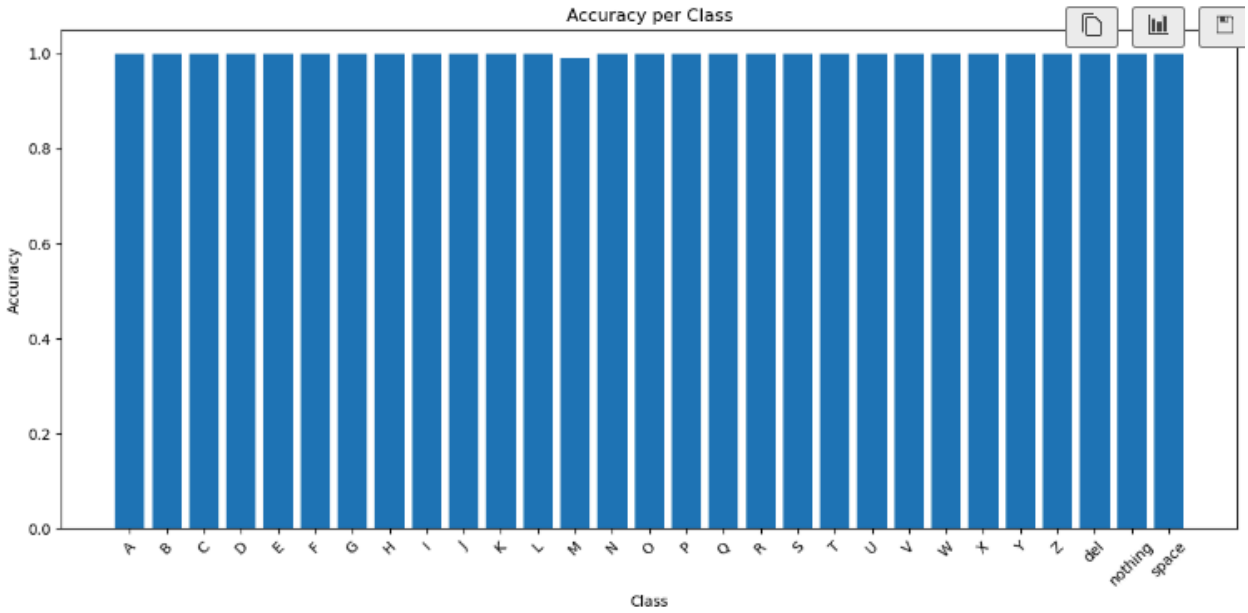
The results of training were quite good with 98.25% accuracy after one epoch and 99.63% after two epochs. However, it is important to note that just measuring accuracy on the train set is not sufficient because there is no guarantee that it will generalize to high accuracy on new, untrained data. Therefore, when I set up training, I purposefully split the 80K images in the data into 70% train and 30% in a strictly "test" dataset that could be used to evaluate the model after training. On the test dataset, the model had an accuracy of 99.96%, which suggests that our training process didn't overfit to the data and our results are both good and generalizable.

	precision	recall	f1-score	support
A	1.00000	1.00000	1.00000	889
B	1.00000	1.00000	1.00000	965
C	1.00000	1.00000	1.00000	934
D	1.00000	1.00000	1.00000	914
E	1.00000	1.00000	1.00000	889
F	1.00000	1.00000	1.00000	912
G	1.00000	1.00000	1.00000	884
H	1.00000	1.00000	1.00000	964
I	1.00000	1.00000	1.00000	919
J	1.00000	1.00000	1.00000	889
K	1.00000	1.00000	1.00000	931
L	1.00000	1.00000	1.00000	872
M	1.00000	0.99025	0.99510	923
N	0.98990	1.00000	0.99492	882
O	1.00000	1.00000	1.00000	889
P	1.00000	1.00000	1.00000	890
Q	1.00000	1.00000	1.00000	856
R	1.00000	1.00000	1.00000	883
S	1.00000	1.00000	1.00000	872
T	1.00000	1.00000	1.00000	878
U	1.00000	1.00000	1.00000	912
V	1.00000	1.00000	1.00000	901
W	1.00000	1.00000	1.00000	809
X	1.00000	1.00000	1.00000	862
Y	1.00000	1.00000	1.00000	916
Z	1.00000	1.00000	1.00000	906
del	1.00000	1.00000	1.00000	931
nothing	1.00000	1.00000	1.00000	906
space	1.00000	1.00000	1.00000	923
accuracy			0.99966	26101
macro avg	0.99965	0.99966	0.99966	26101
weighted avg	0.99966	0.99966	0.99966	26101

Sklearn Classification Report on Test Data w/5-decimal precision



Test Data Confusion Matrix

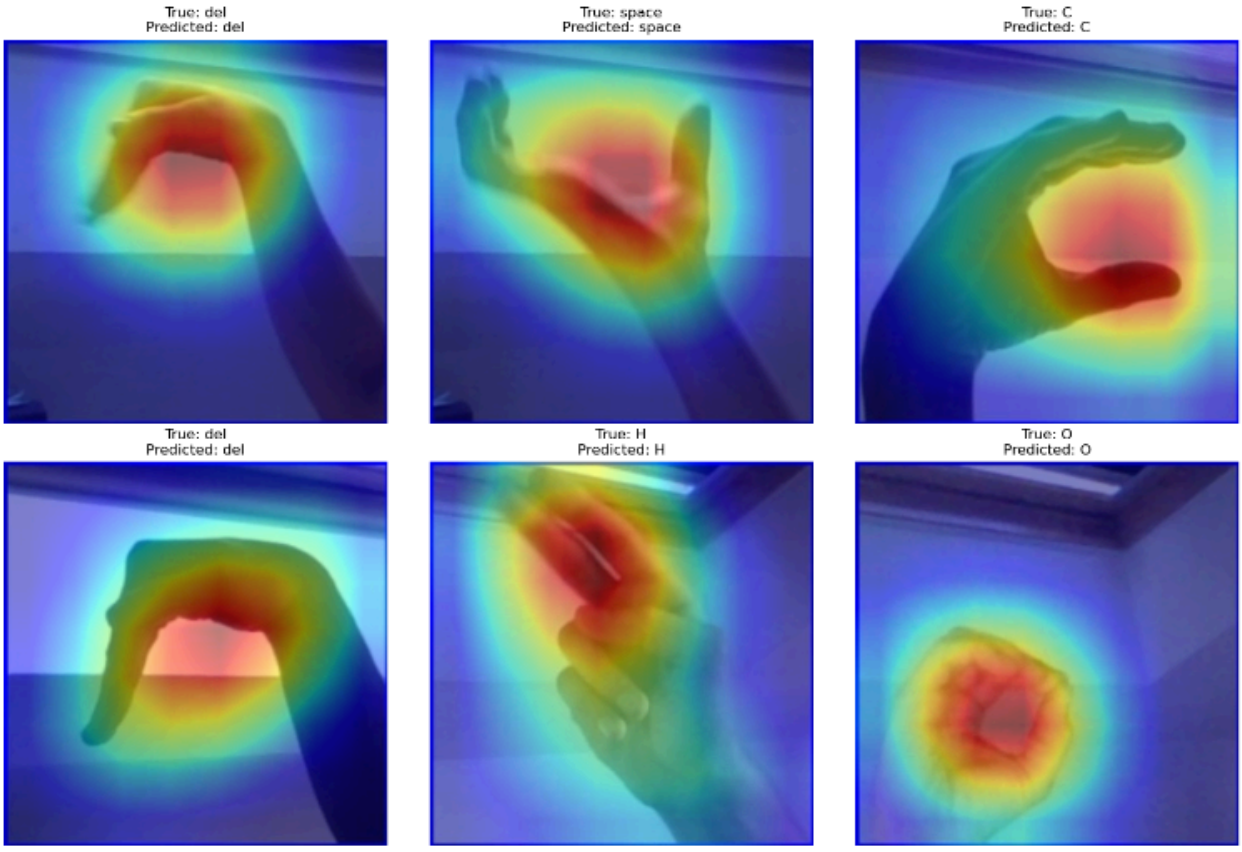


Accuracy by Class on Test Data

Discussion

The high accuracy on unseen data demonstrates that the Resnet-18 model is strong at classifying ASL hand signs from camera images, through various different angles, lighting, and distances. It is clear that via Resnet's ability to extract low-level and high-level features through residual blocks and the usage of Convolutional Layers, which are expert in dealing with spatial data, it is able to provide quite good accuracy on this dataset.

In order to provide more concrete description of what the neural network is actually doing under the hood, I employed a technique called Class Activation Mapping described by the paper: Learning Deep Features for Discriminative Localization authored by Bolei Zhou, the professor in my CS163 course at UCLA. The paper describes a process to see what parts of an input image are most activated by a neural network solution, allowing the user to extract key insights into how the model determines the differences between classes. In the proposed method, we go into the final convolutional layer and "steal" the activation map provided by the filter prior to it being passed to the layer that provides predictions for every class. I ended up implementing this myself for the Resnet model trained on the ASL dataset to get insights into what the model is looking for when trying to differentiate between the different classes.



Output of My Class Activation Mapping applied to ASL Model

The areas with the dark red is where there is the most activation, whereas going towards the blue is less/no activation. The results are quite informative: it shows that the activation occurs primarily in the gesture part of the image, and is able to form differences based on the shape of the hand. For example in the ‘O’, the heatmap is concentrated in the ‘O’ shape being made by the hand, whereas with the ‘del’ character, it is focused on the curvature of the hand (showing how the network learns different properties based on the class).

Therefore, we can see that not only is Resnet strong in classification based on the results showing strong statistical evidence in favor of its strength, but also from a qualitative point of views, in which the model focuses on the differences between images that we as a human also extract (focusing on the shapes of the hands rather than something like the background).

Conclusion

In conclusion, we can see that Resnet, and convolution in general, allows for great results in properly classifying camera-taken pictures depicting the ASL alphabet. The variety in the dataset in terms of hands being used and various camera-related randomness allowed for strong training of the model and extremely generalizable results that could be implanted in any modern

recognition application. Furthermore, via Class Activation Mapping, we ensured the semantic output is sensible and similar to that of humans, further showing the applicability of Resnet for computer vision tasks in general. For future considerations, if a high-quality dataset becomes available, we could extrapolate this project to not only the ASL alphabet, but the entire ASL dictionary, as ASL has varying signs for not just letters, but words. This would likely require more hardware overhead as we have 80K images for 29 labels, but with 30K+ words that are commonly used in English this dataset would become extremely large. With the lazily loading into RAM framework I implemented however, the code would be able to scale without much problem. Overall, classification of ASL is definitely a problem in the scope of the powerful CNNs, and can be integrated into modern applications that help those who use ASL exclusively communicate with American English speakers who, for the most part, do not know ASL.

References

[1] Akash, N. (2018, April 22). Asl alphabet. Kaggle.
https://www.kaggle.com/datasets/grassknoted/asl-alphabet
[2] Belagatti, P. (2024, March 11). Understanding the softmax activation function: A comprehensive guide. SingleStore.
https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/
[3] Catalog-free modeling of galaxy types in deep images - massive dimensional reduction with neural networks | Astronomy & Astrophysics (A&A). (n.d.).
https://www.aanda.org/articles/aa/full_html/2021/08/aa40383-21/aa40383-21.html
[4] GeeksforGeeks. (2024, March 20). What is Adam Optimizer?
https://www.geeksforgeeks.org/adam-optimizer/
[5] Softmax function: Advantages and applications: BotPenguin. BotPenguin AI Chatbot maker. (n.d.-a). https://botpenguin.com/glossary/softmax-function
Softmax function: Advantages and applications: BotPenguin. BotPenguin AI Chatbot maker. (n.d.-b). https://botpenguin.com/glossary/softmax-function
[6] Zhou, B., & Khosla, A. (n.d.). Learning deep features for discriminative localization.
http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf