Sign Language Classification

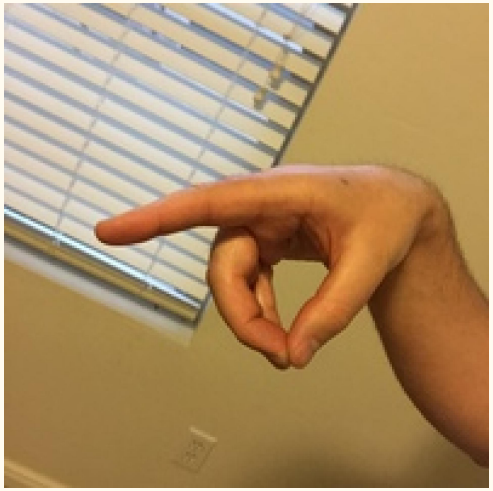
Joshua Wilbur, Amir Seidakhmetov

Problem to solve

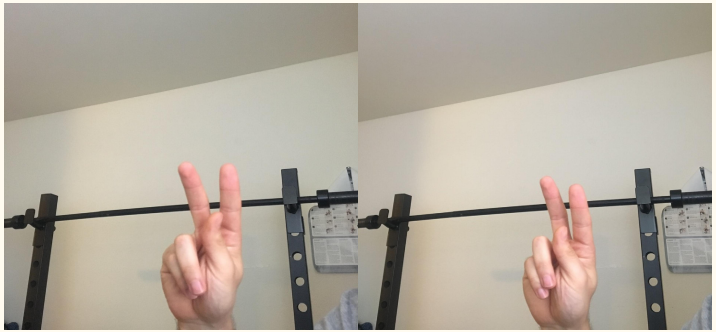
- Many people are unable to interpret American Sign Language (ASL), thus it is often challenging to effectively communicate with deaf people without an interpreter.
- Some people who become deaf later in their life often have initial difficulty transitioning to ASL.
- While papers and companies have created solutions, there isn't a "best method" for solving this problem.

Training Dataset

- 4.56GB dataset from Kaggle
- 29 classes of data
    - Each letter of the alphabet, plus signs for space, delete, and nothing (such as a wall)
- Training subset consists of ~7500 images for each class.
- Test subset consists of 28 test images.
    - Per the database owner, the test subset is small because it encourages use of real world images
- Images differ in hand orientation, position, background, lighting, and presence of objects or human faces.
- Images also vary by the shape and position of the fingers.



Training image for letter “P”



Training images for letter “K” with different finger positions

Data Preprocessing

- Had to split training data into training (80%) and validation (20%)
  - Didn't use "test" dataset that came with the database, only had 28 images
- Dataset is diverse and has multiple sizes of images
- Used ImageDataGenerator to preprocess the data and create generators
  - All images resized to (200, 200, 3)
  - Used shear and zoom to improve model generalization
- Batch size of 64

Model Overview

- Fine tuned VGG19 base model with ImageNet weights preloaded
- Total trainable parameters: 9,614,365
- Six layers placed at the output of the VGG19 model for feature extraction
- Three of these layers were fully connected (F.C.) layers
  - First layer (512 units): ReLU activation, 9,437,696 parameters
  - Second layer (256 units): ReLU activation, 131,328 parameters
  - Above two F.C. layers have L1 bias regularization, improved model convergence
  - Final layer has 29 units and softmax activation to classify image into 1 of 29 categories
- Batch normalization used between layers to improve training efficiency
- 40% dropout used before second F.C. layer to fight overfitting
- Adam optimizer used with 0.0014 learning rate
- Categorical crossentropy loss function

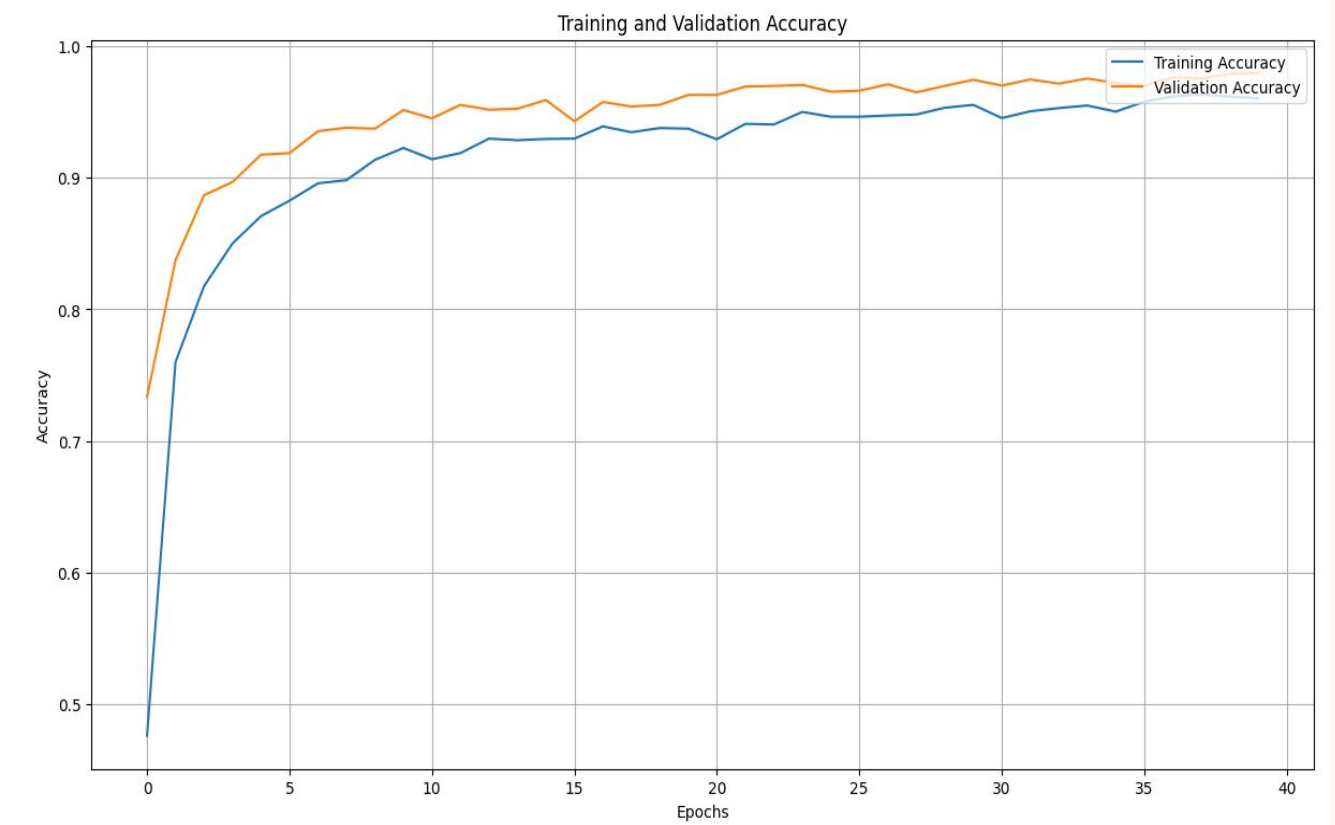
Why VGG19?

- Started by using EfficientNetV2S
- Switched model to ResNet101 then VGG19 to compare
  - ResNetV2 models (50, 101 and 152) performed poorly
- Found that VGG19 was most efficient in training compared to the others
- Table below compares the three models

Model Name	**Avg. Epoch Run Time**	**Loss (10 Epochs)**	**Accuracy (10 Epochs)**
VGG19	70 seconds	0.2716	91.55 %
ResNet101	73 seconds	0.3183	89.43 %
EfficientNetV2S	68.5 seconds	0.3969	86.96 %

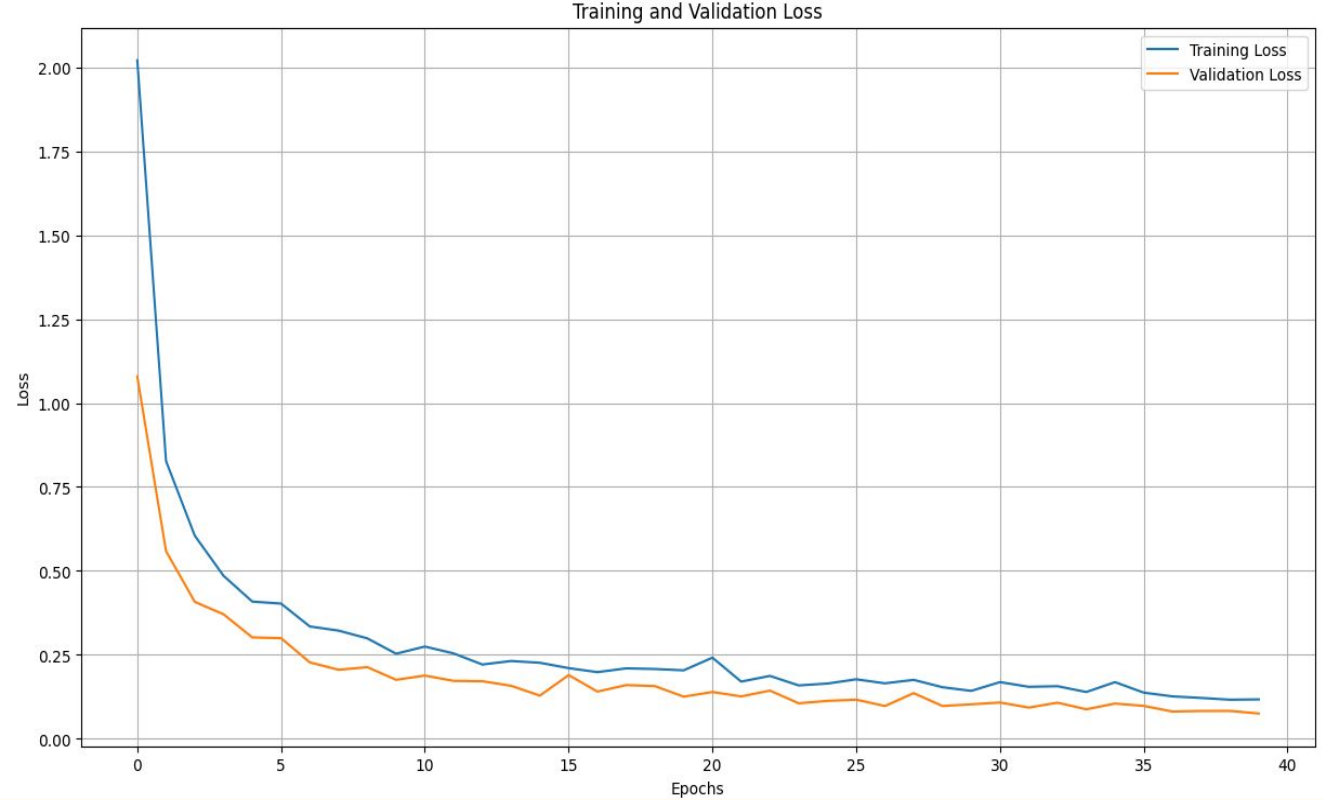
Model Performance, Accuracy Results

Accuracy	Best Epoch
Training	96.19 % (38)
Validation	97.85 % (40)



Model Performance, Loss Results

Loss	Best Epoch
Training	0.1251 (40)
Validation	0.0734 (40)



Encountered Challenges

- Certain image augmentation parameters significantly slowed training
  - Removing the horizontal flip and rotation parameters greatly improved the model performance
  - Model had bad overfitting issues until this was modified
- Loading dataset into Colab took a while to figure out
  - Stack Overflow helped this, linked the page in references
- Splitting training and validation data
- Less is more when it comes to F.C. layer depth

Overfitting Concerns

- Want model to work for diverse use cases, not just on training data
- Although the dataset is diverse, there are images that appear very similar
- Concerns that training and validation datasets are too similar
- Dropout increased testing performance as it reduces reliance on specific neurons
- Utilized independent database for testing, more on this later

Testing Dataset

- 24MB dataset from Kaggle
- Set is subdivided into training and test data
- Both subsets consist of 30 images for every letter of the alphabet, plus pace, delete, and nothing
- Images differ in hand orientation, position, background, and lighting
- Images also vary by the shape and position of of fingers.

Test images for letter “S”

Performance on Testing Dataset

- Ran the model for 5 epochs to get a sense of how it would perform.
- Best accuracy on testing dataset: 77.01 %
- Best loss on testing dataset: 1.034
- These results show that our model isn't perfect, however it is able to identify hand gestures in novel situations.

Possible Extensions and Improvements

- More training using other datasets, including videos
- Implementing ability to upload an image/video for identification by the model and outputting detected text
- Build a CNN instead of fine tuning a VGG19 base model
    - This could help create a less complex and more portable model, may reduce overfitting
- Optimizing the model to identify ASL in real-time for videos or live camera feed
    - This may require an additional program or model to track and screenshot hand gestures
    - An overfitted model would perform bad in this use case

Summary

- Our model performs well, but took a while to get to this point
  - 96% Accurate in training dataset
  - 72% Accurate in testing dataset
- Overfitting isn't a major issue with the model, but is present
- Many lessons learned through creating this model

Databases and Links

- Training dataset: https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset/data
- Testing dataset: https://www.kaggle.com/datasets/danrasband/asl-alphabet-test/data
- https://stackoverflow.com/questions/74037120/how-to-download-kaggle-dataset

Thank You