American Sign Language (ASL) Alphabet Image Classification Using Convolutional Neural Networks

Alexander Costa **ALEXANDER.COSTA@UGA.EDU**
Department of Computer Science, University of Georgia

ABSTRACT

Sign language is an important form of communication for people with impaired hearing and/or speaking abilities. American Sign Language (ASL) is used by approximately 500,000 people in the United States and is also used in Canada, Mexico, and many other countries. Among thousands of other signs, ASL consists of unique poses or gestures for each letter of the English alphabet. Image classification of these 26 signs constitutes a challenging task due to the complexity of ASL alphabet signs, high interclass similarities, large intraclass variations, and frequent self-occlusion of the hand. This work presents a method for ASL alphabet recognition using Convolutional Neural Networks (CNN). Specifically, we investigate 18 variations of a basic CNN architecture to discover which performs the task of ASL alphabet recognition with highest accuracy. To train our models, the ASL Alphabet dataset by Akash Nagaraj, consisting of 87,000 ASL alphabet sign images, was used. The ASL Alphabet Test dataset by Dan Rasband, consisting of 870 alphabet sign images, was used for model evaluation, as well. Of the 18 investigated models, the two top performing networks were the 5 Convolutional Layer, 3 Pooling Layer network trained for 10 epochs (5Conv) and the 5 Convolutional Layer, 3 Pooling Layer networks with L2 Regularization trained for 15 epochs (5Conv-L2-15Epoch), which achieved test accuracies of 0.4977 and 0.4897, respectively. Overall, the 5Conv networks dominated networks using other combinations of convolutional or pooling layers. For its defense against overfitting, we believe the 5Conv-L2-15Epoch to be of optimal ability.

Index Terms: *ASL, sign language recognition, image classification, neural networks, CNN*

I. INTRODUCTION

IA. BACKGROUND INFORMATION

American Sign Language is a critically important mode of communication for individuals with impaired hearing or speaking abilities as it allows for communication through visual cues alone. It is widely used throughout the United States as well as Canada, Mexico, and many other countries [1]. Despite its prevalence, communication between ASL users and non-sign-language speakers is still a considerably difficult problem. While professional interpreters exist, they are often not readily available and costly. An automatic ASL recognition system could constitute a remarkable and especially valuable advancement in the image recognition field and the sign language world at large. Not only could such a system ameliorate the difficulties of communication between ASL and non-ASL speakers but could facilitate numerous advancements in the intersection between ASL and human-computer interaction.

For decades, researchers have tried to solve the challenging problem of sign language recognition. Many proposed solutions rely on external devices, such as depth cameras, sensors, gloves, or motion capturing systems [1, 2]. Such constraints limit the applicability of these solutions to environments where these tools are available. Recently, given the demonstrated success of deep learning methods on computer vision tasks, focus has shifted to purely vision-based sign recognition powered by deep learning techniques. Being non-intrusive and requiring only a basic phone camera or webcam to generate input, these methods have the potential to be significantly more applicable and wide-reaching than solution requiring external devices. Deep learning-based solutions, however, suffer from a lack of large-scale, public sign language databases suitable for machine learning, as well as a weakened ability to differentiate between interclass similarities and increased susceptibility to self-occlusions of the fingers or hands.

In this work, we address the need for a robust sign language recognition system by focusing on the simpler, though still challenging, task of ASL alphabet image recognition using convolutional neural networks.

IB. CONVOLUTIONAL NEURAL NETWORKS

Overview of Neural Networks

Neural networks (NNs) have long been applied to image classification problems and have been shown to perform quite well at the task compared to other classification algorithms such as logistic regression, support vector machines, random forests, and k-nearest neighbors. Inspired by biologic neural networks, NNs consist of multiple layers of fully connected neurons each of which can be though of as a single processing unit. Each neuron has trainable weights and biases, the values of which can be learned to provide desirable predictions or classifications. The structure of a NN is broken down into the input layer, a set of hidden layers (with trainable weights and biases), and the output layer which constitutes the prediction or classification of input by the model. Figure 1 demonstrates the general structure of a NN.

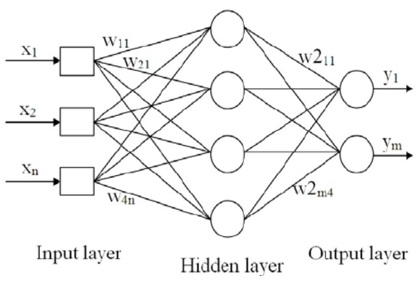

Figure 1. The structure of a basic neural network. The input layer, hidden layers, and output layer can take any desired shape.

Overview of Convolutional Neural Networks

Convolutional neural networks work in a similar fashion to the neural networks described above but incorporate several additions to the layer architecture. At the basic level, the layer structure of a CNN includes Convolutional Layers, Pooling Layers, and Fully Connected Layers. Fully Connected layers work in the same fashion as layers in a NN, with each node of a layer connected to all nodes of the previous layer. Convolution and pooling constitute the major building blocks of a CNN. They are responsible for extracting features from the input image which can then be used by Fully Connected layers to perform logical classification of the input image. Figure 2 presents the general structure of a CNN.

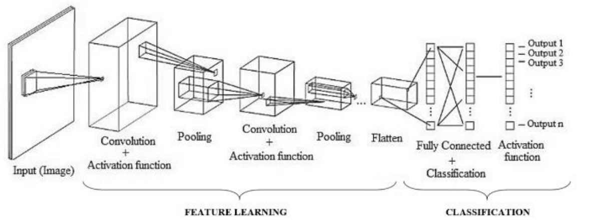

Figure 2. The structure of a basic convolutional neural network. The convolutional and pooling layers perform feature extraction on the input image followed by classification by the fully connected layers.

Convolutional Layers

The convolutional layers are the foundation of a CNN. A convolution is a linear operation involving the multiplication of a set of weights with a set of input values, much like a traditional NN. In the convolution process, each neuron takes input from a rectangular n×n region of the previous layer called the *local receptive field* and computes a scalar product of the local receptive field with an array of weights called a *filter*. This same filter is applied over all n×n regions of the input layer resulting in a 2D array of output values which represent a filtering of the input. This output is often called a *feature map*.

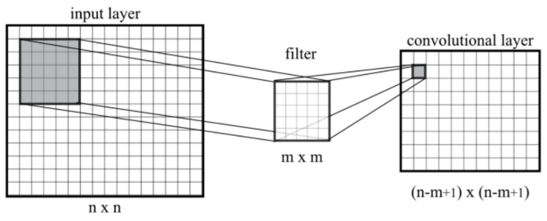

Figure 3. The convolution process [3].

The filter weights are learned by the network to extract specific features from the input layer. These extracted features can range from simple lines to basic shapes to complex structures like a human hand. The systematic application of the same filter across all sections of an input allows the filter the opportunity to discover that feature anywhere in the image, a capability known as *translation invariance*.

Convolutional layers are also not limited to learning only one filter. Often, a convolutional layer learns anywhere from 32 to 512 filters in parallel for a given input. This gives that particular layer 32 to 512 different ways of extracting features from an input which allows for specialization (e.g., the layer may detect not only basic lines, but lines specific to the training data used).

Furthermore, convolutional layers can be applied not only to the input data, but also to the output of other layers. The stacking of convolutional layers allows for a hierarchical decomposition of the input. The filters which operate directly on the input image can extract low-level features such as lines, then filters which operate on the output of these low-level layers can extract features which combine many low-level features like curves and shapes, all the way up to complex structures like faces, animals, hands, etc.

Pooling Layers

A problem with output feature maps from convolutional layers is that they are sensitive to the location of features in the input. In other words, small variations in the position of a feature in the input image will result in different feature maps. To address this issue, CNNs often use Pooling Layers to down sample the feature maps, making them more robust to changes in the position of features in the input image.

The Pooling Layer takes small rectangular blocks (often 2×2) from the Convolutional Layer output and down samples it. There are two common pooling operations, average pooling and maximum pooling, which calculate the average value or maximum value for each block of the feature map, respectively. In this work we universally use maximum pooling for all our pooling layers.

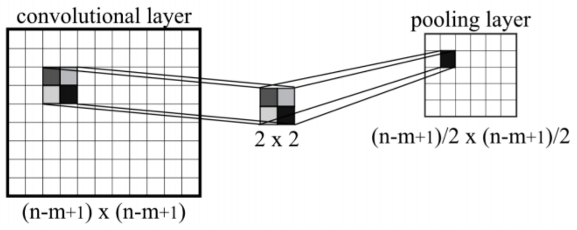

Figure 4. The pooling process [3].

The result of using a pooling layer is a summarized version of the features detected by the previous convolutional layer. They afford a capacity called *local translation invariance* to the network by helping to make feature map representations invariant to small translations in the input image.

Regularization Techniques

Because of their complexity and high number of parameters, convolutional neural networks are prone to overfitting their training data. Two common regularization techniques to prevent overfitting include Dropout and L2 regularization. We use both in many of the models investigated in this work. Dropout is a simple and effective procedure in which a certain proportion of randomly selected neurons do not participate in feedforward or backpropagation during training. This process simplifies the network and guards against overfitting. L2 regularization works by updating the cost function with a *regularization term*. This term encourages weights to decay towards 0 which yields simpler models resistant to overfitting their training data.

IC. RESEARCH OBJECTIVES

Aim 1 – Construct a CNN which can accurately classify ASL alphabet signs in real-world environments.

Our primary research objective is summarized in *Aim 1* above. To achieve this objective, we construct and train 18 CNN model variations on our training and validation datasets. Each model will perform a multiclass classification of input ASL alphabet sign images as one of 29 classes, including the 26

English alphabet letters and 3 auxiliary classes, SPACE, DELETE, and NOTHING (which may be helpful for real-time classification applications). After training and validation of the models, each model is subject to a comparison of top-1 accuracies on an unseen training dataset as an evaluation of model performance.

The datasets used to train and test our models are the ASL Alphabet dataset by Akash Nagaraj and the ASL Alphabet Test dataset by Dan Rasband, respectively. These datasets consist of labeled 200×200-pixel 3-channel RGB ASL alphabet sign images which comprise 29 classes (26 for the letters A-Z and the three aforementioned auxiliary classes). Our training data consists of 87,000 images with 3,000 images per class, and our testing data consists of 870 images with 30 images per class. Both datasets are perfectly balanced, eliminating the need for any dataset balancing. Before training, images are subject to a resizing to 64×64 pixels and a pixel rescaling factor of 1/255 to transform pixel values to the range [0, 1].

After the preprocessing of our image sets, we perform 18 identical experiments for each of the 18 CNN models investigated in this research. First the training data is split into 90% training data and 10% validation data, then each model is trained using training and validation data, and finally each model is evaluating against the testing dataset yielding the top-1 accuracy scoring metric for each model.

Our results indicate that a 5 Convolution Layer, 3 Pooling Layer (5Conv) model is best suited to the task of classifying ASL alphabet sign images. The top two performing models were the 5Conv model and the 5Conv model with L2 regularization (5Conv-L2-15Epoch) which achieved test accuracies of 0.4977 and 0.4897, respectively. Other notable mentions include the 5Conv-15Epoch model with a test accuracy of 0.4747, the 4Conv model with a test accuracy of 0.4690, and the 6Conv model with a test accuracy of 0.4517. All other models achieved a test accuracy under 0.4500.

The remainder of this paper is broken into a number of sections. Section II describes our training and test datasets and discloses our data preprocessing techniques in further detail. Section III describes our experiments in further detail and provides our experimental results in tabular form. Section IV presents an analysis of our experimental results, and finally section V provides a conclusion and discussion of the paper at large.

II. DATA

IIA. INTRODUCTION

The datasets used in this work were based on the American Manual Alphabet (AMA). Figure 5 shows all hand positions of the alphabet.

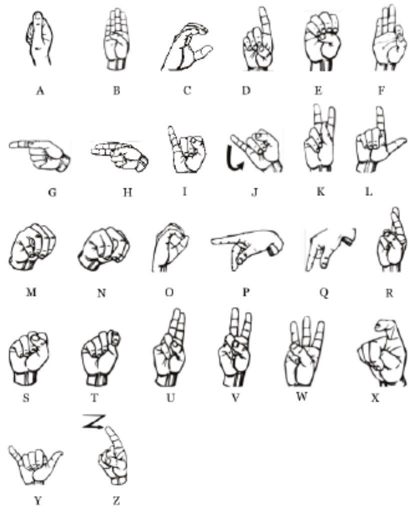

Figure 5. Hand poses used for constructing each letter in the American Manual Alphabet.

The datasets used for training and testing our networks were comprised of static hand poses of each letter from the alphabet, as well as hand poses for a SPACE, DELETE and NOTHING class which could prove to be helpful in real-time classification applications. Gesture-based signs, J and Z, were included in the training and testing datasets for completeness despite their temporal dimension. Static poses of these gesture-based signs taken at different times along the gesture sequence were used in both training and testing.

The training data, the ASL Alphabet dataset by Akash Nagaraj, consists of 87,000 200×200-pixel 3-channel RGB ASL alphabet sign images which comprise 29 classes (26 for the letters A-Z and the three aforementioned auxiliary classes). Each of the 29 classes contains 3,000 instances making a perfectly balanced dataset.

The training data uses varied lighting schemes and relative positioning for poses of each class. This variation is crucial for the application of the trained models to real world data which may exhibit any number of lighting, positioning, or background settings. Such variation enables the network to discern truly distinguishing features between classes instead of overfitting the particular interclass variation of the training data. Though the training data does provide variation in the form of lighting schemes and relative positioning, it still suffers greatly from excessive homogeneity. All images were taken by the same signer against a relatively static background. In ASL signing, there often exists subtle variation in positioning between different signers, not to mention regional and cultural differences which manifest as positioning variation between signers. With signer dependency being one of the most blocking challenges of current non-intrusive sign recognition approaches, the lack of multiple signer representation in the training data constitutes an enormous drawback to the generalizability of our trained models. Furthermore, the lack of background variation in our training data constitutes yet another drawback to generalizability. Backgrounds deviating strongly from those seen in Figure 6 are likely to yield unexpected behavior from our trained models.

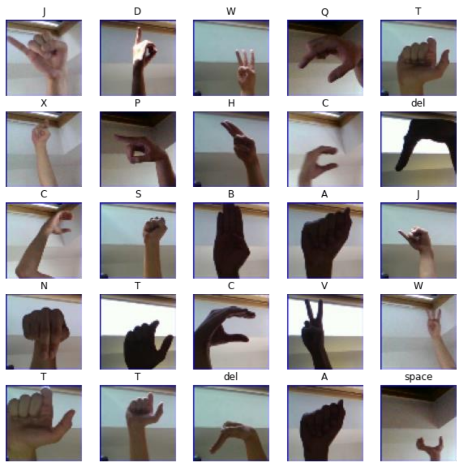

Figure 6. Example training images from the ASL Alphabet dataset by Akash Nagaraj with the true class label located above each image. Of note is the varied lighting and positioning of each pose, but lack of signer and background variation.

The testing data, the ASL Alphabet Test dataset by Dan Rasband, consists of 870 200×200-pixel 3-channel RGB ASL alphabet sign images which comprise the same 29 classes as the training data. Each of the 29 classes contains 30 instances making a perfectly balanced dataset. The testing data also makes the effort to use varied lighting schemes, relative positioning, and backgrounds but still suffers from the same homogeneity issues which affect the training data. Figure 7 displays example testing images from the ASL Alphabet Test dataset.

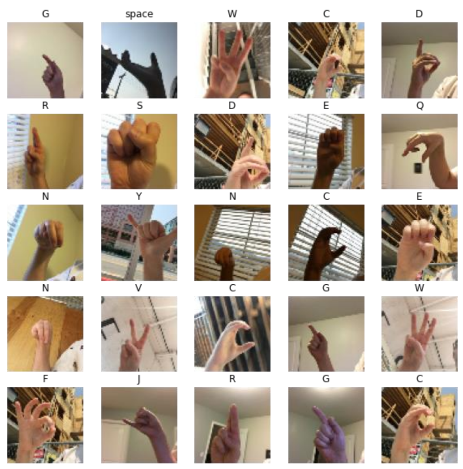

Figure 7. Example testing images from the ASL Alphabet Test dataset by Dan Rasband with the true class label located above each image. Of note is the varied lighting and positioning of each pose, but lack of signer and background variation.

Both testing and training data were shared on Kaggle.com for the express purpose of ASL alphabet image classification.

IIB. PREPROCESSING

The training and testing datasets were quite clean and required little preprocessing. Simple resizing and rescaling preprocessing steps were applied to the training and testing images. The 200×200-pixel images were resized to 64×64 pixels to decrease the number of parameters in each network which in turn sped up training time. Pixel values fall in the range of 0 to 255 which is not ideal for input values to a neural network, so all pixel values are rescaled by a factor of 1/255 giving a new pixel value range of 0 to 1.

IIC. PARTITIONING

The training data was randomly partitioned into 90% training and 10% validation sets to support parameter tuning. Evaluation of models was performed by testing models with the aforementioned ASL Alphabet Test dataset consisting of signs performed by a different signer in foreign environments. The motivation behind evaluating models with such disparate data is to emphasize the goal of generalizability. It is a rather trivial and ineffectual problem to classify signs performed by a single signer in a static environment. To classify signs performed by many signers in varied environments, however, is a much more valuable endeavor, so the dissimilar testing data is used to encourage our models to search for generalizable solutions.

III. EXPERIMENTS

For our experiments, we used the Python programming language, TensorFlow machine learning toolkit and Keras neural network library for data preprocessing, model training and evaluation. Experiments were performed on a cloud computing engine with Tesla P100 16GB VRAM as GPU, 13GB RAM and 2-core Intel Xeon as CPU.

Following data preprocessing we constructed 18 different CNN models using a combination of different convolutional layers, pooling layers, image preprocessing, regularization techniques, and number of training epochs. Model construction proceeded in a systematic fashion in which 3-5 models were constructed and tested. Then, the most promising model was further modified into another 3-5 models. The process was repeated until test accuracies did not improve. The 18 CNN models are summarized in Table 1.

A number of parameters were kept constant throughout the experimental process. These parameters include Convolutional Layer filter dimensions of 4×4 and stride dimensions of 1×1. Pooling Layers always used the maximum pooling operation with a 2×2 pooling window. Convolutional Layers and Fully Connected Layers always utilize the rectified linear activation function (ReLU), all L2 regularizations use a lambda of 0.0001, and all models use the Adam optimization algorithm and a sparse categorical cross-entropy loss function. Unless otherwise noted, all models were trained in 10 epochs.

Table 1: 18 CNN Model Variations

Description	**Params**
Base A base model consisting of only Fully Connected Layers. *Input → Flatten → Fully Connected(32) → Output*	394,205
1Conv A model consisting of a single Convolutional Layer and a single Pooling Layer. *Input → Convolutional(16) → Pooling → Flatten → Fully Connected(32) → Output*	462,573
2Conv A model consisting of two Convolutional Layer and two Pooling Layers. *Input → Convolutional(16) → Pooling → Convolutional(32) → Pooling → Flatten → Fully Connected(64) → Output*	357,069
3Conv A model consisting of three Convolutional Layer and three Pooling Layers. *Input → Convolutional(16) → Pooling → Convolutional(32) → Pooling → Convolutional(64) → Pooling → Flatten → Fully Connected(128) → Output*	250,509
3Conv-NoPool A model consisting of three Convolutional Layer and no Pooling Layers. *Input → Convolutional(16) → Convolutional(32) → Convolutional(64) → Flatten → Fully Connected(128) → Output*	24,826,509
3Conv-Dropout	250,509

A model consisting of three Convolutional Layer, three Pooling Layers, and a Dropout Layer. *Input → Convolutional(16) → Pooling → Convolutional(32) → Pooling → Convolutional(64) → Pooling → Dropout(0.2) → Flatten → Fully Connected(128) → Output*	
4Conv A model consisting of four Convolutional Layer and four Pooling Layers. *Input → Convolutional(16) → Pooling → Convolutional(32) → Pooling → Convolutional(64) → Pooling → Convolutional(128) → Pooling → Flatten → Fully Connected(256) → Output*	213,517
5Conv A model consisting of five Convolutional Layer and three Pooling Layers. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621
6Conv A model consisting of six Convolutional Layer and three Pooling Layers. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Convolutional(512) → Pooling → Flatten → Fully Connected(1024) → Output*	3,350,285
5Conv-Grayscale A model consisting of five Convolutional Layer and three Pooling Layers with grayscale input images. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,109
5Conv-MoreFilters-Grayscale A model consisting of five Convolutional Layer and three Pooling Layers with grayscale input images and doubled output filters for each Convolutional layer. *Input → Convolutional(32) → Pooling → Convolutional(64) → Convolutional(128) → Pooling → Convolutional(256) → Convolutional(512) → Pooling → Flatten → Fully Connected(1024) → Output*	7,536,125
5Conv-Dropout A model consisting of five Convolutional Layer, three Pooling Layers, and three Dropout Layers. *Input → Convolutional(16) → Pooling → Dropout(0.5) → Convolutional(32) → Convolutional(64) → Pooling → Dropout(0.5) → Convolutional(128) → Convolutional(256) → Pooling → Dropout → Flatten → Fully Connected(512) → Output*	1,892,621
5Conv-L2 A model consisting of five Convolutional Layer and three Pooling Layers with L2 regularization on each convolutional layer. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621
5Conv-Dropout-L2 A model consisting of five Convolutional Layer, three Pooling Layers, and three Dropout Layers with L2 regularization on each convolutional layer. *Input → Convolutional(16) → Pooling → Dropout(0.5) → Convolutional(32) → Convolutional(64) → Pooling → Dropout(0.5) → Convolutional(128) → Convolutional(256) → Pooling → Dropout → Flatten → Fully Connected(512) → Output*	1,892,621
5Conv-L2-5Epoch A model consisting of five Convolutional Layer and three Pooling Layers with L2 regularization on each convolutional layer and 5 epochs of training. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621
5Conv-L2-15Epoch A model consisting of five Convolutional Layer and three Pooling Layers with L2 regularization on each convolutional layer and 15 epochs of training. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621
5Conv-5Epoch A model consisting of five Convolutional Layer and three Pooling Layers and 5 epochs of training. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621

5Conv-15Epoch A model consisting of five Convolutional Layer and three Pooling Layers and 15 epochs of training. *Input → Convolutional(16) → Pooling → Convolutional(32) → Convolutional(64) → Pooling → Convolutional(128) → Convolutional(256) → Pooling → Flatten → Fully Connected(512) → Output*	1,892,621

Table 1. The 18 CNN models with accompanying parameter counts. A number within parentheses following a layer name indicate the output dimension in the case of Convolutional Layers and Fully Connected Layers, and the proportion of inactive neurons for Dropout Layers.

Following training, each model was tested against the ASL Alphabet Test dataset for an estimation of model performance. The results of this final test for each model are given in Table 2. No model took a prohibitive amount of time to complete training or testing.

Table 2: Experimental Results for the 18 CNN Models

Name	Training Accuracy	Validation Accuracy	Testing Accuracy	Training Time (s)	Testing Time (s)
Base	0.0323	0.0331	0.0345	60	0
1Conv	0.9864	0.9791	0.0701	75	0
2Conv	0.9964	0.9959	0.1828	87	0
3Conv	0.9952	0.9954	0.3621	97	0
3Conv-NoPool	0.9946	0.9905	0.1575	261	0
3Conv-Dropout	0.9919	0.9980	0.3989	98	0
4Conv	0.9931	0.9926	0.4690	104	0
5Conv	**0.9926**	**0.9977**	**0.4977**	**156**	**0**
6Conv	0.9914	0.9956	0.4517	204	0
5Conv-Grayscale	0.9931	0.9939	0.3299	170	0
5Conv-MoreFilters-Grayscale	0.9916	0.9943	0.2586	300	0
5Conv-Dropout	0.9595	0.9894	0.3793	161	0
5Conv-L2	0.9935	0.9960	0.4195	161	0
5Conv-Dropout-L2	0.0344	0.0343	0.0345	163	0
5Conv-L2-5Epoch	0.9832	0.9893	0.4494	81	0
5Conv-L2-15Epoch	**0.9931**	**0.9852**	**0.4897**	**241**	**0**
5Conv-5Epoch	0.9882	0.9890	0.4414	76	0
5Conv-15Epoch	0.9927	0.9983	0.4747	229	0

Table 2. Experimental results for all 18 CNN models. The 2 models achieving the highest test accuracies are bolded.

The results indicate that a 5Conv model is best suited to the task of classifying ASL alphabet sign images. The top two performing models were the 5Conv model and the 5Conv model with L2 regularization (5Conv-L2-15Epoch) which achieved test accuracies of 0.4977 and 0.4897, respectively.

Most models except the trivial 1Conv and those trained with only 5 epochs converged quickly to 0.9900+ training and validation accuracies but many performed extremely poorly on the testing dataset. As the number of convolutional layers used increased, as did the performance on the testing dataset until the 6Conv model which invariably performed worse than the 5Conv model. Due to the diminishing number of available inputs as more and more layers are stacked and the limited input dimensions of 64×64, we could not investigate models with any more convolutional layers than 6Conv.

The elimination of Pooling Layers (3Conv-NoPool) invariably diminished testing accuracies and the use of Dropout Layers (3Conv-Dropout, 5Conv-Dropout) slightly increased performance in the 3Conv-Dropout case but largely decreased performance in the 5Conv-Dropout case.

Proceeding with the 5Conv model, the use of grayscale input images were used to test whether color was contributing valuable sign-discerning information to the classifier or was being overfit by the classifier and causing the relatively low testing accuracies. The use of grayscale inputs invariably diminished test accuracies and had no major effect on training and validation accuracies, meaning color did contribute valuable and generalizable sign-discerning information to the network.

Given the large disparity between validation and test accuracies, dropout and L2 regularization techniques were tested with the 5Conv model in an attempt to diminish any possible overfitting. Both dropout and L2 regularization techniques invariably decreased testing accuracy of the 5Conv model, though the 5Conv model with L2 regularization trained for an additional 5 epochs (for a total of 15) had comparable performance to the 5Conv model. The lack of improvement in testing accuracy when using regularization techniques indicates that the underlying causes for such low testing accuracies must be something other than overfitting during training.

IV. ANALYSIS

Despite extremely high training and validation accuracies greater than 0.9900 for most models, we were unable to achieve test accuracies greater than 0.5000 for any model investigated. Here we present a number of hypotheses and potential remediations for this disappointingly low testing accuracy.

First and foremost, our training/validation and testing datasets are markedly dissimilar from one another yet excessively homogenous within. As discussed in Section II, our training set consists of images from only one signer in a relatively static environment. This homogeneity does not seem to offer our models enough intraclass variability information in order for them to generalize well to other signers or environments, which is demonstrated by our low testing accuracies. Ideally, a training set consisting of signs by tens, if not hundreds, of different signers in hundreds of different lighting and background environments would be used to account for all kinds of real-world variability which our training set lacks. We hypothesize that a larger, more varied training dataset could offer many improvements to our models' accuracies.

On top of an excessively homogenous training dataset, the classification of ASL alphabet signs itself presents many obstacles due to high interclass similarities. Many letter signs are extremely similar, differing only by one finger's position (G/H, K/V, U/V, E/S, M/N/S/T) or even simply the rotation of the hand (P/K, I/J, D/Z). Even a beginner ASL user might mistake one of these letters for the other. To see whether the CNN models were themselves susceptible to these interclass similarities, we investigate the confusion matrices of the top 2 models presented in Figures 8 and 9 below.

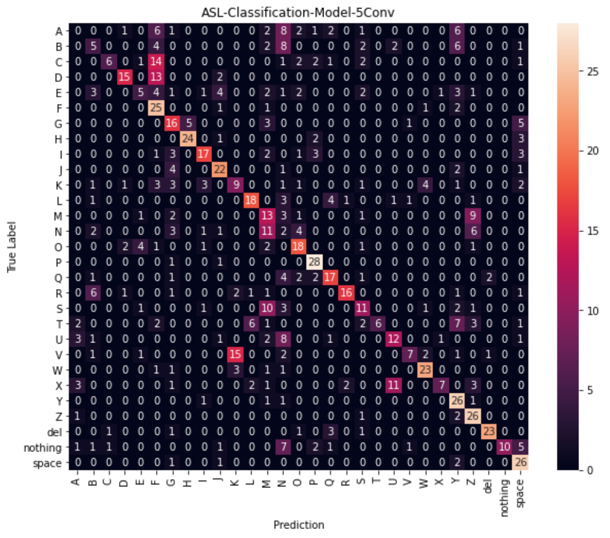

Figure 8. Confusion matrix for the 5Conv model.

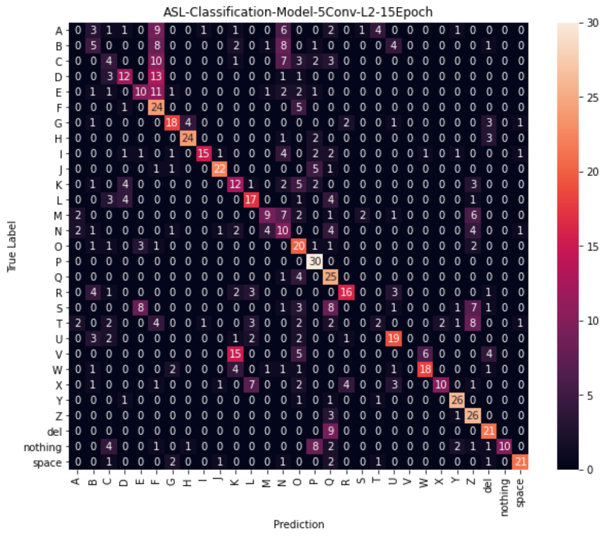

Figure 9. Confusion matrix for the 5Conv-L2-15Epoch model.

Counts falling along the main diagonal are correctly classified test images. Colored squares outside the main diagonal constitute significant and habitual mistakes in the model classification. Many significant misclassifications do in fact correspond with those class pairs having high interclass similarities. For example, 15 Vs were classified as Ks in both models. 5Conv classifies many Ss as Ms. 5Conv-L2-15Epoch classifies many DELETEs as Qs, which share many similarities. Both models misclassified Gs as Hs and both models misclassify many Cs and Ds as Fs which all share a circle-like structure from certain angles.

While a number of unexpected misclassifications exist as well, it is clear that the test accuracies of the best models suffer due to the high interclass similarities between many classes. To accurately discern between such similar classes our models demand higher density input images which may allow the model to differentiate similar classes more easily through the increase in available information. Higher density input images may also allow for the use of more convolutional layers which could strengthen our models' classification abilities. Some form of image segmentation procedure prior to model training may also help the models differentiate similar classes by focusing the areas of interest (hand and fingers) and ignoring superfluous data like background patterns.

V. CONCLUSION

Our work indicates that among many different simple CNN architecture variations, the 5Conv model, consisting of 5 Convolutional Layers and 3 Pooling Layers, performs best at ASL alphabet sign classification. The 5Conv model with additional L2 regularization and trained for 15 epochs exhibits comparable performance, as well. Although high training and validation accuracies were easy to attain, our models failed to reach even 0.5000 testing accuracy, with 5Conv achieving the highest testing accuracy of 0.4977. It seems our models are bottlenecked not by overfitting issues during training as much as by the lack of signer and background variation in our training dataset and by high interclass similarities in the ASL alphabet.

Our findings present a number of opportunities for the experiments herein to be continued and extended. Firstly, the aggregation and preprocessing of many more ASL alphabet datasets in addition to those used in this work could be undertaken to provide the models with much more varied, real-world information and subsequently strengthen their generalizability.

While a number of CNN architecture variations were investigated, very little hyperparameter optimizations were performed. It may prove useful to investigate different input image dimensions, as well as different filter and stride dimensions, different L2 regularization lambdas, and different optimizers and loss functions.

As previously mentioned, the addition of an image segmentation process may greatly improve the models' classification abilities as well as increase the types and variety of images for which the models are able to work well (e.g., with image segmentation, classification of full body images may also work well since model attention could be focused only to hands and fingers within the image).

Finally, a natural extension of the work presented in this paper would be a real-time ASL alphabet sign recognition system based on CNNs which may prove to be useful in many ASL-related areas and could even be generalized to signs and gestures of any kind for use in gesture-based computer control systems.

The code used in this work is available on Kaggle. (https://www.kaggle.com/alexcostaluiz/asl-alphabet-sign-recognition)

REFERENCES

[1] H. R. V. Joze and O. Koller, "MS-ASL: A large-scale data set and benchmark for understanding american sign language," *arXiv preprint arXiv:1812.01053,* 2018.

[2] W. Tao, M. C. Leu, and Z. Yin, "American Sign Language alphabet recognition using Convolutional Neural Networks with multiview augmentation and inference fusion," *Engineering Applications of Artificial Intelligence,* vol. 76, pp. 202-213, 2018.

[3] C. Wang and Y. Xi, "Convolutional neural network for image classification," *Johns Hopkins University Baltimore, MD,* vol. 21218, 1997.