ASL Fingerspelling Recognition

Daqian Dang
Rajkumar Conjeevaram Mohan
Andy Christian

DATS 6303: Deep Learning
Instructor: Dr.Amir Jafari

Introduction

- Motivation
  1. ASL is primary language means of communication for DHH individuals in the US
  2. Fingerspelling is an aspect of ASL that uses 26 hand configurations to denote alphabet letters
  3. Bridge the communication gap between the hearing and deaf or hard of hearing communities
  4. Promote accessibility and inclusivity for deaf and hard of hearing communities
- Objectives
  1. Build proof of concept for ASL character sign (fingerspelling) recognition
  2. Our current focus on ASL fingerspelling serves as a fundamental building block, allowing us to create a strong foundation for more advanced applications
  3. Provide starting point for future efforts to produce live ASL sign reading and transcription to text or conversion to speech
- Methodology
  1. Preprocess the data to ensure consistency, normalize images, and augment data
  2. Create baseline CNN model
  3. Conduct optimization of model to produce best results on validation data
  4. Test model on test data to check ability to generalize on new data

Description of Data Set

Source: Kaggle

Link: ASL Fingerspelling Images (RGB & Depth)

Type: Color images pre-cropped to focus on sign hand

Number of Subjects: 5 (people who contributed signs to the database)

Number of Observations: 131,662 (RGB images, Depth images not used)

Number of Classes: 24 (J and Z require motion, so are not included)

Class Distribution: min 5235, max 6220

Experimental Setup 1

How to use data to train and test the model?

- The data was divided into 3 sections, train, validation, and test.
- Validation data was used to check the loss and accuracy to see if training was successful
- Test data was used at the end to evaluate the performance of the model on unseen data

How to implement the network and judge performance?

- Cross entropy loss was used to judge whether to keep training or not, as this measures how “confident” the model is in the class it has predicted
- The validation set’s prediction accuracy was used to determine when to save the best model, as correct predictions was the ultimate goal
- Macro F1 was used to evaluate the performance on the test set to get a measure of both how precise the model’s predictions were and how sensitive it was to finding each letter.

Experimental Setup 2

How to determine the size of the minibatches?

- Minibatch sizes were found via experimentation
- Smaller minibatches had more randomness in their convergence which seemed to lead to better predictions despite having higher loss

How to determine the learning rate?

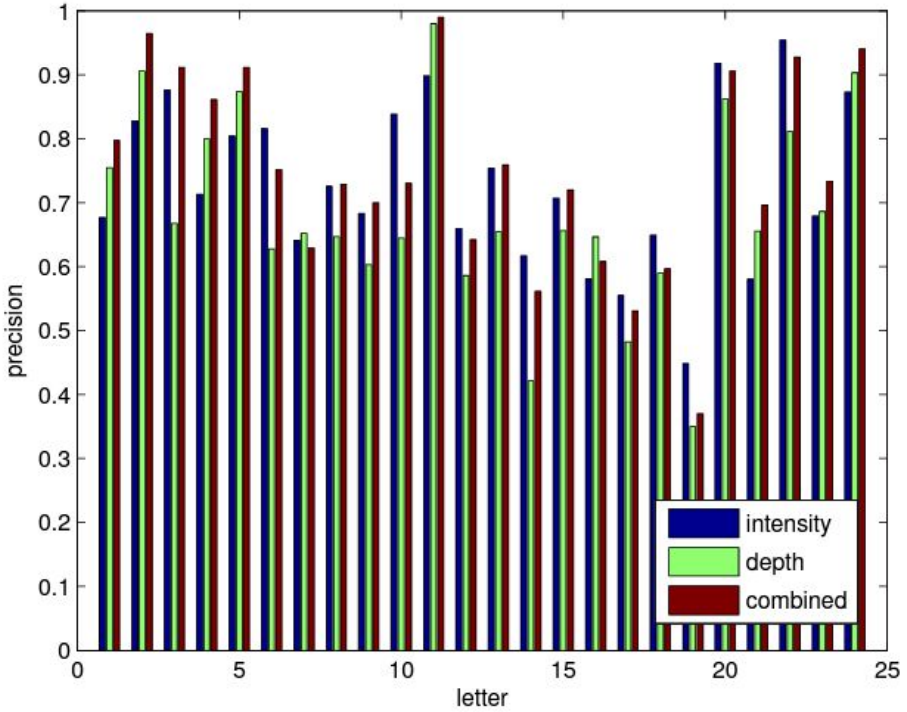
- Learning rate was found via experimentation
- Larger learning rates seemed to lead to better predictions dispute having higher loss
- Learning rate scheduling was used throughout the training to drop the learning rate when improvement wasn't being achieved

How to avoid overfitting?

- Early stopping was used, which measured the loss on the validation set and stopped training if there was not improvement in the loss after a certain number of training epochs
- Experiment: L2 regularization was added inside the Adam optimizer to penalize incorrect predictions

Results of Initial Research from 2011

- The initial data set and research were conducted in 2011
- The researchers used a random forest model
- Results averaged around ~0.7 precision for each letter
- Developments since then allowed us to use a CNN in our approach, which produced better results



Description of Deep Learning Network and Training Algorithm

INITIAL ARCHITECTURE

-3 Convolution Layers

- 16, 32, 64 filter maps
- 5x5, 3x3, 3x3 kernels
- 2x2 MaxPool

-1 Linear Layer

-Relu activation functions used

-Softmax used on output layer

TRAINING ALGORITHM

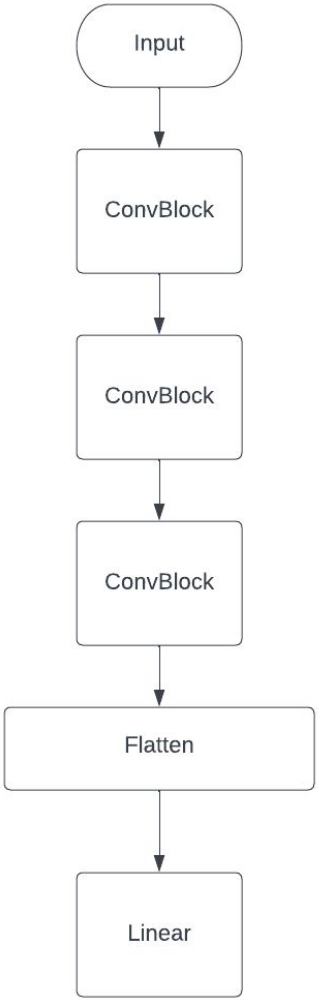
-Training Data

- Feed forward through network
- Calculated cross entropy loss, accuracy
- Calculate gradient, update weights

-Validation Data

- Feed forward, calculate loss, accuracy
- Check for loss improvement otherwise activate early stopping
- Check for accuracy improvement to save best model

Final Architecture



ConvBlock

BatchNormalization2D
Conv2D
ReLU
MaxPool2D

Results: Raj's Optimizations

Data Loader:

- MiniBatch size 100
- Prefetch factor 100

Hyperparameters:

- 5x5, 3x3, 3x3
- Random seed of 3388
- Augmentation unchanged
- 25 Epochs

Results for best model:

- Test macro F1: 0.9888569
- Loss: 0.09125

Other optimizations tried:

- Batch size: 500, Prefetch: 30
- Different group size for Conv
- Depthwise Separable Convolution

Comments:

- Large-scale model only yields subtle improvement.
- Convolution without padding

Training

```
Epoch 19 | Train Loss 0.28535, Train Acc 1.0 - Validation Loss 0.07458, Validation
 Acc 1.0
**Model Saved**
Epoch 20 | Train Loss 0.26402, Train Acc 0.9791666865348816 - Validation Loss
 0.08641, Validation Acc 1.0
early stopping validation loss check activated, early stopping patience remaing: 29
Epoch 21 | Train Loss 0.24875, Train Acc 1.0 - Validation Loss 0.09125, Validation
 Acc 1.0
early stopping validation loss check activated, early stopping patience remaing: 28
**Model Saved**
```

Testing

```
Estimated Total Size (MB): 6.07
----------------------------------------------------------------
None
Starting testing...
Test Set F1: 0.9888569680669316
```

Results: Andy's Optimizations

Hyperparameters for best model:

- Learning rate of 0.001
- 64 minibatch size
- 5x5 kernal
- Dropout (0.3) after every layer
- L2 regularization of 0.001 in Adam
- Random seed of 3388
- Augmentation unchanged
- 100+ epochs of training

Results for best model:

- Test macro F1: 0.9874
- Loss: 0.11 - 0.13
- Prediction "confidence": ~0.91-0.93

Other optimizations tried:

- Learning rate: 10^-4, 10^-5
- Minibatch size: 512, 1000
- Kernel size: 3x3, 4x4
- Dropout: None, 0.2
- L2 regularization: None, 0.01
- Extra convolution layers
- Mimic VGG19 architecture

Comments:

- Larger batch sizes resulted in very low loss but the test macro F1 was only ~0.9
- Complex architecture was not as effective as a simple model

Results: Daqian's optimizations

Hyperparameters for best model:

- Learning rate: 0.001
- Batch size: 512
- Kernel size: 3x3
- Layers N: 3
- Dropout of 0.3 implemented in each layer
- Batch Normalization implemented in each layer
- Epoch N: 30
- Optimizer: ADAM with L2 regularization: weight decay of 0.001

Results for best model:

- Epoch: 29
- Train loss: 0.137
- Train accuracy: 0.950
- Validation loss: 0.145
- Validation accuracy: 0.966
- Test F1 score: 0.965

Comment:

Tried to use pretrained model such as EfficentNet, and noticed the computation time was extremely slow, and the complex model was not effective as the simple model was shown.

Summary and Conclusion

- A CNN was more effective than the random forest approach the researchers used in 2011 for their initial study with this data set
- A simple architecture was most effective in solving the problem
- Proof of concept in using CNN for ASL interpretation
- For future experiments, we wish to work on real-time ASL detection to text, an end-end hybrid architecture.

References

Bowden, R., Pugeault, N. (2011). Spelling It Out: Real–Time ASL Fingerspelling Recognition. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. https://doi.org/10.1109/ICCVW.2011.6130290

Geislinger, V. (2020). ASL Fingerspelling Images (RGB & Depth) (Version 2)[Data Set]. Kaggle. https://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out

Thanks for watching!
Any questions?