CSCI 1430 Final Project Report:
ASL Interpreter: CNN vs. Traditional CV Methods

Team name: Natalie King, Jinho Lee, Sylvia Watts, Domingo Viesca
TA name: Trevor Wiedmann. Brown University

Abstract

American Sign Language (ASL) is a vital tool for communication, yet barriers exist for those who do not understand it. Our project aims to create an ASL interpreter capable of identifying ASL letters from images, comparing the effectiveness of traditional computer vision methods against Convolutional Neural Networks. Using a Bag-of-Words approach with an SVM classifier and two CNN architectures—a custom-built model and a pretrained ResNet-18—we evaluated the performance of these methods on datasets. Despite challenges with training data size and diversity, our models achieved promising results, with the SVM and ResNet-18 models reaching accuracies of 85.7% and 69.48% respectively. The project also features a streamlined frontend application for real-time predictions, emphasizing usability and accessibility. This work highlights the strengths and limitations of both approaches, paving the way for improved ASL recognition systems and fostering better accessibility for ASL users.

1. Introduction

We wanted to create an ASL interpreter that can take in an image of a letter in ASL and identify which letter it is. We wanted to figure out whether traditional CV algorithms or a CNN would perform better at this task, so we implemented both.

The goal of our project is to find the most efficient way to bridge the gap between people who use ASL and those who are learning or don't know it. We wanted to test multiple ways to classify ASL signs to determine which approach would be best to solve this problem. We also wanted to make a clean, fast front end application so that users can figure out what letter a sign represents almost instantly. This would facilitate real time translation for those who want to learn ASL or need to communicate with someone in sign language.

For our traditional CV approach, we used the bag of words image classifier with an SVM classifier.

For our CNN approach, we used two different strategies. One was a fully built traditional convolutional neural network with linear convolutional and max pooling layers. The other model was our fine-tuned pretrained model which was a version of the ResNet-18 model. ResNet-18 is a CNN with 18 layers used for image classification for over 1,000 categories and trained on a large database. For our custom version we froze all the layers except the last and then replaced the final layer with a custom classifier for the specific number of classes in the dataset.

2. Related Work

We started out with our code from Homeworks 4 and 5 and improved/fine tuned the models as we went along.

For our front end, we used an open-source Python framework called Streamlit [2]. It includes plugins that allow us to collect real time images from users and immediately classify them using cached models.

Our training and testing data is entirely from a Kaggle dataset called asl alphabet. It includes 3000 images per letter, plus 3000 images for each of "space" and "nothing" [1]. Because of space limitations, we only used 40 images per character rather than the entire dataset.

Maybe add stuff about pytorch here–

3. Method

We implemented two approaches to classifying ASL characters: a more traditional CV method using bag of words with a SVM, and two CNNs. The goal was to see which could achieve higher accuracy on the limited data we were able to process.

Traditional
We started with the Linear SVM given to us in the homework. We experimented with different vocabulary sizes used in our bag of words implementation. We started with a vocabulary size of 200 and tried doubling it to 400 to see if more "words" in the vocabulary would produce a higher accuracy. Increasing the vocabulary size to 400 increased the vocabulary build time. For 200 words, the build time was 19 seconds. For 400 words, the build time was 44 seconds. Despite having more words and a longer build time, our accuracy dropped from

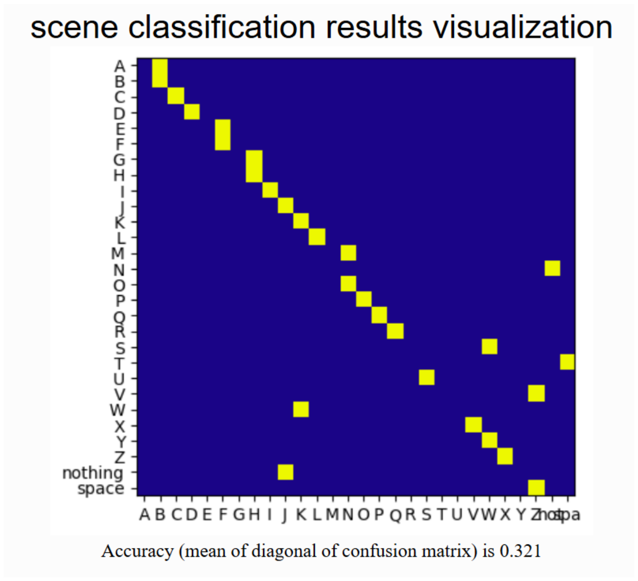


Figure 1. Accuracy with a linear kernel.

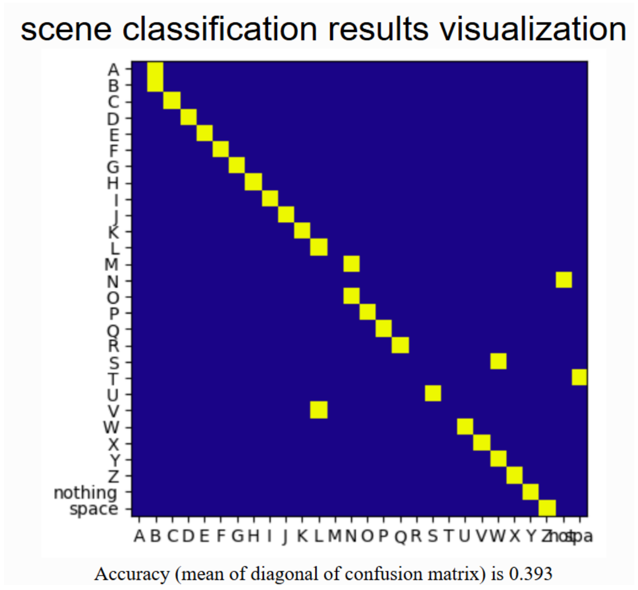


Figure 2. Accuracy with a radial basis function (RBF) kernel.

about 32% to about 21%. We then decided to investigate the effects of decreasing the vocabulary size to 100, which ran in in 22 seconds. We found that the relationship between vocabulary size and vocabulary build time is not linear. For a vocabulary size of 100, we had an accuracy of 32%. Since building the vocabulary for 200 words took less time and was just as accurate, we settled on 200 as our vocabulary size.

However, we wanted to improve this model, so we played around with several different kernels. We started with a Linear SCV. We tested this, a RBF kernel, a sigmoid kernel, and a polynomial kernel. We tested the polynomial kernel with degrees of 3, 6, and 10, and found that 10 degrees gave us the highest accuracy.

As you can see, our test accuracy using the polynomial kernel outperformed each other kernel by at least 46.4%. This result led us to select the SVM with a polynomial kernel with 10 degrees as our classifier for the model to be compared to the CNN and used in our application.

CNN

We started with a neural network similar to the one from homework 5. However, we soon decided to build our own using pytorch as it is more adapted for dynamic projects. We also wanted to explore how effective different frameworks were so we decided to use a pretrained adjusted model and our own built simple model. We wondered whether a model built specifically for this task would be more effective than a model that is pretrained on different data.

The architecture of our custom CNN specifically is: Two convolutional layers (32 and 64 filters, kernel size 3x3) with ReLU activations. MaxPooling layer halving the spatial dimensions. Fully connected layers reducing the feature space

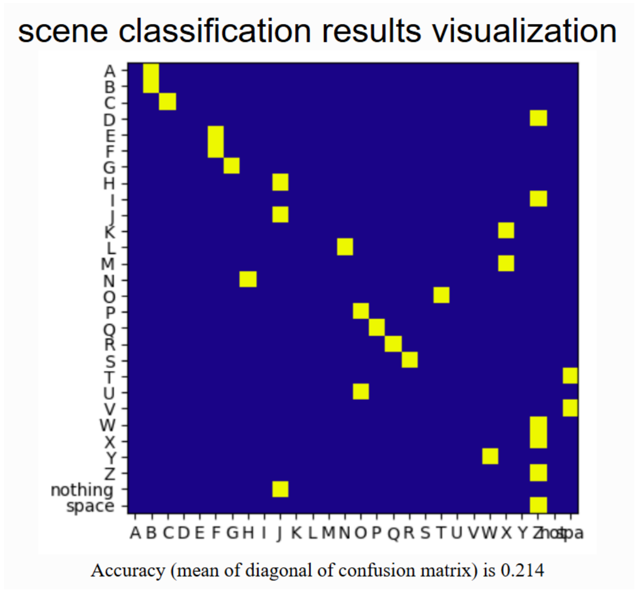


Figure 3. Accuracy with a sigmoid kernel.

(from 64x64x64 to 128 neurons) before outputting predictions for num_classes.

The architecture of our pretrained model is: A pretrained ResNet-18, freezing all layers except the final one. Final layer is a replacement of the original fully connected layer with a new linear layer mapped to num_classes.

This pretrained model allows us to use ResNet's feature extraction ability and its past training on a larger dataset while fine-tuning it for ASL classification.

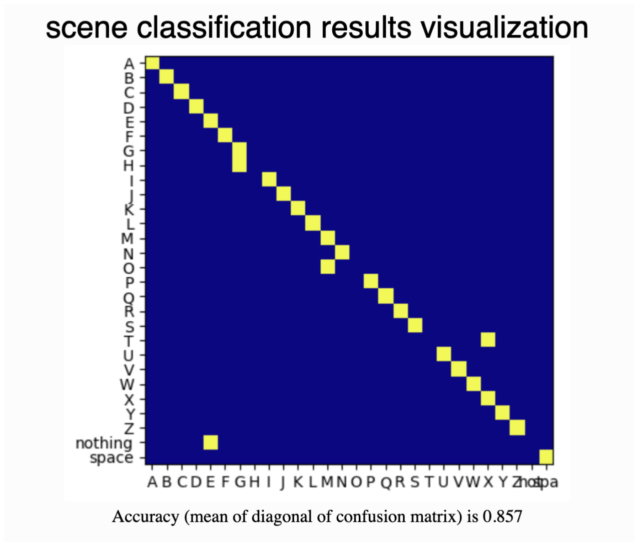


Figure 4. Accuracy with a polynomial kernel of degree 10.

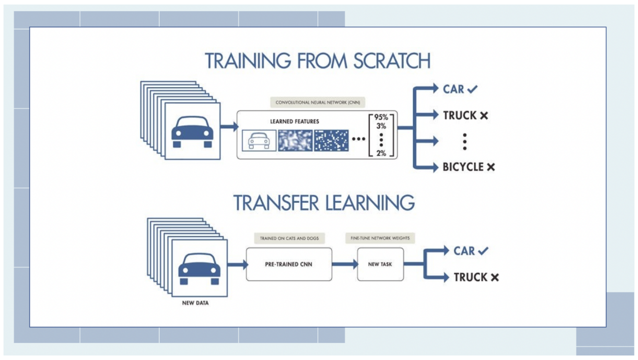


Figure 5. Diagram of how our two CNNs learn

4. Results

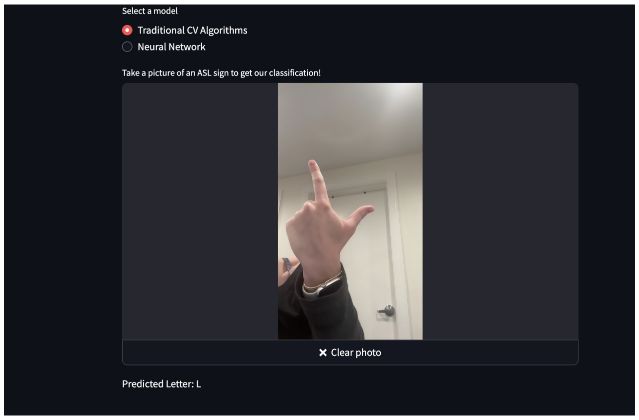
Traditional As shown above, our final test accuracy was 85.7%, meaning that on our test data, 85.7% of images of ASL characters were classified correctly. We used this model as the traditional model option for our application.
We wanted our frontend to be streamlined and accessible. We used a simple design that allows the user to quickly snap an image of an ASL character. The model then collects the image features and classifies it as a letter. This prediction is then written to the app for the user to view.

When we first made the front end, the app was extremely slow and took about five minutes to return a classification. We then figured out how to store our classifiers and vocab so that the model would only have to predict a single image rather than re-train. This made our predictions almost instantaneous which made the app much more usable.

CNN

When using a model similar to the one from Homework 5, we were getting very low accuracy - it was essentially random. This is because we were having some issues with the preprocessing of this specific data using a model similar to the one from Homework 5. We then decided to build our own two models and preprocessing of the data from the ground up, which led to higher accuracy.

For our pretrained model, after training it on 50 epochs our highest training accuracy was 87.86% and the corresponding validation accuracy was 69.48%. For our custom model, after 50 epochs, our highest training accuracy was 100% and our validation accuracy was 51.62%. We do think there is some overfitting in our models, particularly the custom model, as we could only provide 11 test images for each of the 28 classifications. This was an issue for us throughout as the data was very large and thus could not be pushed through github. Instead we had to download the images from


Select a model

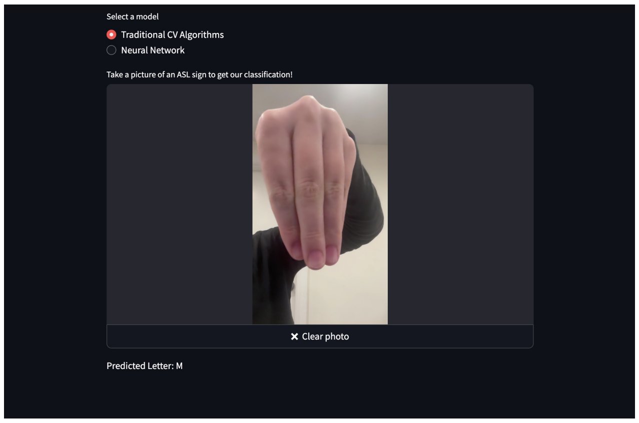
[✓ Traditional CV Algorithms]

Neural Network

Take a picture of an ASL sign to get our classification!

Predicted Letter: L

Figure 6. Frontend true letter ”L”.


Select a model

[✓ Traditional CV Algorithms]

Neural Network

Take a picture of an ASL sign to get our classification!

Predicted Letter: M

Figure 7. Frontend true letter ”M”.

Kernel	Accuracy
Linear	32.1%
Sigmoid	21.4%
RBF	39.3%
Polynomial	85.7%

Table 1. Results. SVM kernel testing results.

a shared google drive each time as the files would become corrupted through git. But generally, we built two models successfully that learned from the data.

Here is our architecture for our custom-built model:

```
def __init__(self, num_classes):
    super(ASLCNN, self).__init__()
    # conv
    self.conv1 = nn.Conv2d(3, 32,
        kernel_size=3, stride=1, padding=1)
         # output: 32x128x128
    self.conv2 = nn.Conv2d(32, 64,
        kernel_size=3, stride=1, padding=1)
         # output: 64x128x128
    self.pool = nn.MaxPool2d(kernel_size=2,
         stride=2)  # half size: 64x64x64

    # fully connected
    self.flatten_size = 64 * 64 * 64  #
        channels * height * width after
        pooling
    self.fc1 = nn.Linear(self.flatten_size,
         128)  # flatten to 128 neurons
    self.fc2 = nn.Linear(128, num_classes)
         # output to 'num_classes'

def forward(self, x):
    x = F.relu(self.conv1(x))  # conv +relu
    x = self.pool(F.relu(self.conv2(x)))  #
         conv relu pool
    x = x.view(x.size(0), -1)  # flatten
        tensor
    x = F.relu(self.fc1(x))  # fully
        connected relu
    x = self.fc2(x)  # output
    return x
```

4.1. Technical Discussion

Using the front end app, we found that our model had low accuracy on images with low color contrast – i.e. a light skin tone in front of a light-colored wall. This may be because many of the training images had high levels of contrast. Given more resources, we would incorporate training data including more skin tones and background types to eliminate this training bias.

At the moment our front end implementation only allows the user to use one of the CNN models at a time, if one wants to compare results they may modify the loader to instead load in the weights for the preferred model, or simply run the last cells of the jupyter notebook where both testing nodes for a single file path are situated.

Generally, our two CNN models did not perform that well outside of the given training data. This is for two reasons, one we could not use that much training and testing data due to github issues and corrupted images from large datasets. Secondly, the dataset itself consists of images of the same person in the same room with no other background meaning that it is difficult for this model to work on new images. More data of different types would help with this.

To run our code, due to the issues with GitHub and the size of our data, one should download the pretrained weights from the models in GitHub and download the data from this Google Drive link: https://drive.google.com/drive/folders/1Ye_Az1KMMXi9NrHQp25vgNUyr-K7Y-RZ?usp=drive_link.

Alternatively, you could use any data and dataset you prefer, but make sure to replace both the data and dataset folders with either this downloaded data or the data of your choosing.

You can also watch a demo of our program here: https://drive.google.com/file/d/1yn_1dTN4pfM59jGQ23mw0tLCFhJHFPQd/view?usp=sharing

5. Conclusion

Our models for American Sign Language (ASL) recognition can provide a tool to assist ASL speakers in communicating more easily with people who do not know the language, improving accessibility. It can also be used as a helpful learning tool for those who want to learn ASL.

Additionally, by comparing SIFT-based methods with neural networks, one sees insights in the strengths and limitations of these approaches. While conventional methods like SIFT are more interpretable and lightweight, neural networks offer better performance, especially in handling complex patterns and diverse hand shapes. This work paves the way for more accurate and scalable ASL recognition systems, with a significant impact on accessibility for ASL users in various applications.

References

[1] Bao Anh. Asl alphabet. https://www.kaggle.com/datasets/baoanhcr7/asl-alphabet/, 2021. 1

[2] Streamlit Creators. Streamlit. https://streamlit.io/, 2024. Used for frontend and integration. 1

Appendix

Team contributions

Jinho Lee Worked on implementing the SVM model and the bag of words feature collector. Tested different kernels and parameters to improve performance on test and training dataset. Helped design and build front end.

Natalie King Pulled and cleaned data from Kaggle. Worked with Jinho on the bag of words feature collector and the SVM model. Helped test different kernels to improve the classifier. Worked with Jinho on building frontend and integrated real time model predictions from the SVM model/backend.

Sylvie Watts Worked with Domingo on building the two CNN models — the pretrained and custom models. For this model I also ran the epochs and tested to build them up. Additionally, I cleaned up the data and organized it for all the models to use.

Domingo Viesca In charge of building the CNN models in pytorch. Set up all of the different classes and function structures. Built the jupyter notebook to more quickly develop and debug each step. Worked in parallel with Sylvie and tried different development strategies for this model. In charge of linking up these back end processes with the front end so the end user could upload an input to the two CNN models.