A Deep Learning Approach to American Sign Language Character Classification

By Arvind Shankar, Ryan Leavitt, James Dika, Stephen Young

Data Acquisition and Preprocessing

- Data Acquisition:
  - Class data – 1844 images
  - MNIST data – added 13,013 more images
- Preprocessing:
  - Integer label encoding
  - Gray scaling provided data
    - Reduces dimensionality by factor of 3
  - Standardizing grayscale data
    - This data has mean 0, unit variance
  - Training and Test Split
- Other preprocessing methods performed poorly in validation
  - PCA, HOG, and LDA

Model Selection

- In Lab 3 we trained and validated these classifiers:
  - LDA, KNN, Decision Trees, Random Forests, SVM, and MLPs
- The highest accuracy classifier was an MLP with 74% test accuracy
- Then we investigated deep learning options for the classifier and decided to go with a Convolutional Neural Net for our model

	Algo	Accuracies
0	LDA	0.308943
1	KNN	0.634146
2	Decision Tree	0.346883
3	Random Forest	0.430894
4	SVM	0.092141
5	MLP	0.739837

Our CNN

- Following online tutorials and other resources, we created a CNN class with the following parameters kept constant:
  - Output activation function
    - Softmax activation function since this is a multiclass problem
    - Softmax activation outputs a distribution of probabilities – the index with the highest probability is the predicted label
  - Objective function
    - Literature also revealed that minimizing the Cross-Entropy loss function works well along with the Softmax function
  - Learning Rate
    - Initialized to 0.001 based on PyTorch documentation
  - Gradient Descent Optimizer
    - Used Adam since it is the most common method used
    - This results in an adaptive learning rate

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

https://pytorch.org/docs/stable/nn.html

Tuning CNN Parameters

- We did testing to find optimal values for the following parameters:
  - Number of convolutional layers
  - Kernel size
  - Number of inputs to first linear layer
  - Training set
  - Batch size
  - Number of epochs

- Testing methodology:
  - Train CNNs with different values for the parameter in question, other parameters kept constant
  - Score CNNs on validation partition of the grayscale provided dataset – representative of the real test set
  - Keep parameter values which maximize validation accuracy

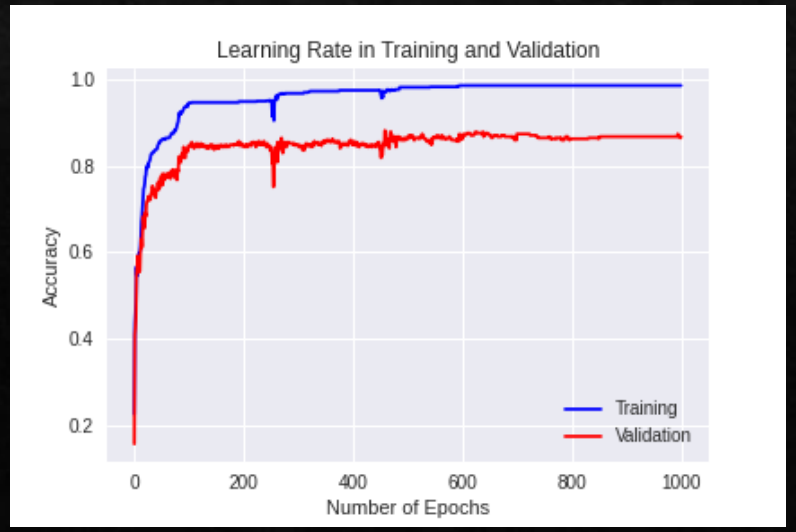
Tuning The CNN cont.

- Kernel Size
  - Tested kernel sizes 3 and 5, found that 3x3 was better
- Convolutional Layers
  - Tested 3 and 4 convolutional layers – 3 layers performed much better
- Number of Inputs to First Linear Layer
  - Reducing number of outputs in last convolutional layer lead to much worse performance
- Training Data
  - 3 datasets – grayscale provided set, MNIST combined with grayscale set, grayscale & standardized provided set
  - MNIST data didn't help – makes sense since the MNIST set is not representative of the real test data
  - Standardized dataset performed very poorly
  - The model performed best with data with just the grayscale provided dataset

Tuning The CNN cont.

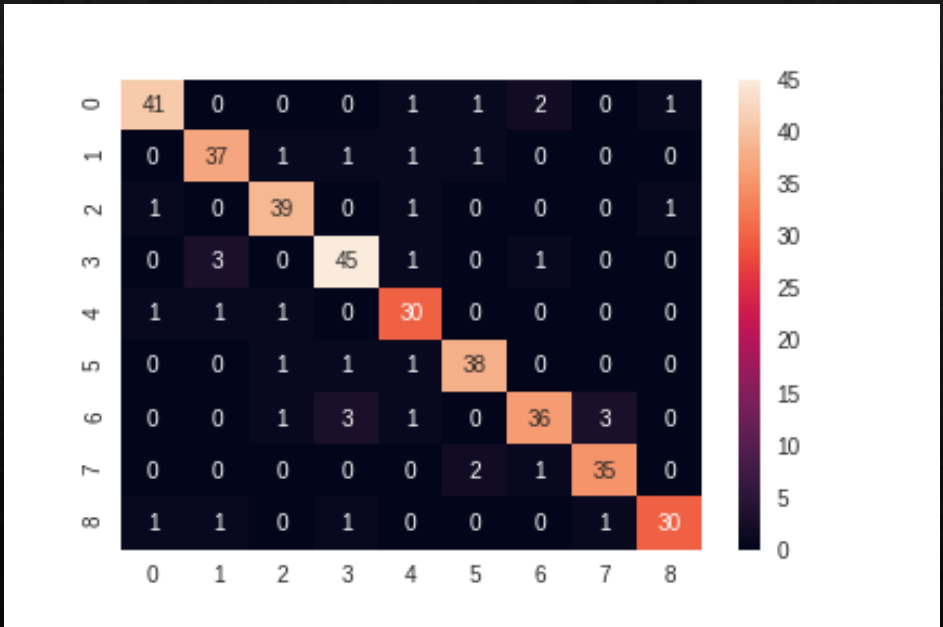
- Batch Size and Epochs
  - Tested 19 different combinations with small to large values for both parameters – found that batch size 128 and 1000 epochs gave us the best model
  - More epochs might lead to increased accuracy, but execution time was already very long – this made it difficult to do more tests

- Examining training and validation accuracy over 1000 epochs:



Results

- The best model we obtained had an overall accuracy of 89.7%
- The confusion matrix and the accuracy for each letter for this model are shown below:



Letter	Accuracy	Letter	Accuracy
A	89.1%	**F**	92.7%
B	90.2%	**G**	81.8%
C	92.9%	**H**	92.1%
D	90.0%	**I**	88.2%
E	90.9%		

Possible Improvements

- There were a few parameters we didn't vary that could potentially be optimized through further testing:
  - Loss function
    - There are others besides cross-entropy that work with multiclass problems
  - Optimizer
    - AdaGrad and RMSProp are alternative adaptive learning rate optimizers
- Could also have researched alternative preprocessing methods for feature extraction generation besides the ones we tried in Lab 2

Conclusions

- We made a CNN with 3 convolutional layers using 3x3 kernels, 2D pooling layers, and 2 linear layers
- We determined most parameters for our model through incremental experimentation while the constant parameter choices were based on the literature and resources
- Training this CNN on the grayscale provided data with a batch size of 128 for 1000 epochs, we used the Adam optimizer to minimize cross-entropy loss
- Testing this CNN on a realistic test set, we got an overall accuracy of 89.7%, with 6 out of the 9 letters being classified with at least 90% accuracy