Real-time American Sign Language Recognition with Convolutional Neural Networks

Brandon Garcia
Stanford University
Stanford, CA
bgarcia7@stanford.edu

Sigberto Alarcon Viesca
Stanford University
Stanford, CA
salarcon@stanford.edu

Abstract

A real-time sign language translator is an important milestone in facilitating communication between the deaf community and the general public. We hereby present the development and implementation of an American Sign Language (ASL) fingerspelling translator based on a convolutional neural network. We utilize a pre-trained GoogLeNet architecture trained on the ILSVRC2012 dataset, as well as the Surrey University and Massey University ASL datasets in order to apply transfer learning to this task. We produced a robust model that consistently classifies letters a-e correctly with first-time users and another that correctly classifies letters a-k in a majority of cases. Given the limitations of the datasets and the encouraging results achieved, we are confident that with further research and more data, we can produce a fully generalizable translator for all ASL letters.

1. Introduction

American Sign Language (ASL) substantially facilitates communication in the deaf community. However, there are only ~250,000-500,000 speakers which significantly limits the number of people that they can easily communicate with [1]. The alternative of written communication is cumbersome, impersonal and even impractical when an emergency occurs. In order to diminish this obstacle and to enable dynamic communication, we present an ASL recognition system that uses Convolutional Neural Networks (CNN) in real time to translate a video of a user's ASL signs into text. Our problem consists of three tasks to be done in real time:

1. Obtaining video of the user signing (input)
2. Classifying each frame in the video to a letter
3. Reconstructing and displaying the most likely word from classification scores (output)

From a computer vision perspective, this problem represents a significant challenge due to a number of considerations, including:

- Environmental concerns (e.g. lighting sensitivity, background, and camera position)
- Occlusion (e.g. some or all fingers, or an entire hand can be out of the field of view)
- Sign boundary detection (when a sign ends and the next begins)
- Co-articulation (when a sign is affected by the preceding or succeeding sign)

While Neural Networks have been applied to ASL letter recognition (Appendix A) in the past with accuracies that are consistently over 90% [2-11], many of them require a 3-D capture element with motion-tracking gloves or a Microsoft Kinect, and only one of them provides real-time classifications. The constraints imposed by the extra requirements reduce the scalability and feasibility of these solutions.

Our system features a pipeline that takes video of a user signing a word as input through a web application. We then extract individual frames of the video and generate letter probabilities for each using a CNN (letters *a* through *y,* excluding *j* and *z* since they require movement). With the use of a variety of heuristics, we group the frames based on the character index that each frame is suspected to correspond to. Finally, we use a language model in order to output a likely word to the user.

2. Related Work

ASL recognition is not a new computer vision problem. Over the past two decades, researchers have used classifiers from a variety of categories that we can group roughly into linear classifiers, neural networks and Bayesian networks [2-11].

While linear classifiers are easy to work with because they are relatively simple models, they require sophisticated feature extraction and preprocessing methods to be successful [2, 3, 4]. Singha and Das obtained accuracy of 96% on 10 classes for images of gestures of one hand using Karhunen-Loeve Transforms

[2]. These translate and rotate the axes to establish a new coordinate system based on the variance of the data. This transformation is applied after using a skin filter, hand cropping and edge detection on the images. They use a linear classifier to distinguish between hand gestures including thumbs up, index finger pointing left and right, and numbers (no ASL). Sharma et al. use piece-wise classifiers (Support Vector Machines and k-Nearest Neighbors) to characterize each color channel after background subtraction and noise removal [4]. Their innovation comes from using a contour trace, which is an efficient representation of hand contours. They attain an accuracy of 62.3% using an SVM on the segmented color channel model.

Bayesian networks like Hidden Markov Models have also achieved high accuracies [5, 6, 7]. These are particularly good at capturing temporal patterns, but they require clearly defined models that are defined prior to learning. Starner and Pentland used a Hidden Markov Model (HMM) and a 3-D glove that tracks hand movement [5]. Since the glove is able to obtain 3-D information from the hand regardless of spatial orientation, they were able to achieve an impressive accuracy of 99.2% on the test set. Their HMM uses time-series data to track hand movements and classify based on where the hand has been in recent frames.

Suk et al. propose a method for recognizing hand gestures in a continuous video stream using a *dynamic Bayesian network* or DBN model [6]. They attempt to classify moving hand gestures, such as making a circle around the body or waving. They achieve an accuracy of over 99%, but it is worth noting that all gestures are markedly different from each other and that they are not American Sign Language. However, the motion-tracking feature would be relevant for classifying the dynamic letters of ASL: *j* and *z*.

Some neural networks have been used to tackle ASL translation [8, 9, 10, 11]. Arguably, the most significant advantage of neural networks is that they learn the most important classification features. However, they require considerably more time and data to train. To date, most have been relatively shallow. Mekala et al. classified video of ASL letters into text using advanced feature extraction and a 3-layer Neural Network [8]. They extracted features in two categories: hand position and movement. Prior to ASL classification, they identify the presence and location of 6 "points of interest" in the hand: each of the fingertips and the center of the palm. Mekala et al. also take Fourier Transforms of the images and identify what section of the frame the hand is located in. While they claim to be able to correctly classify 100% of images with this framework, there is no mention of whether this result was achieved in the training, validation or test set.

Admasu and Raimond classified Ethiopian Sign Language correctly in 98.5% of cases using a feed-forward Neural Network [9]. They use a significant amount of image preprocessing, including image size normalization, image background subtraction, contrast adjustment, and image segmentation. Admasu and Raimond extracted features with a Gabor Filter and Principal Component Analysis.

The most relevant work to date is L. Pigou et al's application of CNN's to classify 20 Italian gestures from the ChaLearn 2014 Looking at People gesture spotting competition [11]. They use a Microsoft Kinect on full-body images of people performing the gestures and achieve a cross-validation accuracy of 91.7%. As in the case with the aforementioned 3-D glove, the Kinect allows capture of depth features, which aids significantly in classifying ASL signs.

3. Approach and Methods

3.1. Classifier Development

3.1.1 Transfer Learning

Our ASL letter classification is done using a convolutional neural network (CNN or ConvNet). CNNs are machine learning algorithms that have seen incredible success in handling a variety of tasks related to processing videos and images. Since 2012, the field has experienced an explosion of growth and applications in image classification, object localization, and object detection.

A primary advantage of utilizing such techniques stems from CNNs abilities to learn features as well as the weights corresponding to each feature. Like other machine learning algorithms, CNNs seek to optimize some objective function, specifically the loss function. We utilized a softmax-based loss function:

$$Loss = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{i,y_i}}}{\sum_{j=1}^{C} e^{f_{i,j}}}\right) \quad (1)$$

$$f_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}} \quad (2)$$

$\mathrm{N} = \text{total number of training examples}$
$\mathrm{C} = \text{total number of classes}$

Equation (2) is the softmax function. It takes a feature vector z for a given training example, and squashes its values to a vector of [0,1]-valued real numbers summing to 1. Equation (1) takes the mean loss for each training example, x_i, to produce the full softmax loss.

Using a softmax-based classification head allows us to output values akin to probabilities for each ASL letter. This differs from another popular choice: the SVM loss.

Using an SVM classification head would result in scores for each ASL letter that would not directly map to probabilities. These probabilities afforded to us by the softmax loss allow us to more intuitively interpret our results and prove useful when running our classifications through a language model.

3.1.2 Transfer Learning

Transfer Learning is a machine learning technique where models are trained on (usually) larger data sets and refactored to fit more specific or niche data. This is done by recycling a portion of the weights from the pre-trained model and reinitializing or otherwise altering weights at shallower layers. The most basic example of this would be a fully trained network whose final classification layer weights have been reinitialized to be able to classify some new set of data. The primary benefits of such a technique are its less demanding time and data requirements. However, the challenge in transfer learning stems from the differences between the original data used to train and the new data being classified. Larger differences in these data sets often require re-initializing or increasing learning rates for deeper layers in the net.

3.1.3 Caffe and GoogLeNet

We employed Caffe, a deep learning framework [12], in order to develop, test, and run our CNNs. Specifically, we used Berkeley Vision and Learning Center's GoogLeNet pre-trained on the 2012 ILSVRC dataset. This net output three different losses at various depths of the net and combines these losses prior to computing a gradient at training time.

3.2. General Technique

Our overarching approach was to fine-tune a pre-trained GoogLeNet. Our data is composed exclusively of hands in 24 different orientations, while the ILSVRC data is composed of 1000 uniquely different objects or classes. Even though the ILSVRC data has some classes that are quite similar to each other (e.g. various breeds of dogs), our data was comprised of the same object merely positioned and oriented in different ways. Considering the stark differences between our data and the data that GoogLeNet was pre-trained on, we decided to test the effectiveness of altering a variety of the pre-trained weights at different depths. We amplified learning rates by a factor of ten and completely reset weights in the first one, two or three layers of the net using Xavier initialization.

3.3. Developing our pipeline

In order to obtain images of the user signing in real-time, we created a web application that is able to access a native camera on a laptop through the browser solely using HTML and JavaScript. This was done using an API created by the World Wide Web Consortium (W3C). Image capture rate was a massive problem we struggled with. We looked to balance network request speeds with computation speeds of our neural network (discussed below). The latter proved to be the primary bottleneck, forcing us to settle on an image capture rate of one frame per second. Increasing our capture rate created larger delays in giving the user feedback than we were satisfied with.

Our web application sends images to our server one by one. Each time, the server classifies the image and presents probabilities for each letter. It keeps a running cache of classified images. When it feels confident about the sign being made by the user, it records the top-5 most likely letters based on the cache. It then clears the cache, and lets the user know to move on to the next letter.

Once the user has indicated that they're finished signing, the top-5 letters for each position in the spelled word are passed to a custom unigram language model based on the Brown Corpus. The model takes into account substitution costs for similar-looking letters that are often confused by the classifier as well as the probabilities for the top-5 letters at each position in the word. Using these, single word unigram probabilities and a handful of other heuristics, it returns the optimal word to the user.

4. Datasets and Features

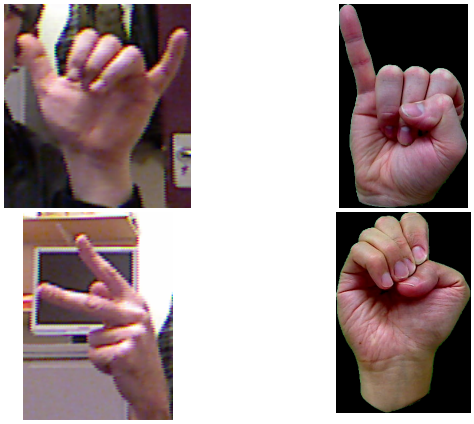


Fig. 1. Dataset examples. Left: Surrey University. Right: Massey University. Top left: *y*. Bottom left: *k*. Top right: *i*. Bottom right: *e*.

4.1. Dataset Description

The ASL FingerSpelling Dataset from the University of Surrey's Center for Vision, Speech and Signal Processing is divided into two types: color images (A) and depth images (B). To make our Translator accessible through a simple web-app and laptop with a camera, we chose to only use the color images. They are close-ups of hands that span the majority of the image surface (Fig. 1).

Dataset A comprises the 24 static signs of ASL captured from 5 users in different sessions with similar lighting and backgrounds. The images are in color. Their height-to-width ratios vary significantly but average approximately 150x150 pixels. The dataset contains over 65,000 images.

The Massey University Gesture Dataset 2012 contains 2,524 close-up, color images that are cropped such that the hands touch all four edges of the frame. The hands have all been tightly cropped with little to no negative space and placed over a uniform black background. Coincidentally, this dataset was also captured from five users. The frames average approximately 500x500 pixels.

Since there was little to no variation between the images for the same class of each signer, we separated the datasets into training and validation by volunteer. Four of the five volunteers from each dataset were used to train, and the remaining volunteer from each was used to validate. We opted not to separate a test set since that would require us to remove one of four hands from the training set and thus significantly affect generalizability. Instead, we tested the classifier on the web application by signing ourselves and observing the resulting classification probabilities outputted by the models.

4.2. Pre-processing and data augmentation

Both datasets contain images with unequal heights and weights. Hence, we resize them to 256x256 and take random crops of 224x224 to match the expected input of the GoogLeNet. We also zero-center the data by subtracting the mean image from ILSRVC 2012. Since the possible values in the image tensors only span 0-255, we do not normalize them. Furthermore, we make horizontal flips of the images since signs can be performed with either the left of the right hand, and our datasets have examples of both cases.

Since the difference between any two classes in our datasets is subtle compared to ILSRVC classes, we attempted padding the images with black pixels such that they preserved their aspect ratio upon resizing. This padding also allows us to remove fewer relevant pixels upon taking random crops.

5. Experiments, Results and Analysis

5.1. Evaluation Metric

We evaluate two metrics in order to compare our results with those of other papers. The most popular criterion in the literature is accuracy in the validation set, i.e. the percentage of correctly classified examples. One other popular metric is top-5 accuracy, which is the percentage of classifications where the correct label appears in the 5 classes with the highest scores.

Additionally, we use a confusion matrix, which is a specific table layout that allows visualization of the performance of the classification model by class. This allows us to evaluate which letters are the most misclassified and draw insights for future improvement.

5.2. Experiments

For each of the following experiments below, we trained our model on letters *a*-*y* (excluding *j*). After some initial testing, we found that using an initial base learning rate of 1e-6 worked fairly well in fitting the training data - it provided a steady increase in accuracy and seemed to successfully converge. Once the improvements in the loss stagnated, we manually stopped the process and decreased the learning rate in order to try and increase our optimization of our loss function. We cut our learning rate by factors ranging from 2 to 100.

Furthermore, we used the training routine that performed best with real users on our web application ('2_init'). We also built models to only classify letters *a*-*k* (excluding *j*) or *a*-*e* to evaluate if we attained better results with fewer classes.

1 layer reinitialization and learning rate multiple increase ('1_init'):
We initialized this model with the pre-trained weights from the GoogLeNet trained on ILSVRC 2012. We then reinitialized all the classification layers with Xavier initialization and increased the learning rate multiple of only this layer in order to help it learn faster than the rest of the net's pre-trained layers.

2 layer reinitializaiton and learning rate multiple increase ('2_init')
We initialized this model with the pre-trained weights from the GoogLeNet trained on ILSVRC 2012. We then reinitialized all the classification layers with Xavier initialization and appropriately adjusted their dimensions to match our number of classes. We increased the learning rate multiples for the top three layers beneath each classification head.

1 layer reinitialization, learning rate multiple increase, and increased batch size ('1_init')
We initialized this model with the pre-trained weights from the GoogLeNet trained on ILSVRC 2012. We then reinitialized all the classification layers with Xavier initialization and increased the learning rate multiple of only this layer in order to help it learn faster than the rest of the net's pre-trained layers. Finally, we drastically increased the batch size from 4 to 20.

1 layer reinitialization, uniform learning rate ('full_lr')
We initialized this model with the pre-trained weights from the GoogLeNet trained on ILSVRC 2012. We then reset all the classification layers with Xavier initialization but kept learning rate constant through the net.

5.3. Results

The results for all models are summarized below.

Model	Top-1 Val Accuracy	Top-5 Val Accuracy
a - y [1_init]	0.6847	0.9163
a - y [2_init]	0.6585	0.9043
a - y [batch]	0.6965	0.9076
a - y [full_lr]	0.7200	0.9098
a - k [2_init]	0.7430	0.897
a - e [2_init]	0.9782	1.000

Table 1. Optimal accuracy ranges for all models trained on each letter subset.

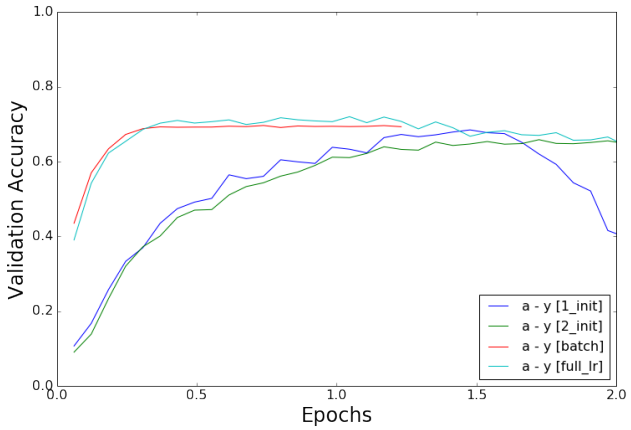


Fig. 2: Epochs vs. validation accuracy for all models trained on letters *a-y* (excluding *j*)

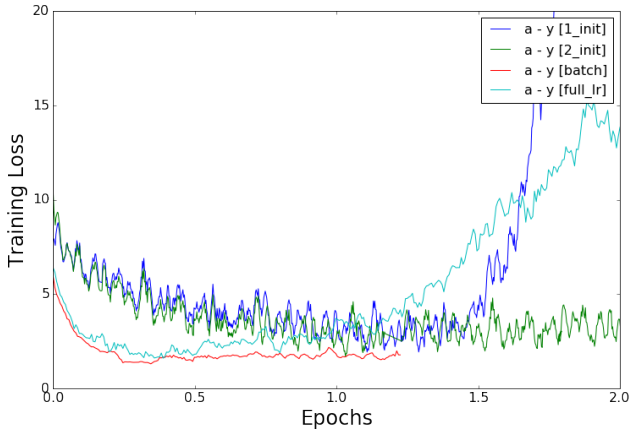


Fig. 3: Epochs vs. training loss for all models trained on letters *a-y* (excluding *j*)

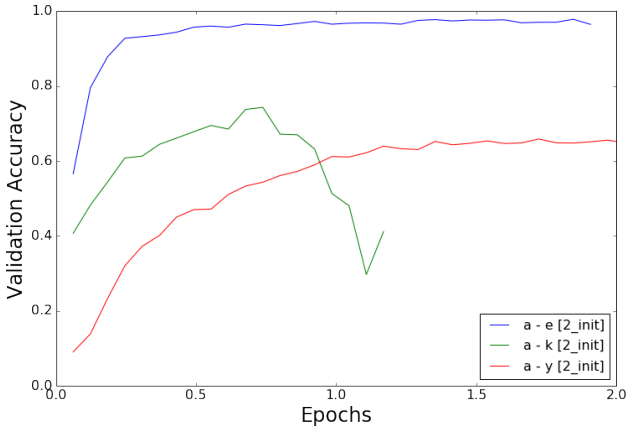


Fig. 4: Epochs vs. validation accuracy for the 2_init models trained on each letter subset (excluding *j*)

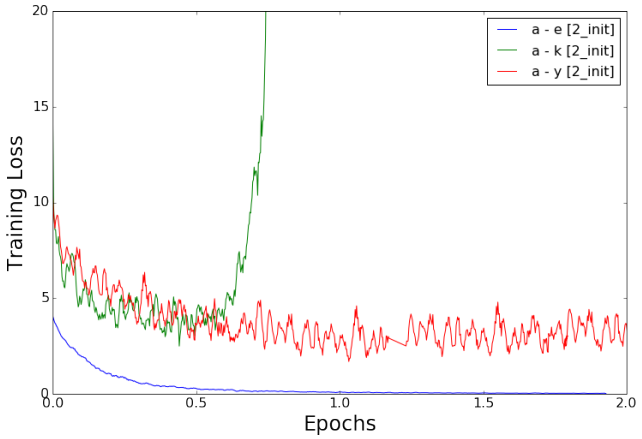


Fig. 5: Epochs vs. training loss for the 2_init models trained on each letter subset (excluding *j*)

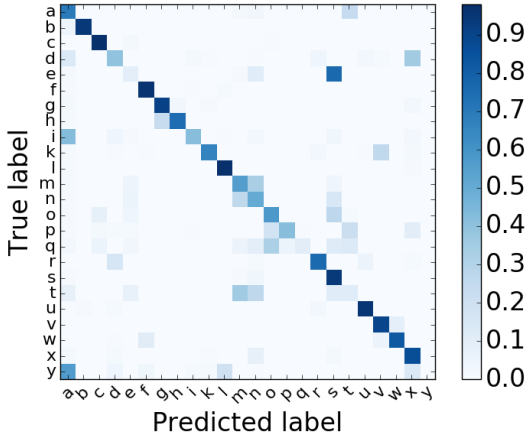


Fig. 6: Confusion matrix for the 2_init model trained on letters *a-y* (excluding *j*)

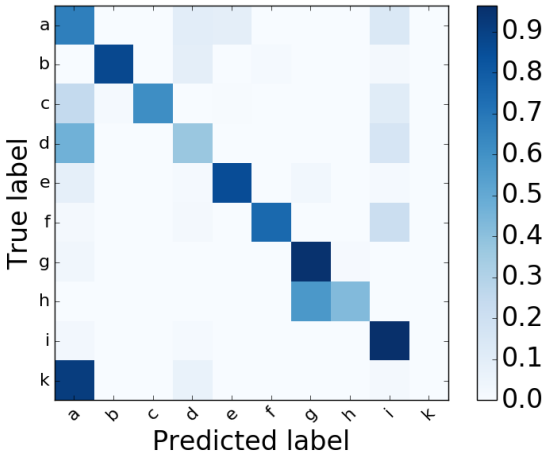


Fig. 7: Confusion matrix for the 2_init model trained on letters *a*-*k* (excluding *j*)

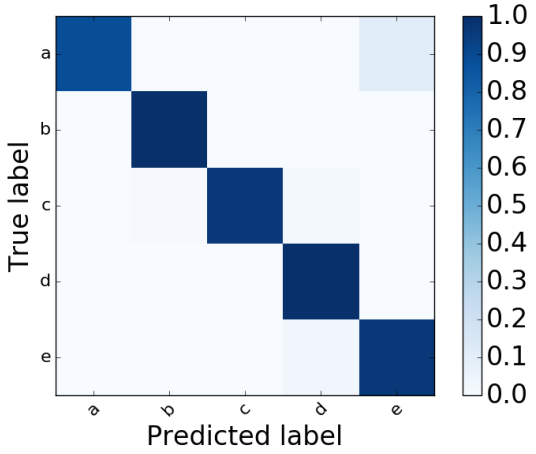


Fig. 8: Confusion matrix for the 2_init model trained on letters *a*-*e*

5.4. Discussion

5.4.1 Loss and Accuracy

Our losses (Fig. 3) were very noisy in the '1_init' and '2_init' models. Our space and time constraints initially required us to choose a less-than-optimal batch size value of 4, resulting in the noisy loss. However, after seeing these results, we trained a net using a Lighting Memory-Mapped Database (LMDB) and were able to increase the batch size to 20. This allowed us to reduce our loss more smoothly and monotonically, in addition to more quickly converging on a validation accuracy. Similarly, the 'full_train' model had uniform learning rates throughout each layer in the net, allowing us to learn more quickly. This is likely due to the fact that we are able to more easily alter the pre-trained weights in the GoogLeNet and correct for the significant differences between the datasets. While this leads to a tighter fitting of the training data, it did not seem to result in lower validation accuracy than those attained with other models. After analyzing many of the images from our dataset, we concluded that they were most likely produced by taking video frames of individuals making ASL signs in the same room and in a single sitting. The lack of variation in our data set explains the similar validation accuracy in our 'full_train' model relative to others.

Interestingly, changing our re-initialization scheme and learning rates had little effect on the final top-1 and top-5 accuracies: the largest difference between two models was less than 7% on top-1 and just over 1% on top-5 accuracy. We did notice, however, that the models utilized that only re-initialized weights at the classification layer strictly outperformed the 2-layer re-initialization model. Notwithstanding the fact that the distinction between classes in our dataset and the ILSVRC 2012 dataset are starkly different, this is not unexpected because of the high quality of the features extracted by GoogLeNet. It was pre-trained on significantly more images than we had available in our dataset.

There was little difference between the '1_init' and '2_init' models. Given the fact that GoogLeNet is 22 layers deep, intuition would lead us to believe that reinitializing 2 layers (versus reinitializing 1 and increasing the learning rate multiple on another) would not prove to be incredibly advantageous in fine-tuning our model to our validation set. This was confirmed in our experiments.

We qualitatively tested the four models on real users with our web application (see below). We decided to take the '2_init' model and create classifiers for letters *a* – *e* and *a* – *k* (excluding *j*) since we expected it would be less difficult to distinguish between fewer classes.

Not surprisingly, there was a direct, negative correlation between the validation accuracies attained using the '2_init' model, and the number of letters we sought to classify (Fig. 4). We attained a validation accuracy of nearly 98% with five letters and 74% with ten.

5.4.2 Confusion matrices (validation)

The confusion matrices reveal that our accuracy suffers primarily due to the misclassification of specific letters (e.g. *k* and *d* in Fig. 7). Often the classifier gets confused between two or three similar letters or heavily prefers one of the two in such a pair (e.g. *g*/*h* in Fig. 7 and *m*/*n*/*s*/*t* in Fig. 6).

The confusion matrix for the ten-letter model reveals that with the exception of classifying *k*, it performed reasonably well. We believe that there are two main

reasons for this result. First, in the dataset, *k* is signed from various different perspectives - from the front to the back of the hand facing the camera - and rotations - with the fingers pointing up and to the sides. The only consistent aspect is the center mass of the hand in all pictures, which is precisely what *a* resembles. Second, in the range *a-k*, *k* it has features that are very similar to letters *d*, *g*, *h*, and *i* – it can be constructed by a combination of segments of these letters. Hence, if the classifiers over-relied on any of these features individually, it would be easy to misclassify *k* as any of them and would make it harder to learn the latter.

5.4.3 Real-time user testing

As aforementioned, we initially tested our four *a – y* classification models on real users through our web application. This consisted of testing on images in a wide range of environments and hands. A key observation was that there is no significant correlation between the final validation accuracies of the models and their real-time performance on our web application. For example, 'full_lr' classified an input as *a* in over half the cases with > 0.99 probability, almost independently of the letter we showed it.

It was very apparent that the '2_init' model outshined the rest even though it produced the lowest validation accuracy. However, it still failed to consistently predict the correct letter in the top-5. For this reason, we created the models alluded to above with five (*a – e*) and ten (*a – k*, excluding *j*) classes.

Testing the classifiers with fewer classes on the live web application also yielded significantly different results from testing on the validation set. On ten-letter classification, some letters that were classified correctly in > 70% of cases (e.g. *b*, *c,* Fig. 7) were almost always absent in the top-5 predictions. Moreover, some letters were noticeably overrepresented in the predictions, like *a* and *e* (Fig. 7). We attribute this to the fact that neither of these has any fingers protruding from the center of the hand. They will thus share the central mass of the hand in the pictures with every other letter without having a contour that would exclude them.

6. Conclusions and Future Work

6.1. Conclusions

We implemented and trained an American Sign Language translator on a web application based on a CNN classifier. We are able to produce a robust model for letters *a-e*, and a modest one for letters *a-k* (excluding *j*). Because of the lack of variation in our datasets, the validation accuracies we observed during training were not directly reproducible upon testing on the web application. We hypothesize that with additional data taken in different environmental conditions, the models would be able to generalize with considerably higher efficacy and would produce a robust model for all letters.

6.2. Future Work

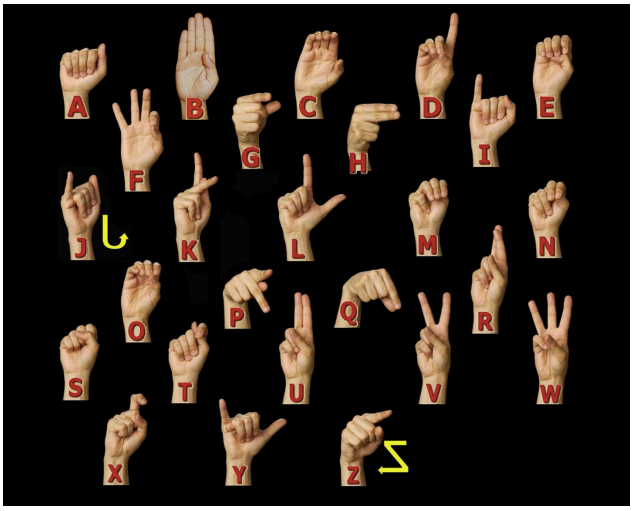
Additional Models: We focused our efforts on optimizing GoogLeNet, but it would be worth exploring different nets that have also been proven effective at image classification (e.g. a VGG or a ResNet architecture).

Image preprocessing: We believe that the classification task could be made much simpler if there is very heavy preprocessing done on the images. This would include contrast adjustment, background subtraction and potentially cropping. A more robust approach would be to use another CNN to localize and crop the hand.

Language Model Enhancement: Building a bigram and trigram language model would allow us to handle sentences instead of individual words. Along with this comes a need for better letter segmentation and a more seamless process for retrieving images from the user at a higher rate.

WEBAPP DEMO: https://youtu.be/79Fmu0TZ9xc

7. Appendix



American sign language fingerspelling alphabet [13]

8. References

[1] Mitchell, Ross; Young, Travas; Bachleda, Bellamie; Karchmer, Michael (2006). "How Many People Use ASL in the United States?: Why Estimates Need Updating" (PDF). Sign Language Studies (Gallaudet University Press.) 6 (3). ISSN 0302-1475. Retrieved November 27, 2012.

[2] Singha, J. and Das, K. "Hand Gesture Recognition Based on Karhunen-Loeve Transform", Mobile and Embedded

Technology International Conference (MECON), January 17-18, 2013, India. 365-371.
[3] D. Aryanie, Y. Heryadi. American Sign Language-Based Finger-spelling Recognition using k-Nearest Neighbors Classifier. 3rd International Conference on Information and Communication Technology (2015) 533-536.
[4] R. Sharma et al. Recognition of Single Handed Sign Language Gestures using Contour Tracing descriptor. Proceedings of the World Congress on Engineering 2013 Vol. II, WCE 2013, July 3 - 5, 2013, London, U.K.
[5] T.Starner and A. Pentland. Real-Time American Sign Language Recognition from Video Using Hidden Markov Models. Computational Imaging and Vision, 9(1); 227-243, 1997.
[6] M. Jeballi et al. Extension of Hidden Markov Model for Recognizing Large Vocabulary of Sign Language. International Journal of Artificial Intelligence & Applications 4(2); 35-42, 2013
[7] H. Suk et al. Hand gesture recognition based on dynamic Bayesian network framework. Patter Recognition 43 (9); 3059-3072, 2010.
[8] P. Mekala et al. Real-time Sign Language Recognition based on Neural Network Architecture. System Theory (SSST), 2011 IEEE 43rd Southeastern Symposium 14-16 March 2011.
[9] Y.F. Admasu, and K. Raimond, Ethiopian Sign Language Recognition Using Artificial Neural Network. 10th International Conference on Intelligent Systems Design and Applications, 2010. 995-1000.
[10] J. Atwood, M. Eicholtz, and J. Farrell. American Sign Language Recognition System. Artificial Intelligence and Machine Learning for Engineering Design. Dept. of Mechanical Engineering, Carnegie Mellon University, 2012.
[11] L. Pigou et al. Sign Language Recognition Using Convolutional Neural Networks. European Conference on Computer Vision 6-12 September 2014
[12] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/, 2014.
[13] Lifeprint.com. American Sign Language (ASL) Manual Alphabet (fingerspelling) 2007.