American Sign Language Alphabet Recognition Using Computer Vision and Machine Learning Techniques

William Ritchie
School of Computer Science
San Diego State University

Kristi Werry
School of Computer Science
San Diego State University

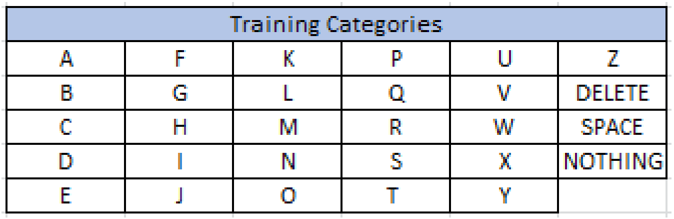
***Abstract*—American Sign Language (ASL) exists as a method of communication using physical gestures and hand signs instead of the verbal alternatives. We seek to bridge the gap between ASL users and non-ASL users by applying computer vision and machine learning techniques to images of the ASL alphabet. We propose a comparison of three methods for ASL alphabet recognition: error-correcting output code (ECOC) using multiple binary support vector machines (SVM) with histogram of oriented gradient (HOG) feature descriptors, bag of features (BOF) learning model with speeded-up robust features (SURF), and a convolutional neural network (CNN). Using an available 87000 image dataset, we trained these models with various parameters to see which one achieved highest accuracy in correctly classifying the ASL signs. We hypothesized the CNN would be the most successful in this task, due to its current prominence in the field of the computer vision. The results of our experiments show that the optimized version of ECOC using multiple binary SVMs with HOG feature descriptors achieved the highest accuracy of the three models with 99.4 percent on only 29000 training images.**

I. Introduction

Sign language is a language that is fundamentally different from any other language. Most languages are spoken or written; however, sign language is a complex language that uses the movement of hands, facial expressions, and body postures to convey a message. Each sign language also has its own grammar and lexicon. With the recent advancements in computer vision and machine learning, computer recognition of sign language is now feasible. With that said, there still exist a few problems in translating sign languages using current computer vision techniques, such as the number of complex signs and subtle motions that differentiate words and phrases. There is also a problem of non-uniformity amongst sign language users, in other words different sign language users might sign the same word slightly differently than another user. Another problem is the context provided by facial expressions. Such context can change the meaning of what is being said, thus making the need for facial recognition along with hand sign recognition valuable. There also exists a problem where the majority of people in the world are non-sign language users which can make communication between someone who uses sign language and a non-sign language user challenging. In this paper, we will focus on the hand signs of the alphabet for the American Sign Language (ASL). Convolutional neural networks (CNN) and support vector machine (SVM) are two of the more popular and successful machine learning techniques used to recognize an image of a signed letter. We chose SVM as opposed to other machine learning methods, due to its notoriety as a strong model for both binary and multi classification of images and because of its general simplicity to implement. CNN was chosen over other deep learning neural networks because it is specifically suitable for images as input whereas others are not. A big difference between CNN and other neural networks is that subregions might overlap, hence the neurons of a CNN produce spatially-correlated outcomes, whereas in other types of neural networks, the neurons do not share any connections and produce independent outcomes. In addition, in a neural network with fully-connected neurons, the number of parameters (weights) can increase quickly as the size of the input increases. A convolutional neural network reduces the number of parameters with the reduced number of connections, shared weights, and downsampling. We propose a comparison between these two techniques in tandem with various image processing and feature extraction algorithms such as histogram of oriented gradients (HOG) and speeded up robust features (SURF), to provide a highest accuracy model for ASL alphabet detection. The goal is to use the most accurate model to bridge the language gap between ASL users and non-users to make communication easier.

II. Task Description

We use image classification to assign category labels to images and test the correct classification. Figure 1 shows the 29 unique categories we used to classify the ASL alphabet [1]. The first 26 categories are the letters of the english alphabet, and the remaining three are: delete, space, and nothing. Delete and space are the sign language representation



Training Categories					
A	F	K	P	U	Z
B	G	L	Q	V	DELETE
C	H	M	R	W	SPACE
D	I	N	S	X	NOTHING
E	J	O	T	Y	

Fig. 1: Training categories used for classifying ASL alphabet.

of those respective terms, and nothing represents no sign present (that is, no hand is present).

A. *Error-Correcting Output Code with SVM Binary Learners*

The process of assigning categories involves collecting unique features over a collection of training images in a specific category, and applying a machine learning model to learn the collection of features in each category. The first method we propose involves an error-correcting output code (ECOC) multi-class model using SVM binary learners, to learn the features extracted by applying the HOG algorithm on the dataset. ECOC classification requires a coding design which determines the classes that the binary learners train on, and a decoding scheme, which determines how the results (predictions) of the binary classifiers are aggregated [2]. A coding design is a matrix where elements direct which classes are trained by each binary learner, that is, how the multiclass problem is reduced to a series of binary problems [2]. HOG is an algorithm that extracts a feature descriptor for object detection from images. It does this by counting each instance of gradient orientation at specific points of an image. The appearance and shape of an image can be described through the intensities and directions of the gradients. These features are useful because the gradient magnitudes are usually larger around corners and edges which can tell a lot about an object. Figure 2 shows the result of applying HOG on an image of a signed letter. As you can see the larger gradient magnitudes around the edges of the hand. The resulting features from

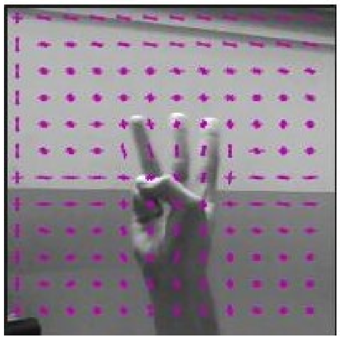


Fig. 2: HOG features of the ASL sign ‘W’.

applying HOG are used as an input to the ECOC model, which selects the most optimal features to provide classification of the image. Each SVM binary learner searches for an optimal hyperplane that separates the data into two classes. For separable classes the optimal hyperplane maximizes a margin surrounding itself (space that does not contain any observations), which creates boundaries for the positive and negative classes [3].

B. *Bag Of Features with Speeded Up Robust Features Descriptors*

After further investigation, we decided to compare HOG against SURF which is a fast approximation of scale-invariant feature transform (SIFT). The SIFT descriptor is based on HOG, so we expected the faster and more efficient SURF algorithm to yield higher results. In Figure 3 you can see the results of applying SURF on an image. We realized we could

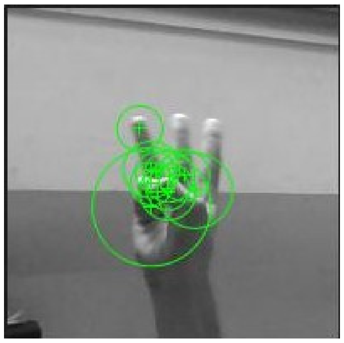


Fig. 3: SURF features of the ASL sign ‘W’.

not use SURF with ECOC due the varying number of features per image, leading us to the second proposed method for ASL alphabet recognition, called bag of features (BOF). BOF still utilized SVM and naturally handled the issues presented by SURF. To detect interest points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed using a precomputed integral image. Its feature descriptor is based on the sum of the Haar wavelet response around the point of interest [5]. BOF constructs a vocabulary of SURF feature descriptors representative of each image category. In constructing this vocabulary it reduces the number of features through quantization of feature space using K-means clustering. Then, based on results of the clustering, it counts the visual word occurrences in each image and produces a histogram that becomes a new and reduced representation of each image. This histogram forms a feature vector of an image, which is then fed into a multiclass linear SVM classifier during the training process [4].

C. *Convolutional Neural Network*

The third method we propose for ASL alphabet recognition is training a CNN. A CNN eliminates the need for manual feature extraction as the features are learned directly by the CNN. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters become complex features that uniquely define the object. CNN is composed of an input layer, many hidden layers, and then an output layer. These layers perform operations that alter the data with the intent of learning features specific to the data. The three most common middle layers are: convolution, ReLU (activation), and pooling. Convolution puts the input images through a set of convolutional filters, each of which activates certain features from the images. Rectified linear unit (ReLU) allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as activation, because only the activated features are carried forward into the next layer. Pooling simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn. Following these layers are the classification layers which is comprised of a fully connected layer that contains the probabilities for each class of any image being classified, and a softmax layer to provide the classification output. There are different types of CNN such as R-CNN, Fast R-CNN, and Faster R-CNN. While those models have been proven to be successful we decided to go with our own implementation of a CNN in order to have more control over the network and attempt to produce equally impressive results.

For each of the proposed methods we conducted experiments in an attempt to maximize the accuracy of each model. These experiments involved manipulating many of the parameters under which the different models could run and observing how those manipulations affected the accuracy. The set of parameters that provided the highest accuracy model was then compared against the other methods using the same exact set of training and testing images.

III. Major challenges and Solutions

Our greatest challenge was lack of adequate resources to train the machine learning models. Normally these models would run on clusters of CPUs or GPUs to drastically decrease the time it takes to train the model. Despite the fact that our training images were small in size and in some cases we converted the entire image set to grayscale, training time still could take many hours to finish (the more extreme cases being upwards of 12 hours). During those hours of training, the huge number of extracted features completely filled the computer's memory and the bus lines to disk would be completely taken up, preventing us from doing much else while the models trained. We did not have the money to purchase multiple CPUs or highly expensive GPUs. Our solution was to use an alternate computer in order to continue to work while the other computer trained the models, thus optimizing our time. We also set the settings for the CNN to multi-CPU (parallel processing) to help speed up the process. We still ran into the issue where some of the models were so exhaustive that they crashed the computers which meant we had to restart the training of the model. There was no straightforward solution to this; instead we accounted for failed model training sessions in the estimated time it would take for us to train all of the models. Our lack of hardware and resources definitely slowed down our testing, but we were able to produce results after an abundant amount of time.

IV. Experiments

A. Dataset description

The data set we used to train and test our machine learning models was downloaded from kaggle.com[1]. The training data set is split into 3000 images per category, with there being 29 different categories (26 for each ASL letter in the alphabet and 3 for delete, space, and nothing). The 29 categories are those previously mentioned in the first paragraph of Task Description, and are the same as the labels we used in our classifications for training the models. The maximum number of images we trained per category was 2800 due to the constraints previously mentioned in Major Challenges and Solutions. The images are close up pictures of the hand producing the sign, thus the signs took up most of the space in the image. Each image varies from a solid to a very noisy background. These images also range from dark pictures to brightly lit photos. These different types of images of the same object help us train our models and make them more robust. The dataset also includes images for testing, with one image to test per category. We decided to not use these 29 images and instead use 200 (per category) of the remaining images from the original 3000 (per category). We felt this would give a more accurate result from the testing process.

B. Evaluation Metrics

1) Parameters: The experiment consisted of manipulating the various parameters to our three proposed methods of ASL alphabet recognition in order to achieve the highest accuracy possible. Since the different methods are very different in design, their respective parameters do not correlate to the other models. This means each model has its own experiment of altering parameters, and the only comparison that occurs between the different models is with respect to the highest accuracy achieved for each model. Thus, it is necessary to also self-compare the models, and understand why certain parameter values for a particular model may have caused higher accuracy.

Due to our lack of resources (previously mentioned in the Major Challenges and Solutions section) we only increased the number of training images for the parameters that yielded better results. The increase amount is as follows: 300 images (per category) for the parameters with better results from the 100 training images (per category) run, and 2800 images (per category) for the singular run that performed the best in the previous 300 image (per category) run. Occasionally we trained with 1000 images (per category) if we felt 2800 was too computationally intensive.

For the ECOC model we used two of Matlabs built-in functions: fitcecoc and extractHOGFeatures. The extractHOGFeatures extracts HOG features from a particular grayscale image, and the result is stored into a matrix. This step repeats for each image in the training image data set. The resulting matrix is used as an input to fitcecoc, which takes those features and a vector with the actual labels for the respective images, to train the multiple binary SVMs.

The parameters that we felt were worth considering for extracting HOG features were: InterestPoints, CellSize, BlockSize, and NumberOfBins. We decided not to use interest points from a corner or edge detector as an input to the function. Our reasoning behind this is that our training image dataset consists of close up photos of the hand sign with little

[1]The ASL alphabet image dataset we used can be found at the website Kaggle.com and it provided us with a collection of 87,000 images. There are 29 classes, 26 for each letter (A-Z) and 3 for Space, Delete, and Nothing. Each class has 3,000 images and each image is 200 by 200 pixels.

background noise, thus we felt no need to detect interest points and that applying HOG over the entire image was sufficient. This also makes the model more closely related to the CNN model, since the CNN also extracts features over the entire image. In our conclusion and future works we discuss more on using interest points as an input to the HOG algorithm and to CNN (forming essentially an R-CNN model). Block size refers to the number of cells in a block. A large block size value reduces the ability to suppress local illumination changes. Reducing block size helps to capture the significance of local pixels and help suppress illumination changes of HOG features [7]. The default block size was set to 2 x 2, which we felt was sufficiently small and increasing the size did not seem desirable so we decided to not have block size be a parameter that we would change. The number of bins refers to the number of orientation histogram bins. Increasing this value increases the size of the feature vector, which requires more time to process, but is able to encode finer orientation details. The default number of bins is 9, and we decided not to change this value because it seems to be the standard in literature [7]. Cell size refers to the size of the HOG cell, which is used to capture spatial information. Increasing the cell size captures large scale spatial information, but loses small-scale detail. We decided this was an important parameter worth altering for the experiment. The value of cell size is a 2 element vector.

For the fitcecoc function, there are two parameters we felt were worth considering: Coding and Learners. Coding refers to how a multiclass problem is turned into a series of binary classifiers. Among the different options for the coding parameter two stood out as the most probable for success: onevsone and onevsall. One versus one is where for each binary learner, one class is positive, another is negative, and the rest are ignored. This design exhausts all combinations of class pair assignments. One versus all is where for each binary learner, one class is positive and the rest are negative. This design exhausts all combinations of positive class assignments [8]. The other parameter we considered was the Learner which refers to the type of binary classifier. As previously stated, we desired the binary classifier to be SVM, which happened to be what fitcecoc was set to use as a default. The default kernel for the templateSVM class Matlab provides, is a linear model. In our experimentation we tried both a gaussian kernel and the linear kernel [14].

We used the built-in Matlab functions bagOfFeatures and trainImageCategoryClassifier for the BOF model. The bagOfFeatures function takes in our dataset as an input, and begins by extracting SURF features.This bagging step was helpful because SURF would return varying number of features per image, thus making it difficult to compare and learn using fitcecoc. All of the relevant features were placed in a bagOfFeatures object which is used as an input to the trainImageCategoryClassifier method. This trains an SVM using the dataset with the category labels along with the bag of features.

A few important parameters for bagOfFeatures are: StrongestFeatures, PointSelection, GridStop, and BlockWidth. StrongestFeatures allows us to pick a percentage of the best features [9]. By default, this is 80 percent; however, this produced a large amount of features that took a huge amount of time to process. We changed this to 50 percent to speed up computation and after testing on a small scale we realized the accuracy was slightly less, and felt the increased computation time in this case was worth the increase in accuracy so we kept this parameter at 80 percent. PointSelection has two options for the method of picking point locations. We decided with Detector because it selects point based on SURF whereas Grid picked points based on a predefined grid. Since we did not use Grid in the point selection, there was no reason to change the GridStep parameter. For BlockWidth, we also left this as default [32 64 96 128]. The minimum block width is 32, thus we could not make this smaller. On the other hand, since our images are only 200 by 200 pixels, and based on our previous experimenting with larger block widths, we decided we did not want to make the block widths any larger in fear of losing important features.

The only parameter that would change the accuracy of the training model trainImageCategoryClassifier was the LearnerOptions [10]. This function trains a multi-class SVM. Like the fitcecoc function, we decided to use a Gaussian filter for training to help reduce noise.

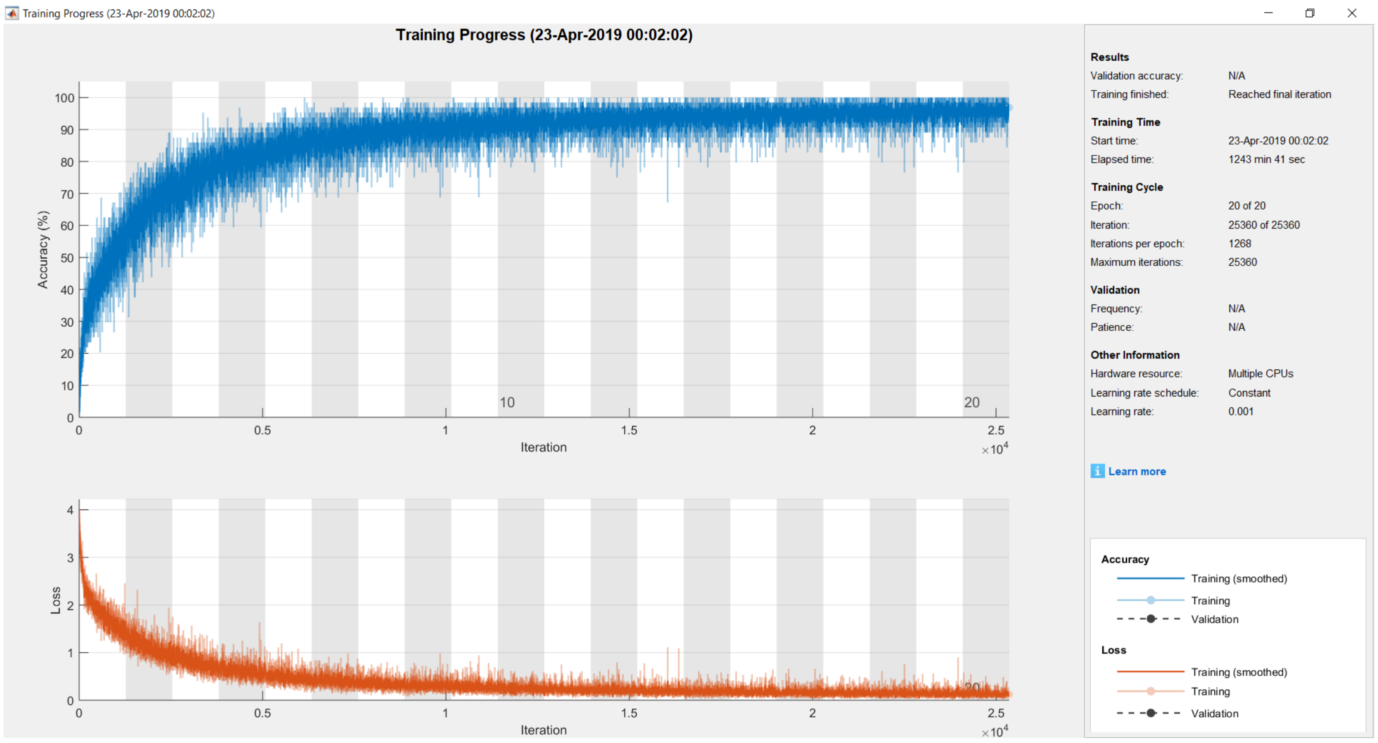
Creating a custom CNN was significantly different from training the other models. The CNN only has two requirements for its layers: to begin with an image input layer, and to end with an output (classification) layer. The layers in between can be anything. The large freedom in number of layers and types of layers makes creating a custom CNN very time consuming, and so for the scope of this project we tried to limit ourselves. After specifying the layers of the CNN, we also needed to specify the options under which the CNN would train. These options include: the MaxEpochs, MiniBatchSize, InitialLearnRate, along with many others. Both the layers, the options, and the image dataset with the image labels are then given to the function: trainNetwork, which will train the CNN.

The first layer in every CNN is the image input layer which defines the image size; in this case, 200 by 200 by 3. We decided to use the RGB colors and not change the original images to grayscale because this would give the neural network more information to work with while traversing through the other layers. In Matlab this functionality is achieved with the imageInputLayer function. This function has three parameters worth noting: Normalization, AverageImage, and DataAugmentation. The Normalization parameter defaults to zerocenter, which subtracts the average image specified by the average image property. We felt this normalization technique was sufficient and did not need to be changed. Likewise the AverageImage property defaulted to a h-by-w-by-c array, which was designed for the Normalization parameter which we decided to change. The DataAugmentation parameter would have been useful except we already augmented the training data prior to this layer, therefore we kept this value as none. The reasoning behind our data augmentation is discussed later in this section. The next layer is usually a convolutional layer which applies a number of filters of a specific size on the padded images. The goal of the convolutional layer is to activate the key interest points of the image. We achieved this by using Matlabs convolution2dLayer. This function has many possible input parameters, so for the scope of this project we limited ourselves to these few: FilterSize, NumFilters, PaddingMode, and PaddingSize. There are other important parameters to this function that we chose not to try such as: Weights and Bias. We felt the Weights and Bias was out of scope and left those parameters to their default values. The batch normalization layer usually follows the convolutional layer. This layer normalizes the activations and gradients from the convolution layer to optimize the network training. Following the batch normalization layer, the ReLU layer is the most common to use for nonlinear activation. Both the batch normalization layer and the ReLU layer seemed to be the standard and the corresponding Matlab functions did not take in many inputs so we decided to leave these layers at their default settings. At this point there is usually a very large feature map. We used a down-sampling layer like max pooling layer to remove the redundant spatial information. Down-sampling makes it possible to increase the number of filters in deeper convolutional layers without increasing the required amount of computation per layers. This functionality was achieved using Matlabs built-in function: maxPooling2dLayer. The primary parameters we modified for this function were: PoolSize and Stride. The pool size refers to the dimensions of the pooling regions. The stride is the step size for traversing the input vertically and horizontally. After the first pooling layer, we attempted repeating and trying different combinations of the convolutional layer, batch normalization layer, ReLU, and max pooling layer. Regardless of the different combinations, the last three layers were always the same. The first is the fully connected layer. The fully connected layer connects to all of the features from previous layers across the image to identify a pattern which is used to classify the image. We used Matlabs fullyConnectedLayer function to achieve the necessary functionality of this layer. This function has an input of output size, this number corresponds to the number of different classifications. There was no need to experiment with the input to this function since it was set to the number of classes we had, thus 29 was the input. Other possible parameters included Weights and Bias, which (as we previously have mentioned) are out of the scope of this assignment and left those parameters at their default values. Following the fully connected layer is the soft max layer. The soft max layer normalizes the fully connected layer and creates a single positive number used for classification probability. We used Matlabs: softmaxLayer function. This function did not take in any relevant inputs. The last layer is the classification layer. The classification layer uses the probability from the soft max layer to assign input into one of the mutually exclusive class and computes the loss. Matlabs classificationLayer function also does not have any relevant input parameters.

After an initial run with basic layer and option parameters, we got decent cross-validation results around 85 percent, but our validation results with the test images were very poor (approximately 35 percent). Therefore, we decided to add an image data augmenter to preprocess the training images. This would change the images in the dataset by rotating and translating randomly. This would help the CNN be more robust against images not in our dataset. This change to pre-processing significantly increased the accuracy of our CNN.

The primary options for the CNN we explored were: InitialLearnRate, LearnRateDropPeriod, MaxEpochs, MiniBatchSize, LearnRateSchedule, L2Regularization, Shuffle, and the ExecutionEnvironment. We felt that these were the major options that would affect the training accuracy. The default MaxEpochs is 30, but we found that 20 full passes of the training algorithm over the entire training set was sufficient enough for the model to learn from our data. The MiniBatchSize is the size of the training subset used to evaluate the gradient loss and update the weights of our model. After testing different sizes, we found that 64 was the most optimal for our dataset because it provided more updated information than the default 128 mini batch size. We set the Shuffle parameter to every-epoch because we did not want the model to learn the order in which the images appear. We played around with InitialLearnRate because if the learning rate is too low then training takes a long time, but if the rate is too high then training might reach a suboptimal result or diverge. We also wanted to know how changing the LearnRateSchedule would affect the training process. By default, the learning rate was constant, but theorized it might be beneficial to change the rate to decrease in piecewise manner in order to save time and increase accuracy. We determined the results from changing the learning rate were not as beneficial as we had hoped, so we kept this parameter at constant. We could also define a specific epoch that the learn rate would change at with learn rate drop period. L2Regularization changed the amount of weight decay. The last option was changed almost immediately was the ExecutionEnvironment. By default, CNN training runs on a single CPU, which would not take the computers entire resources; however, this took a tremendous amount of time. Thus, we switched to parallel computing on multiple CPUs.

2) Accuracy: Accuracy is the primary metric we looked at in determining the best parameters for each model, as well as determining the overall best model for ASL alphabet recognition. In this case, accuracy refers to a trained models ability to correctly classify an image of an ASL sign with in the test set of images. Our project did not use the data sets test images; instead, we used the remainder of the unused training images as the test set. We felt this provided a more robust accuracy result. Calculating the accuracy for the CNN required use of Matlabs classify function, which gets the label predictions for all of the images in the testing set, compares them to the correct corresponding labels, sums together the number of correctly classified images and divides by the total number of testing images. In calculating the accuracy for the ECOC and BOF models, we used the sum of the diagonal of the confusion matrices and divided by the number of the test images. The confusion matrix used the result of Matlabs built-in predict function.



Training Progress (23-Apr-2019 00:02:02)

Results

Validation accuracy:	N/A
Training finished:	Reached final iteration

Training Time

Start time:	23-Apr-2019 00:02:02
Elapsed time:	1243 min 41 sec

Training Cycle

Epoch:	20 of 20
Iteration:	25360 of 25360
Iterations per epoch:	1268
Maximum iterations:	25360

Validation

Frequency:	N/A
Patience:	N/A

Other Information

Hardware resource:	Multiple CPUs
Learning rate schedule:	Constant
Learning rate:	0.001

Accuracy

Training (smoothed)
Training
Validation

Loss

Training (smoothed)
Training
Validation

Fig. 4: Training plot for the CNN.

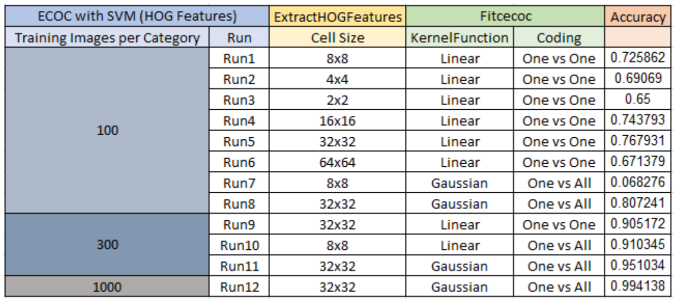
3) Loss: Although we did not use loss in determining the best model, we did take it into consideration. We only looked at the loss of the CNN because it was only relevant to this model. Over time we expected loss to decrease in value, producing a graph similar to that of exponential decay. Unlike accuracy, loss is a summation of errors and not a percentage. Reducing the loss value depends on changing the CNNs weight vector values through different optimization techniques. A plot of our most accurate CNN models loss after the training phase can be seen in Figure 4. We took loss into consideration while creating, training, and testing our custom CNN to attain the best accuracy with a minimal loss during the training phase.

4) Confusion Matrix: We used confusion matrices for both of the SVMs to see an overview of the performance of the classification algorithm with respect to the different categories. Calculating a confusion matrix can show where the classification model is succeeding, but more importantly, what kind of errors the model is making. We used these matrices to optimize our accuracies in the SVM methods while testing different training options. We used loss to improve optimization for the CNN, thus had no need for a confusion matrix. Matlab provides a function that creates the confusion matrix called: confusionmat which takes in an input of the predicted test image labels (determined using the predict function) and the correct corresponding labels.

C. Major Results

Each classification model went through vigorous experimenting to find the best accuracy for the model. Each model would then be compared to the accuracy of the other models to find the overall best performing accuracy of ASL alphabet recognition.

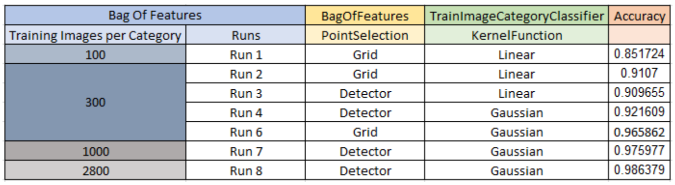
1) Error-Correcting Output Code with SVM Binary Learners: After testing the different cell sizes, we found that using a 32 by 32 cell produced the best accuracy. Once we increased our number of training images per category from 100 to 300, we saw a significant increase in accuracy when using a Gaussian filter combined with the one vs all method. As you can see in Figure 5, run 11 shows this with a 95 percent accuracy. At this point, we decided to increase the number of training images per category to 1000 with a 32 by 32 cell size, Gaussian filter, and onevsall method. We tested the 200 images per category (the same set tested on the other models) on this trained SVM model which produced a 99.4 percent accuracy.



ECOC with SVM (HOG Features)		ExtractHOGFeatures	Fitcecoc		Accuracy
Training Images per Category	Run	Cell Size	KernelFunction	Coding	
100	Run1	8x8	Linear	One vs One	0.725862
	Run2	4x4	Linear	One vs One	0.69069
	Run3	2x2	Linear	One vs One	0.65
	Run4	16x16	Linear	One vs One	0.743793
	Run5	32x32	Linear	One vs One	0.767931
	Run6	64x64	Linear	One vs One	0.671379
	Run7	8x8	Gaussian	One vs All	0.068276
	Run8	32x32	Gaussian	One vs All	0.807241
300	Run9	32x32	Linear	One vs One	0.905172
	Run10	8x8	Linear	One vs All	0.910345
	Run11	32x32	Gaussian	One vs All	0.951034
1000	Run12	32x32	Gaussian	One vs All	0.994138

Fig. 5: Accuracy results for the ECOC.

2) Bag Of Features with Speeded Up Robust Features Descriptors: Our initial runs tested whether using a Grid or a Detector to find SURF features generated better accuracy. Using the Detector gave better accuracy for our small training set. Then we tested if using a Gaussian filter would perform any better, which it did. The results from this test can be seen in Figure 6. From there, we increased our training set to first 1000 images per category then finally to 2800 training images per category while using a Detector and a Gaussian filter. Testing using the remaining 200 images per category finished with a 98.6 percent accuracy.



Bag Of Features		BagOfFeatures	TrainImageCategoryClassifier	Accuracy
Training Images per Category	Runs	PointSelection	KernelFunction	
100	Run 1	Grid	Linear	0.851724
300	Run 2	Grid	Linear	0.9107
	Run 3	Detector	Linear	0.909655
	Run 4	Detector	Gaussian	0.921609
	Run 6	Grid	Gaussian	0.965862
1000	Run 7	Detector	Gaussian	0.975977
2800	Run 8	Detector	Gaussian	0.986379

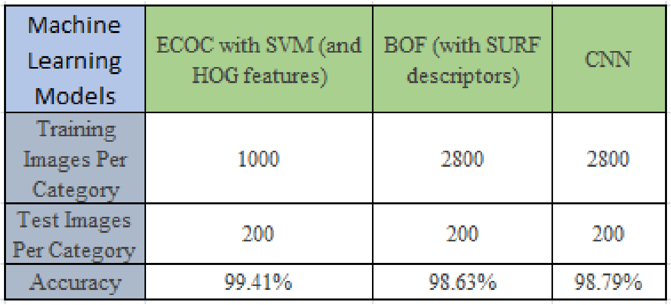
Fig. 6: Accuracy results for the BOF.

3) Convolutional Neural Network: After experimenting with the different layers of the CNN, we decided to go with the following (in chronological order): image input layer, convolutional layer with 4 filters with a size of 4x4 and zero padding, a batch normalization layer, a ReLU layer, a max pooling layer with a stride of 2, followed by another convolutional layer with 8 filters each with a size of 4x4 and zero padding, another batch normalization layer, a ReLU layer, followed by another max pooling layer with the same inputs as the previous one, then the final convolutional layer with 16 filters with a size 4x4 and zero padding, a batch normalization layer, ReLU layer, then the fully connected layer, the softmax layer, and finally the classification layer. This exact sequence of layers and inputs yielded the best accuracy for CNN. You can see the results of our experimenting with different layers and parameters in Figure 8. Along with these layers, we decided our options would be: sgdm, 20 max epochs, 0.001 initial learn rate, 64 mini batch size, and default for learn rate schedule, learn rate drop period, and L2 regularization. We trained 2800 images per category and tested the remaining 200 which produced a 98.7 percent accuracy.

4) Machine Model Comparison: We took the best performing from each model and compared the accurracies. As you can see in Figure 7, extracting HOG features with an error correcting SVM did the best with a 99.4 percent accuracy.

D. Analysis

1) Error-Correcting Output Code with SVM Binary Learners: The range of cell size values we experimented with were: 2x2, 4x4, 8x8, 16x16, 32x32, and 64x64. Interestingly, 32 by 32 cell size yielded the best results with a 76.8 percent accuracy with 100 training images per category. We suspect the success of this large cell size is due to each training image being a close up of the hand sign and thus the large spatial information consisted primarily of the object of interest. 64x64 was likely too large in comparison to the 200x200 images and likely lost information about the hand sign. We suspected SVM learners that also applied a Gaussian filter improved accuracy due the smoothing effect that reduced noise. We expected onevsall would outperform onevsone since it seemed to more closely address our specific use case, were if one hand sign is recognized then it cannot be any of the other hand signs so assign them to a negative class as opposed to ignoring them.

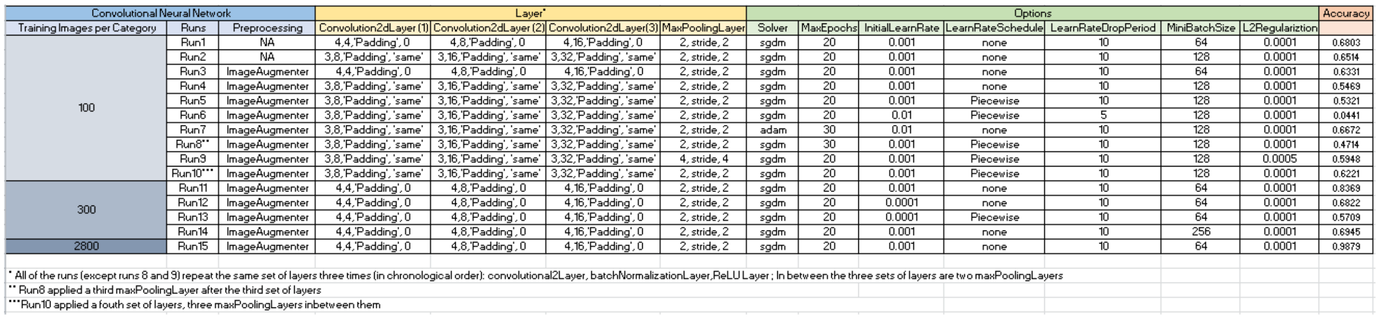


Machine Learning Models	ECOC with SVM (and HOG features)	BOF (with SURF descriptors)	CNN
Training Images Per Category	1000	2800	2800
Test Images Per Category	200	200	200
Accuracy	99.41%	98.63%	98.79%

Fig. 7: Accuracy results for the machine learning models.

2) Bag Of Features with Speeded Up Robust Features Descriptors: Applying the Gaussian filter likely increased the accuracy due to filtering out noise. The point selection method from Grid to Detector is something we are inconclusive about. While the interest points provided by Grid yielded better results, the huge number of points filled up the entirety of our memory for any run beyond 300 training images per category. Because of this inefficiency with use of space and our lack of a sufficient amount of memory we decided the Detector was the most accurate, but we suspect if there was sufficient memory then Grid would outperform Detector at 2800 training images per category.

3) Convolutional Neural Network: Augmenting the training data set in terms of rotation and translation distinctly made the model more robust. This is likely because the network could end up just memorizing the training set and any slight deviation could be wrongly classified (this is shown by our results from a test we previously mentioned). As to why the number of filters from 4 to 8 to 16 did better than 8 to 16 to 32, is something we are not sure on. Originally we expected the larger number of filters (which directly corresponds to the number of neurons at that layer) to yield higher accuracy. We felt that more neurons meant more knowledge from the images was being extracted. We suspect now it might have more to do with essential information being spread out too far across the neurons and basically get lost by the time the fully connected layer is reached.The larger filter size, 4x4 compared to 3x3, is something we correctly hypothesized would yield better results. We hypothesized the smaller filter would not able to gather appropriate information about the images since they were zoomed in to the hand sign. This reasoning has a similar basis to our reasoning behind the cell size parameter in ECOC. The MiniBatchSize was found to be most successful at 64, since this is the number of images used by stochastic gradient descent to update the parameters of the CNN, we were surprised to see a smaller mini batch size yield the best



Convolutional Neural Network			Layer*				Options							Accuracy
Training Images per Category	Runs	Preprocessing	Convolution2dLayer (1)	Convolution2dLayer (2)	Convolution2dLayer(3)	MaxPoolingLayer	Solver	MaxEpochs	InitialLearnRate	LearnRateSchedule	LearnRateDropPeriod	MiniBatchSize	L2Regulariztion	
100	Run1	NA	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.001	none	10	64	0.0001	0.6803
	Run2	NA	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	20	0.001	none	10	128	0.0001	0.6514
	Run3	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.001	none	10	64	0.0001	0.6331
	Run4	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	20	0.001	none	10	128	0.0001	0.5469
	Run5	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	20	0.001	Piecewise	10	128	0.0001	0.5321
	Run6	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	20	0.01	Piecewise	5	128	0.0001	0.0441
	Run7	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	adam	30	0.01	none	10	128	0.0001	0.6672
	Run8**	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	30	0.001	Piecewise	10	128	0.0001	0.4714
	Run9	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	4, stride, 4	sgdm	20	0.001	Piecewise	10	128	0.0005	0.5948
	Run10***	ImageAugmenter	3,8,'Padding', 'same'	3,16,'Padding', 'same'	3,32,'Padding', 'same'	2, stride, 2	sgdm	20	0.001	Piecewise	10	128	0.0001	0.6221
300	Run11	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.001	none	10	64	0.0001	0.8369
	Run12	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.0001	none	10	64	0.0001	0.6822
	Run13	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.0001	Piecewise	10	64	0.0001	0.5709
	Run14	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.001	none	10	256	0.0001	0.6945
2800	Run15	ImageAugmenter	4,4,'Padding', 0	4,8,'Padding', 0	4,16,'Padding', 0	2, stride, 2	sgdm	20	0.001	none	10	64	0.0001	0.9879

* All of the runs (except runs 8 and 9) repeat the same set of layers three times (in chronological order): convolutional2Layer, batchNormalizationLayer,ReLU Layer ; In between the three sets of layers are two maxPoolingLayers

** Run8 applied a third maxPoolingLayer after the third set of layers

***Run10 applied a fouth set of layers, three maxPoolingLayers inbetween them

Fig. 8: Accuracy results for the CNN.

results, we expected that a larger number images to update the parameters with would provide a better and more accurate result.

4) Overall: It is interesting that the ECOC with SVM binary learners and HOG features did better than the CNN, since this model only used 1000 training images per category to yield the highest results. Whereas both of the other models had more images to train from (2800 per category). We expected the CNN to outperform the other models due to its popularity and success in the field of computer vision. This is likely because our knowledge of CNN and BOF is limited, whereas we knew more about HOG features and SVM allowing us to better fine tune the parameters. It would have been interesting to see if ECOC would have remained the most accurate model if we had been able to completely train the BOF model with a Grid point selection method. Also, due to our limited resources (mentioned in Major Challenges and Solutions) and limited time, it was not within the scope of this project to test every parameter, and likely some of the parameters we chose to not adjust for the other models would have likely increased the accuracy.

V. Conclusion and Future Works

Sign language is a visual language that uses hand signs, facial expressions, and body postures to convey information. We used computer vision and machine learning to create a model for sign language recognition. Our original goal was to produce an accurate classification model to eliminate the language gap between ASL users and non-users to ease communication. Each of our proposed models performs exceptionally well to identify the ASL alphabet. The best accuracy of all the models is using error-correcting output code using multiple binary support vector machines trained on histogram of oriented gradient features which produced a 99.4 percent accuracy. However, our model only recognizes the letters in the alphabet for ASL. It is possible to spell out every word using the alphabet, but this would not be very helpful in the real world where hand signs usually use represent words or phrases. Thus, to improve on our model, it would be practical to be able to recognize many different words in the language and not just the alphabet. Also, sign language users may sign words differently than others; the model must be made more robust in order to still be able to accurately recognize these words. It is also possible to sign without using any facial expressions or body movements, but doing so might be confusing or cause misunderstandings. Therefore, creating a facial expression, body posture model, as well would be beneficial in recognizing the entire message in ASL. We propose detecting interest points to find the regions of interest and using that information as inputs to our CNN and ECOC models. For CNN there is a specific version called region convolutional neural network (R-CNN) that achieves exactly that. The ability to accurately recognize hand signs, facial expressions, and body postures will close the language gap between ASL users and non-users to make communication easier.

References

[1] Akash, (2018 May). *ASL Alphabet*. Retrieved April 2, 2019 from https://www.kaggle.com/grassknoted/asl-alphabet

[2] MathWorks, (2019). *Classification ECOC*. Retrieved April 12, 2019 from https://www.mathworks.com/help/stats/classificationecoc.html

[3] MathWorks, (2019). *Classification SVM*. Retrieved April 12, 2019 from https://www.mathworks.com/help/stats/classificationsvm.html

[4] MathWorks, (2019). *Image Category Classification Using Bag of Features*. Retrieved April 12, 2019 from https://www.mathworks.com/help/vision/examples/image-category-classification-using-bag-of-features.html

[5] Wikipedia, (2019). *Speeded up robust features*. Retrieved April 26, 2019 from https://en.wikipedia.org/wiki/Speeded_up_robust_features

[6] MathWorks, (2019). *Convolutional Neural Network*. Retrieved April 25, 2019 from https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[7] MathWorks, (2019). *ExtractHOGFeatures*. Retrieved April 15, 2019 from https://www.mathworks.com/help/vision/ref/extracthogfeatures.html

[8] MathWorks, (2019). *fitcecoc*. Retrieved April 15, 2019 from https://www.mathworks.com/help/stats/fitcecoc.html

[9] MathWorks, (2019). *bagOfFeatures*. Retrieved April 16, 2019 from https://www.mathworks.com/help/vision/ref/bagoffeatures.html

[10] MathWorks, (2019). *trainImageCategoryClassifier*. Retrieved April 16, 2019 from https://www.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html

[11] MathWorks, (2019). *Create Simple Deep Learning Network for Classification*. Retrieved April 17, 2019 from https://www.mathworks.com/help/deeplearning/examples/create-simple-deep-learning-network-for-classification.html

[12] MathWorks, (2019). *trainingOptions*. Retrieved April 17, 2019 from https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html

[13] MathWorks, (2019). *trainNetwork*. Retrieved April 17, 2019 from https://www.mathworks.com/help/deeplearning/ref/trainnetwork.html

[14] MathWorks, (2019). *templateSVM*. Retrieved April 16, 2019 from https://www.mathworks.com/help/stats/templatesvm.html