AMERICAN SIGN LANGUAGE IMAGE CLASSIFICATION

PROJECT REPORT

Ishita Jain, Jyothi Ganji, Mihir Gadgil

Introduction

Communication is a method of exchanging ideas whether it is through speech or gesture. The Sign language is a visual language used by people with speech and hearing disabilities for communication in their daily conversation activities. One example is the transcription of a lecture, where a translator is using sign language to convey the lecture to hearing impaired audience. Since there are only a small and finitely many sign symbols possible, we believe this might be an easier problem to solve than audio transcription.

Gestures can be classified into two types: Static and Dynamic, Ashok K Sahoo (2006). Lots of study has been done on sign language recognition and various approaches in the last few years. Few approaches namely HMM (Hidden Markov Model), ANN (Artificial Neural Network) C Vogler (1999), and SVM (Support Vector Machine), Ashish S. Nikam (2016) are proposed for classification. We focus on recognition of the signs for letters. We are using Convolutional Neural Networks for this purpose. The image has to be preprocessed before passing it to the model. PyTorch is our choice for the framework. Multiple rounds of training and testing, and some practical limitations showed us that a pre-trained and then fine tuned ResNet model works best for us.

Data

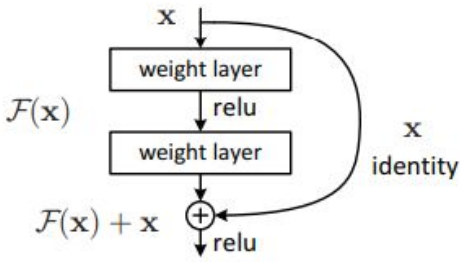
The data comes from Kaggle (https://www.kaggle.com/datamunge/sign-language-mnist). The data is provided as a csv file, with the first column as the alphabet encoded as a number (0 to 25) and the rest of the 784 columns give pixel values of a 28 x 28 grayscale image. The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion).

There are no images for the letters J (9) and Z (25) since the gestures for those letters involve a motion. The dataset is not balanced between classes; E has the least number of samples (957) and R has the most (1294). The original hand gesture image data represented multiple users repeating the gesture against different backgrounds.

We added another 48 images - one for each letter, hands of two people with white background - to the dataset since we are planning to use the model in real time.

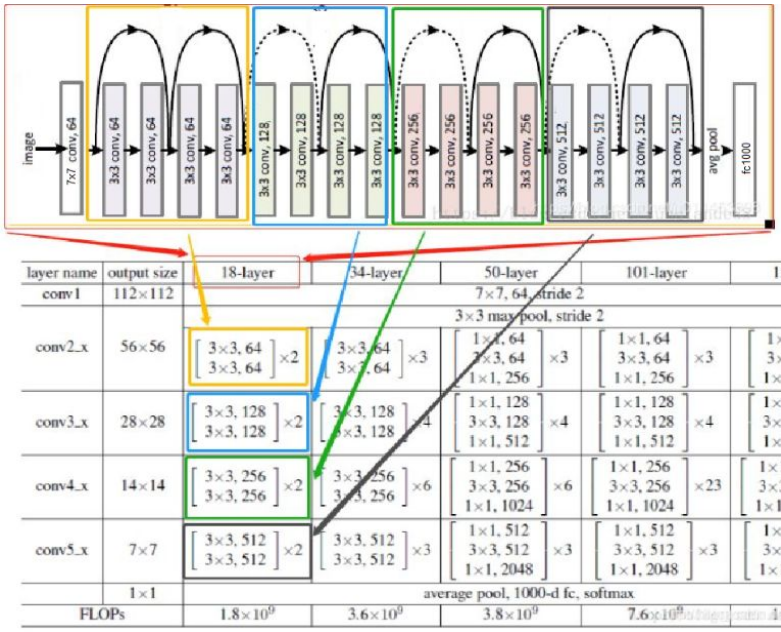
Model

We investigated the use of a custom built model and some popular models pre-trained on the ImageNet (2012) dataset. After testing out performances of the models and considering practical limitations, we have settled on using pre-trained ResNet 18 as the final model. As Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015) demonstrated, a **residual neural network** (**ResNet**) is implemented with double or triple layer skips that contain nonlinearities (ReLU) and batch normalization in between. Skipping helps to speed up learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through.



Residual Learning: a building block (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015)

The last layer of the model is changed to a Linear layer of 26 neurons, which correspond to the 26 classes in the data.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet18 Architecture (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015)

Experimental Setup

At the time of the first run, the data is read as a pandas DataFrame. The pixel values are reshaped into 28 x 28 arrays and stored in a npy formatted file for faster loading on subsequent runs. We also calculate the weights of the classes to account for the imbalance in the data. These weights are used in the loss function.

When training, the “training” dataset is split into a training and a validation set. 70% images are used as train and 30% as validation, and the relative distribution of the classes is preserved in both the training and validation sets. These sets are then wrapped in the MyDataset class which extends the pytorch Dataset class. The pytorch DataLoader then iterates over this class to load the images for training. MyDataset performs some transformations on the images before returning them for training. The transforms we have used are:

Resize: The ResNet model expects RGB images of size 224 x 224, this transform upsamples the 28 x 28 images for this purpose.

Horizontal Flip: The image is flipped horizontally at random (50% chance). Since either the left or the right hand can be used for sign language, this is a required step for data augmentation.

Rotate: The image is rotated by random angle between (-10, 10) degree. This is to account for the fact that a hand’s orientation won’t always be exactly the same in real life scenarios.

Colour Jitter: The image’s brightness, contrast, saturation and hue are randomly changed. This accounts for varying lighting conditions.

Replicate: This replicates the image to convert it into an RGB image, since the model expects it to be so.

Normalization: Normalizes the image with the mean and standard deviation expected by the model.

We are using Cross Entropy Loss as the training performance metric. And stochastic gradient descent with momentum and learning rate scheduling is used for optimizing the loss. The training images are loaded in batches of size 128 by 8 worker processes, this corresponds to 163 batches. The validation images are loaded in batches of size 512 by 8 worker processes, this corresponds to 40 batches. The base learning rate of the optimizer is 0.01. The learning rate scheduler reduces it by a factor of 10 if the loss on the validation set doesn’t decrease for 3 consecutive steps. The training occurs for a maximum of 100 epochs or stops early if the validation loss doesn’t decrease for 10 consecutive epochs. This addresses overfitting. The best model is saved for further evaluation and use.

The best model is then evaluated on both the training and the test data, separately. We are using the mean of macro averaged F1 score and Cohen's Kappa as the evaluation metric (same as Exam 1). If the score on the test data is comparable to the score on training data, it is an indication of the model generalizing well.
The model can also be used for making prediction on real time data using the predict script. It involves accessing a smartphone camera via internet and using the image captured by it for prediction. The image also goes through some preprocessing before being passed to the model. If the model is also able to perform well on real time data, it is a very strong indication that the model has generalized.

Results

We tried training multiple custom-built architectures with many different parameters, like various optimizers, learning rates, batch sizes and regularization techniques. They managed to achieve good performance on the non augmented training data, but never performed well on either the test data or when we augmented the data.
The pre-trained, and then fine tuned, ResNet 18 model, achieves 100% accuracy and 1.0 on both the F1-score and Cohen's Kappa metrics, both on the training data and the testing data. Its performance on real time data also shows promise but requires proper performance measurement.

Summary and Conclusion

The fine tuned models clearly outperform the custom models. We have two guesses regarding the failure of custom models. Either there isn't enough data to learn from or the learning happens to be very slow in the early epochs. Which means the custom model could train on the data, but it would take a very long time; which we cannot handle practically.
The fine tuned model performs much better on real time data than the custom models, but their performance still isn't as good as the scores on testing data would suggest. We have taken steps to avoid overfitting, but it is still possible that the model is overfitting the training data. Another possibility is that we don't know how the training images were preprocessed. If our preprocessing on real time images doesn't match the preprocessing of the training images, it can also lead to worse performance.

References

1. https://www.kaggle.com/datamunge/sign-language-mnist.
2. Sahoo, Ashok K., Gouri Sankar Mishra, and Kiran Kumar Ravulakollu. "Sign language recognition: state of the art." ARPN Journal of Engineering and Applied Sciences 9.2 (2014): 116-134.
3. Vogler, Christian, and Dimitris Metaxas. "Parallel hidden markov models for american sign language recognition." Proceedings of the Seventh IEEE International Conference on Computer Vision. Vol. 1. IEEE, 1999.
4. Nikam, Ashish S., and Aarti G. Ambekar. "Sign language recognition using image based hand gesture recognition techniques." 2016 Online International Conference on Green Engineering and Technologies (IC-GET). IEEE, 2016.
5. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.