ASL Translator

Juan Ceja-Vargas
CS4210
Cal Poly Pomona
Pomona, United States
jcejavargas@cpp.edu

Tai-Wei Chien
CS4210
Cal Poly Pomona
Pomona, United States
tchien@cpp.edu

Eugene Mondkar
CS4210
Cal Poly Pomona
Pomona, United States
ysmondkar@cpp.edu

Abhishek Vangipuram
CS4210
Cal Poly Pomona
Pomona, United States
avangipuram@cpp.edu

***Abstract*—Convolutional Neural Networks are a very powerful algorithm used in the field of computer vision. This paper uses a CNN to build a model that converts ASL(American Sign Language) to text. This enables none ASL speakers to communicate with ASL speakers and vice-versa. The model is susceptible to over fitting without the implementation of an effective image pipeline. The best image pipeline to use requires an extensive trial and error analysis.**

***Index Terms*—Computer Vision, ASL, Convolutional Neural Network, Hand Gestures, Machine Learning**

I. Introduction

The problem of interest is that of visually recognizing specific hand gestures that match the official hand gestures of the American Sign Language (ASL). Sign language is used throughout the world to help people who are deaf communicate. Unfortunately, most people do not understand sign language, specifically those who are not deaf. The ability to understand sign language can be very helpful and make society more adaptive to the needs of the deaf and to expand their access and ability to contribute to culture. Our project hopes to further develop solutions based on machine learning algorithms and contribute to the field of computer vision in specific applications in classifying ASL hand gestures.

II. Related Work

Our goal is to build a system which can solve the ASL alphabet recognition problem. To accomplish this system, an alphabet translation model will be the most important component. In the field of Computer Vision, CNN is a widely used algorithm to solve various problems in the field. Hence, we want to implement a robust CNN model as the core pre-processor in our system. After doing research on CNN's, we figured out the kinds of problems/scenarios that are most suitable for using a CNN the scenarios are as followed:

- Some patterns are very small compared to the whole image
- The same patterns appear in different regions of the image
- Sub-sampling the pixels will not change the object

The ASL data set we used satisfied the traits above, which gave us more confidence on using a CNN model in our project. Inspired by the CNN implementation of the AlphaGo project, we believe the reasons that AlphaGo succeeded is due to the minimal noise in the go board input data. Therefore, we decided to create an image pipeline to clean our data. We used the open source libraries MediaPipe and rembg, to detect the hand region in the image and remove the background of the hand image respectively. The models used by the MediaPipe library contain TFLite model (lite), TFLite model(Full), TF.js model which are provided by the Google Tensorflow library. The model used by the rembg library is based on the U2-Net paper. CNN's have a weakness related to the spatial various problem which happens because CNN's do not have a strong ability to recognize the same pattern in different spatial aspects. To make our CNN model more general, we referenced the Google DeepMind paper regarding Spatial Transformer Networks. We used the STL (spatial transform layer) as one of our data augmentation methods, we use this approach to enhance the spatial recognizing ability of our model. Our data prepossessing strategy is to normalize the illumination, skin colors and vary the spatial features of our data sets. We use the color format converter which was proposed by Mayand Kumar to make a skin mask effect. The spatial aspect of data augmentation are implemented by the STL and we use some image transformation functions from tensorflow.keras such as zoom ranging, width height shifting, and shear ranging to construct our image-pipeline.

III. Data Set

The project uses 4 different data sets. Our training data set consists of 87000 images with a resolution of 200X200 pixels. We select 57720 images from them and discard the images which cannot go through our image-pipeline. The images represent the 26 letters of the American Sign Language (ASL) alphabet. The data set is separated into 26 folders one for each of the above characters which are the classes of our data set. Each folder contains around 3000 images of a hand performing the gestures that represent the specific ASL characters. The test data set has a good variety of images of the hand gestures at different angles. However, some possible issues with the data set is that it doesn't contain any images of darker skin tone hands performing the gestures. This means that our model will perform worst when testing ASL gestures on hands with darker skin complexions. All the images also have similar lighting conditions which will mean that our model will be biased towards images with similar lighting conditions. From this training data set 20% of the data set was set aside for data validation. The other three data sets we used, which are data set 2, data set 3, data set 4 are

used for testing and evaluation. Out of the three testing data sets, data set 4 has the least amount of test images with 30 images per each of the 26 letters of the ASL alphabet but the images are very diverse. The data set has varies backgrounds, lighting conditions and orientations with regards to the images of the ASL hand gestures. Data set 3 is very large but contains duplicated images from our training data set. Lastly data set 2 is skewed and the images have a different size distribution as demonstrated in the figures 1 and 2 below.

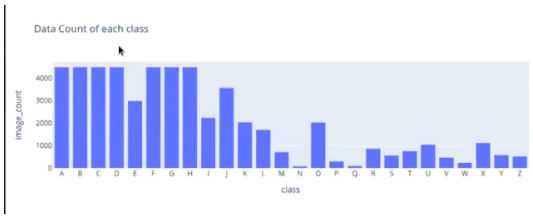

Fig. 1. Data set 2 class distribution

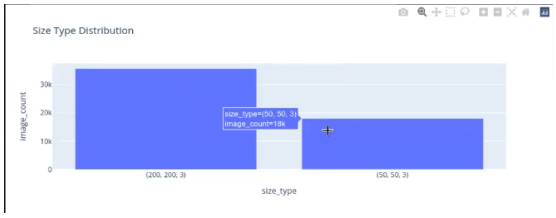

Fig. 2. Data set 2 size distribution

From these three data sets we randomly picked 555 images for each of our 26 classes for a data set size of 14430 test images. Each data set was assigned a weight which determine the probability of choosing an image from that data set. Data set 4 was assigned the highest weight because it is the best data set we have, it's followed by data set 2, and finally by data set 3.

IV. Methodology

We designed the data set to match the classic MNIST data set which is a data set of hand written numbers from 0-9. Therefore,in the data set, each element is labelled from 0 to 25 for both the training and test data samples, which corresponds to the letters A-Z in the English alphabet.

A. Data Prepossessing

The core task of the problem we are going to solve is to build a model that can recognize image data. As it is a typical Computer Vision problems, the image prepossessing will be a vital phase. We used the MediaPipe library to recognized and capture hands in our training and testing images. We also used the rembg library to remove the background of our images ounce we got our region of interest which is the hand from the MediaPipe api.There are too many factors that we cannot control for each image. We can use image normalization procedures like "Gray-scale Normalization" or data augmentation procedures like zooming, rotation and flipping to avoid over-fitting when working on image recognition problems. To implement this procedures we created a tool kit of data normalization and augmentation functions for used in the prepossessing phase. From this functions we created two image prepossessing pipelines a **general pipeline** with the following steps

- region of interest (roi) normalization (by MediaPipe)
- background normalization (by rembg)
- skin normalization
- channel normalization
- resolution normalization

and a **training pipeline** with the following steps

- background normalization (by rembg)
- region of interest (roi) normalization (by MediaPipe)
- skin normalization
- channel normalization
- resolution normalization

The steps for the pipelines were chosen by referencing varies computer vision papers regarding image recognition. The transformations that an image goes though as it passing through the general pipeline is demonstrated in the figure 3 below.

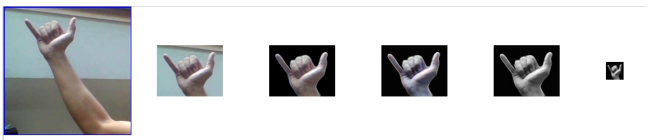

Fig. 3. General image pipeline

We also utilize a spatial transformer network to enhance the ability of our CNN model to handle images of the ASL hand gestures at varies spatial orientations.

B. Model Building

For our regular CNN model we reference the paper proposed by Mayand Kumar. The first step towards building a model for us was to initialize the model as sequential. The next step is to create our first layer of the CNN to perform convolution operations. We then created the input layer of our CNN which takes 32 filters of size 3x3 to detect the patterns and features from the image data. The layer takes input as a batch of images. In our case, the batch size is 128. The shape of our image for the input is (28, 28, 1) where the height and width of an image is 28 pixels with one color channel because the images are gray scaled. We set the stride to 1 and disable the padding operation. Each filter produces a feature map after the convolution operation over the image with an output volume of 26x26x1. After the convolution operation, the ReLU activation function is applied to increase non-linearity of our CNN model. We setup all the convolutional layers with same stride, padding and activation function settings. Next, we did batch normalization to keep the output of the layer in the same range to ensure an unbiased result. After this, the

output of our first convolution layer with volume 26x26x32 is passed to the max-pooling operation in the next layer. Max pooling helped us get rid of unnecessary features and helped in preventing over-fitting because it avoids a few parameters while performing a pooling operation to get a pooled matrix. Max pooling is used to focus only on the relevant information from the image by extracting only higher values from a portion of input by using an empty feature detector. Pooling is also called down sampling. We are taking strides of size 2 and a filter feature detector size of 2x2. As a result, the height and width are reduced by a factor of 2. Thus, the output volume will be 13x13x32. Now we have a second convolutional layer, which consists of 1 convolution 2d layer, one dropout and one max pooling. The input of this layer is the output of the previous layer, which is 13x13x32. The kernel size is 3x3, with 64 different feature detectors. Then, the output volume is 11x11x64 and we applied a dropout layer with proportion 0.2. Now we again applied batch normalization and passed the output volume 11x11x64 as input for max pooling with 2x2 filter size, and we get the output as 5x5x64. Now we have our last convolutional layer, the number of feature filter we took is 128. And again we do batch normalization and max pooling with a 2x2 filter. Here we've finished the feature extraction by using the CNN layers. We do a flattening operation for the feature map we obtained from our last convolutional layer and feed the output to the dense layer (Hidden layer in ANN) with 512 neurons. The output of the dense layer is passed through the ReLU activation function to increase the non-linearity of the model. Next, we use a dropout layer with a dropout rate of 0.25, which means 25% of random neurons are switched off so that we can prevent over-fitting of our model. Now the output of this dense layer is fed to the final output dense layer with 26 neurons. This layer has 26 neurons representing 26 classes. The activation function we used here is Softmax because this is a multi-class classification problem. The source code for the model is shown in figure 7.

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.BatchNormalization())

model.add(layers.MaxPool2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.BatchNormalization())

model.add(layers.MaxPool2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool2D((2, 2)))

# finish feature extraction
model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(26, activation='softmax'))
```

Fig. 4. Regular CNN model Source Code

For our STL model we performed a spatial transformer in the beginning layers. We tried to implement the core concept of the Spatial Transformer Networks paper, provided by Google DeepMind. We believe that a spatial transformer can help our CNN to improve the abilities in spatial aspect, so we've tried our best to implement it in Python. In the end, we referenced from the source code of the project on GitHub. But this project is based on the MNIST data set and we don't have enough time and knowledge to fine-tune the parameters in the implementation code of the STL to find the best parameters for our data set. So, we just used the original parameter settings from the project. After initializing the spatial transform layers, we are supposed to add the CNN model structures right after the STL. The convolutional layers we used for our STL model are similar to the convolutional layers in our regular model. There are two differences between them: first We've done several operations for cleaning our input data, we assume that the all the pixels left in the image are important patterns. Therefore, we remove all the layers of max pooling and Dropout. Second we do some modification on the Dense layers. We create one 1 dense layer with 512 neurons and ReLU activation function and add a dropout layer with 0.5 dropout rate after it. Then, we add another Dense layer with 256 neurons and ReLU activation function, and also add a dropout layer after it. We set the dropping proportion as 0.25 this time. Finally, we create our output Dense layer with 26 output neurons and use Softmax activation function as well.

```
input_layers = layers.Input((size, size, 1))
x = stn(input_layers)

x = layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu',
                  kernel_initializer=initializers.glorot_uniform(seed=STN_SEED))(x)
x = layers.BatchNormalization()(x)

x = layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu',
                  kernel_initializer=initializers.glorot_uniform(seed=STN_SEED))(x)
x = layers.BatchNormalization()(x)

x = layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu',
                  kernel_initializer=initializers.glorot_uniform(seed=STN_SEED))(x)
x = layers.BatchNormalization()(x)

x = layers.Flatten()(x)

x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)

x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.25)(x)

output_layers = layers.Dense(26, activation="softmax")(x)

model = tf.keras.Model(input_layers, output_layers)
```

Fig. 5. STN model Source Code

C. Model training

After we defined our model structure for the regular CNN and STL model's respectively, we perform the training operation. In this phase, we use the early stopping and the ReduceLROnPlateau callback function API from tensorflow.keras. For the other parameters, we set the batch size as 128, the max epoch as 100, and Adam as the optimizer. We set 20% of our training data set to be used as validation data as we mention in section III.

D. Model Evaluation

For the evaluation, we choose Categorical Cross entropy as the loss function and accuracy as the metrics. We also use precision, recall and F1-score of each alphabet to do further evaluate our regular CNN model and STL model.

V. Results

After doing the model structure selection from the 9 different STL models and 3 regular CNN models, we've trained our best regular model and our best STL model.

A. Training & Validation Data

The accuracy according to training and validation data of the regular model and the STL model are shown in Figure 6. The epoch accuracy evolution of the regular model and the STL model are shown in fgure 7 and fgure 8 respectively.

	Training Data	Validation Data
Normal Model	0.9917	0.9864
STL Model	0.9871	0.9883

Fig. 6. Regular vs STN models

Fig. 7. Epoch regular model

Fig. 8. Epoch stl model

B. Test Data

After evaluating the model with our test data, the result of each model is shown in figure 9, and the prediction performance of each letter of the regular and STL model's are shown in figure 10 and figure 11 respectively.

	Accuracy	Precision	Recall	F1-Score
Normal Model	0.8949	0.90	0.89	0.89
STL Model	0.9060	0.91	0.91	0.91

Fig. 9. Testing data accuracy

The maximum accuracy obtained on the evaluation of the testing data is 90.6%, which is provided by the STL model. The best regular model has an accuracy of 89.5% on the testing set. Until now, the models have not shown over-fitting directly on our testing data. The reason why we created more instances of the STL models is that the literature on CNN with image recognition problems suggests that the STL model should be much better than the regular CNN model for our particular type of problem. Our hypothesis as to why that is not the case for our models is that we need to tune our model more. Since our reference for our STL models came from papers using the MNIST data set, we think that the parameters provided in those papers were specially tuned for that MNIST data set. We need to test a lot more than nine STL models to determine the best parameters for our problem. We also observe that our models had some difficulties in translating some specific letters. There is a skew in the distribution of failed prediction count for each letter as shown in figure 10 and figure 11. It is an implicit sign of over-fitting.

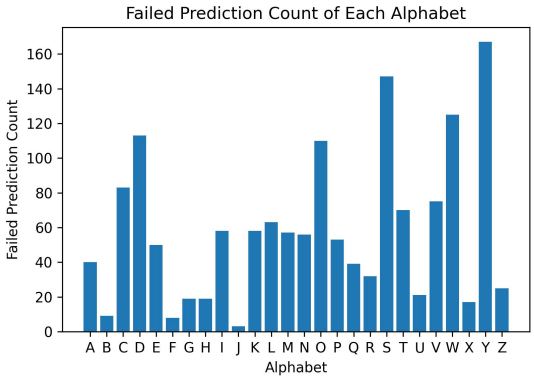

Fig. 10. Normal model fails

C. Our Own Hands Data

The accuracy of our model with the test data set was quite good at 90.6% for the STL model and 89.5% for the regular model. However, when we evaluated our model with new images that we created by taking photos of own hands, our accuracy plummeted to about 35% for our best model. This means that our model is over fitting to our training data. This also implies that our test data is very similar to our training

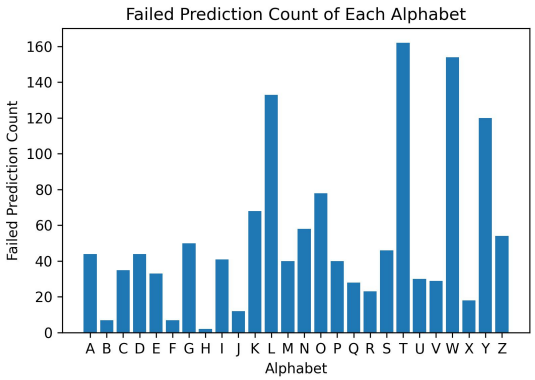

Fig. 11. STL model fails

data even after selecting different images from various data sets.

VI. Conclusion

This project demonstrated the importance of the prepossessing phase in the machine learning pipeline. Over 60% of our effort went into creating, cleaning, and augmenting our data sets. This proves crucial in helping increase the accuracy of our models when using the training data set we created by combining various images from other data sets. But the accuracy didn't translate to the new testing data which we created manually. This implies that our models are over-fitting. We have some hypotheses as to why our models are over-fitting:

- Our issue wasn't the quantity of data but the quality; Either our training or testing data are too similar
- The data set volume of 57k images are still too few for training a general hand gesture recognition model.
- We do too much high constraint prepossessing in our image pipeline. The noise in the original data set might be a necessary factor for a robust model.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] Mayand Kumar; Piyush Gupta; Rahul Kumar Jha; Aman Bhatia; Khushi Jha; Bickey Kumar Shah, "Sign Language Alphabet Recognition Using Convolution Neural Network", doi: 10.1109/ICICCS51141.2021.9432296

[4] Zhaoyang Yang, Zhenmei Shi, Xiaoyong Shen, Yu-Wing Tai, "SF- Net: Structured Feature Network for Continuous Sign Language Recognition". *arXiv preprint arXiv:1908.01341v1, 2019*.

[5] Ashish S. Nikam, Aarti G. Ambekar, "Sign Language Recognition Using Image Based Hand Gesture Recognition Techniques", *Online International Conference on Green Engineering and Technologies,* Available doi: 10.1109/GET.2016.7916786.

[6] Rajesh, X. Mercilin Raajini, K. Martin Sagayam, Hien Dang,"A statistical approach for high order epistasis interaction detection for prediction of diabetic macular edema",Informatics in Medicine Unlocked, Volume 20,2020,100362,ISSN 2352-9148.

[7] M. Ashok Kumar, Jeevan Babu Maddala, K. Martin Sagayam (2021) "Enhanced Facial Emotion Recognition by Optimal Descriptor Selection with Neural Network", *IETE Journal of Research*, doi: 10.1080/03772063.2021.1902868

[8] Aditya Das, Shantanu Gawde, Khyati Suratwala and Dr. Dhananjay "Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images".In: *International Conference on Smart City and Emerging Technology (ICSCET), 2018* . Available doi: 10.1109/ICSCET.2018.8537248.

[9] Vivek Bheda and Dianna "Using Deep Convolutional Networks for Gesture Recognition in American Sign Language". In: *CoRR* abs/1710.06836 (2017). arXiv: 1710.06836. url: [http://arxiv.org/abs/1710.](http://arxiv.org/abs/1710) 06836.

[10] PADMAVATHI . S, SAIPREETHY.M.S, V. "Indian sign language character recognition using neural networks". *IJCA Special Issue on Recent Trends in Pattern Recognition and Image Analysis, RTPRIA*.

[11] https://Www.Ucbmsh.Org/Colleges-In- Dehradun/Courses/Agriculture-Courses-In-Dehradun/Indias-First Sign-Language-Isl-Dictionary [image]

[12] SAKSHI GOYAL, ISHITA SHARMA, S. "Sign language recognition system for deaf and dumb people".In: *International Journal of Engineering Research Technology*

[13] "David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis". Nature volume 529, pages484–489 (2016), "Mastering the game of Go with deep neural networks and tree search"

[14] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane and Martin Jagersand University of Alberta, Canada, "U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection"

[15] "Max Jaderberg, Karen Simonyan, Andrew Zisserman, Koray Kavukcuoglu", Google DeepMind, London, UK, "Spatial Transformer Networks"

[16] Sayakpaul, Spatial Transformer Networks with Keras, "https://github.com/sayakpaul/Spatial-Transformer-Networks-with-Keras/blob/main/STN.ipynb"

VII. Supplementary Material Section

- Source Code Directory
- Latex Files