ASL Interpreter

Author: Malcolm Fitts, Aaron Selesi, Young Woo Kim: Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—A system capable of real time translation of American Sign Language to English using machine learning, computer vision and web application development**

***Index Terms*—Deep Learning, Neural Network, Image Processing, Feature Extraction, Edge Detection, Image Smoothing, Background Subtraction**

I. Introduction

This project is motivated by the need for communication between hearing and hearing-disabled individuals. Its goal is to provide a consistent, easy-to-use program that facilitates this communication in real-time. In order to do so, we have defined a 29 word lexicon of recognizable signs: the letters A-Z, space, delete and an empty sign. Many of the competing technologies in this area are hardware and software based, requiring the use of glove or wearable technology to increase accuracy of reading. On the other hand, our approach avoids additional hardware requirements since we aimed for our final product to be more widely accessible to users. Instead, we use raw camera frames to extract the hand and the important features to use for recognition. This way, there is no extra technology required other than a device with a camera that can access a browser.

Our application is made to classify static hand signs that represent letters. Static signs are stationary signs that only require a single frame to capture and consist of most letters and numbers as well as a few words. By capturing single letters along with spaces and deletes, we allow the users to build more complex phrases with significantly less knowledge of the ASL language as a whole. As an initial goal, we intended for our app to correctly classify static signs with at least 90% accuracy. Furthermore, recognition and feedback should aim to happen within 1 second of the end of a gesture. The detection should also be scale and translation invariant so that signs made from 1 to 3 feet of the camera should be recognized so as to allow user flexibility. Additionally, users must be signing in a bright enough area so that the images can be properly detected.

II. Design Requirements

The key technical challenges include successfully recognizing the ASL hand signs as words, in relatively noiseless environments that are consistent in lighting, angle, and interfering objects in the background. Since a language includes many words and phrases, we have decided to limit the capacity of our project to recognize the letters and a subset of the words in ASL so that we can train the program within a reasonable time frame. Specifically, we planned to recognize 29 unique static signs that are required for basic written communication. Since we want the app to recognize signals within a reasonable real time constraint, we would not want to generate an output with too many classes because an unrealistically high number of output classes could introduce more similar signs that could be confused together and increase the size of the neural network, which could slows down response time such that it longer operates within real time requirements.

In order to meet these timing constraints, we implemented image processing and feature extraction algorithms so that our neural network can

train and operate on video data in a reasonable time frame. The methods of image processing deal with edge detection, image smoothing, frame subtraction and background subtraction.

There are many words in ASL that are motions as opposed to single frame pictures, also known as dynamic signs, which presented challenges in both feature representation and building the predictor model. Initially, we planned on creating multi-frame videos that would define our input space with dynamic sign labels that define the output class. After testing and experimenting with this kind of design, we found that especially in the dynamic sign classification problem, we would need more reliable feature extraction in order for this kind of endeavor to be realizable. So one example of the feature extraction methods we attempted was extracting the locations of the joints in the fingers and wrists, so that we could defer the work of learning important features of the input image to another convolutional neural network that operates independently of the classification stage. This would also allow the input vector to be sufficiently small so that we could train a large network fairly efficiently. In fact, without this feature engineering scheme, training on large image files would have been very difficult. Ideally, we could use this feature extraction across the frames of a video feed so that we could analyze each frame and train the neural net based on those features.

Accuracy testing involved reserving a subset of the training data to use as the validation set,, upon which the accuracy is evaluated each epoch. Cross validation methods, such as varying the subset of the training data to be used as the validation set, was useful in tuning the hyperparameters of our network to optimize both accuracy and computation speed, which are the measurable qualities of the classification stage as conflicting sides of the accuracy-speed tradeoff. Test data came from both untrained samples of the existing data set and manual testing by us directly to test compatibility with varied distances, light settings, hand shapes and colors, and backgrounds to ensure the generalizability of the network. We planned to use binary correctness classification as well as Top K classification, in which correctness equates to the correct label belonging in the set of the K most likely outputs. However, it turned out that the binary correct versus incorrect metric was sufficient enough for measuring the network since the performance exceeded our expectations.

III. Architecture and/or Principle of Operation

A. overview

Our system architecture can best be broken down into two stages. The actual training occurs in the latter of the two, in the classification stage, where the neural network is fed data in order to learn. The datasets we used were composed of 3000 instances of each of the 29 hand signs we planned on recognizing. Since the gathering of the data manually would have taken too much effort and time given the scope of the project and how much time we had to budget, we primarily relied on publicly available datasets such as the Kaggle ASL Alphabet dataset built by Akash Nagaraj (6). By using these datasets to train the dataset, we could focus our effort on the algorithmic portion of the endeavor. In the first stage of the workflow, this data is sent to the web-app for preprocessing and feature engineering of a representation that is suitable for good training. Since these datasets consist of all single frame images, we can perform a universal preprocessing sequence on the images so that the data we feed into the neural net does not pick up irrelevant noise from the background of the dataset or suffer from a lack of generalizability. This same sequence was run on a statically defined portion of the video frame that is marked as the signing region at test time in addition to background subtraction to clean the bounding box of noise. Therefore, we would be able to identify where the

sign exists relative to the global image without using too many resources or imposing too uncomfortable a restriction on the user.

B. Image processing

The actual process of image processing is a combination of edge detection and background subtraction. The edge detection was implemented through the use of Gaussian blurs and Canny edge detection, which both blur the image to reduce noise and detect harsh changes in the frame as the edges. From there we ran a background subtraction algorithm that calculates the average of frames to determine a static background from a moving foreground. The algorithm then subtracts out the background to create a mask of the moving image, or in our case the hand sign being displayed.

For the specifics of the Gaussian blur, we would convert the RGB (red, green, blue) input image to GRY (grayscale) which changes the three channel image to an image described by a single weight of each pixel. The weight ranges of this new GRY image were between 0 and 255, and represented how light or dark the image appears at the pixel. From there, we fed the GRY image into the Gaussian blur to smooth the edges of the image. After much experimentation, we used a kernel size of 7x7 pixels with a standard deviation in the x-axis of 2. This means that every pixel is averaged with the 7x7 block around it in order to determine the average value of the image around this pixel and adjust the pixel accordingly.

$$B(x,y,t) = \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i)$$
$$\Downarrow$$
$$|I(x,y,t) - \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i)| > Th$$

Figure: Background Subtraction

After the image has been blurred, we feed the result into a canny edge detection algorithm with upper and lower bounds of 50 for the hysteresis procedure as well as aperture size of 3. We decided on an equal upper and lower bound to remove the hysteresis procedure, since the image blurring helps remove the need for it. The red boundary line in the figure below indicates that all pixels with an intensity gradient less than minVal are surely not considered as edges while the blue boundary line indicates that all pixels with an intensity gradient above maxVal are guaranteed to be edges. If the gradient falls between threshold lines, the detector considers adjacent pixels and connectivity in a more sophisticated classification procedure.

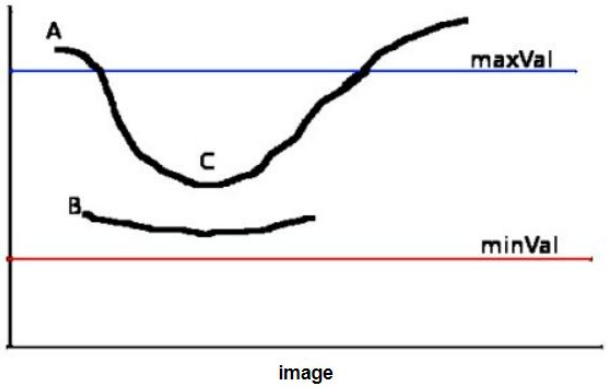


Figure: Canny Edge Detection Thresholds

The procedure also affected our image processing efficiency so by setting this empty range we were able to reach faster analysts speeds. The edge detection algorithm also transforms the image into a binary representation of the original, where pixels are either black or white. The white pixels represent the areas in which there are harsh changes to the frame, which can be construed to be edges. As can be seen in Figure 3.3, this means that objects in the background, such as a picture frame, will be detected as edges and represented in the final product.

Figure 3.3: Example of Edge Detection

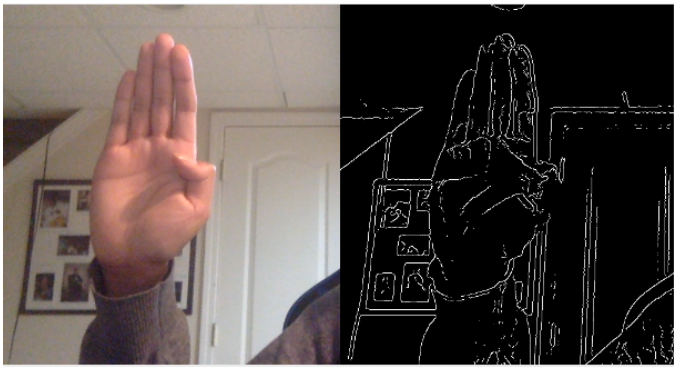
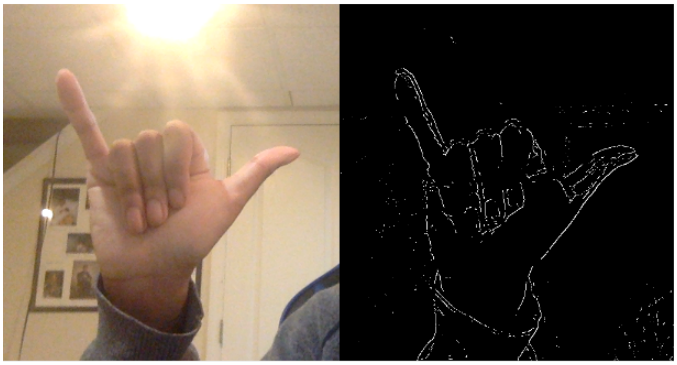


Figure: Canny Edge Detection without Background Subtraction

By running this process on all of the images both during training and in use, we achieve multiple different advantages for our system in terms of execution speed and overall correctness. Firstly, background subtraction accomplished generalizing the background of the inputs to the network. The Kaggle ASL dataset is produced by one signer under various light conditions with the same background. This dataset on its own without preprocessing lacks variety and comprehensiveness. When we kept the background in the test inputs, we observed much lower success rates during evaluation. This can be explained by the fact that the neural network has never been exposed to signs where the background has never been seen before. While it is true that the network is able to identify the features that distinguish the classes in the training dataset, background subtraction essentially equalizes the backgrounds of both training and test inputs to an empty background. Since our dataset consisted of images with only a single background, training on the raw train images would have caused the network to become over-familiar with that background, and be unable to generalize to all backgrounds. Thus, assuming that the background subtraction does not accidentally delete edges of the hand it is strictly better to include it for higher accuracy gains, as the features that are irrelevant to the classification could be filtered out.

The edge detection serves to further filter the noise that is unimportant, such as the skin color of the signer and the lighting conditions. Knowing that the only features needed to identify a hand sign is the shape of the hand itself, we decided that using edges to show the shape of the hand would be able to unify images of the same sign where variables such as skin color and lighting may vary. As a welcome side effect, this technique also allows us reduce the dimensionality of the image since instead of triples with values in the range [0, 255], each pixel would simply be a single binary scalar that represents whether there is an edge at this pixel. When we compare the results of our edge detection with the combined background subtraction, we could resolve an image that contains a more reduced background without losing much of the hand's image. An example can be seen in Figure 3.4. From this example, using the same background as Figure 3.3, we can see that the background picture frame has been removed from the image without losing the shape or definition of the hand itself.

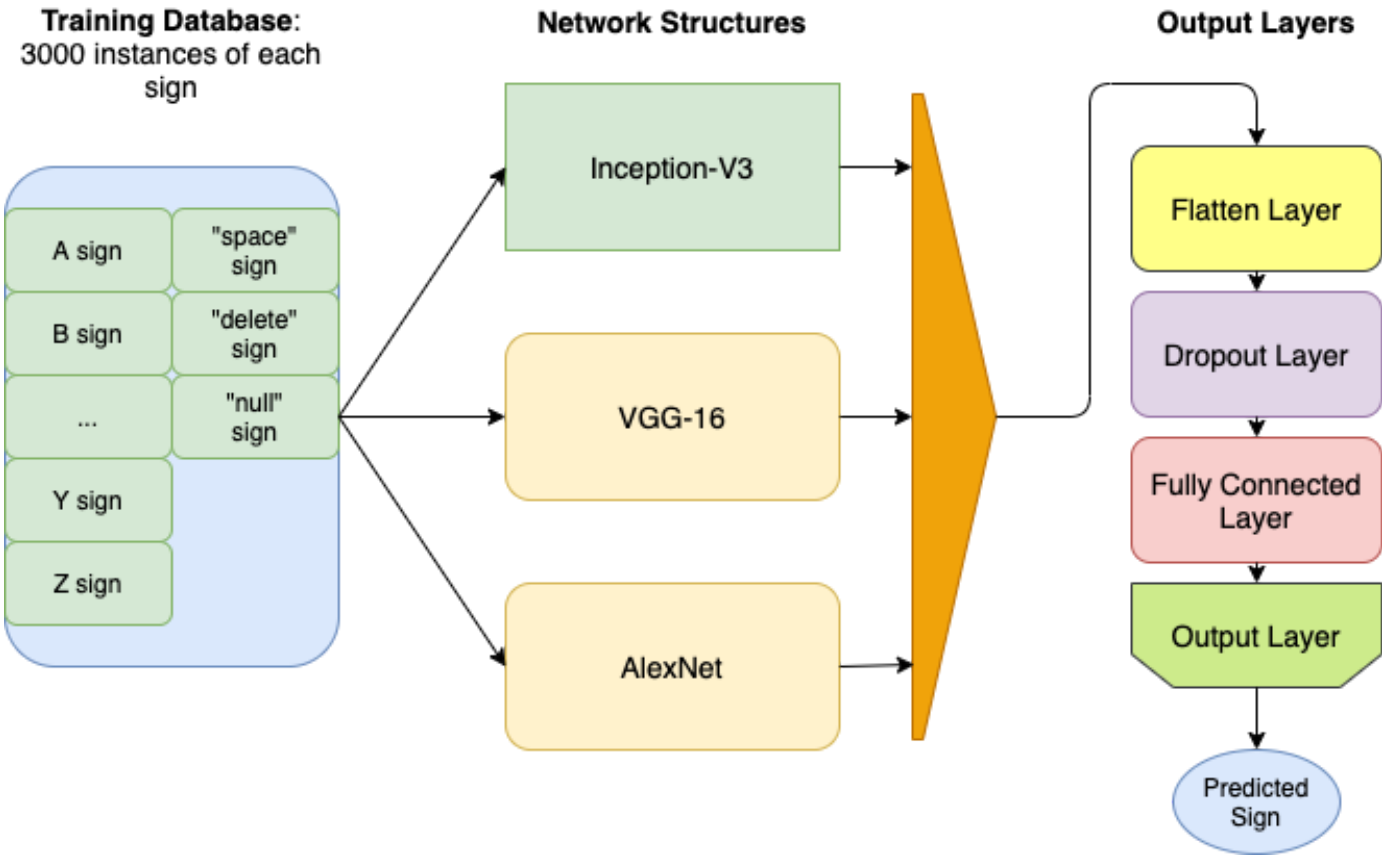
Figure 3.4: Example of Edge Detection And Background Subtraction



C. CLASSIFICATION

The final neural network model was built off of the Inception-V3 network architecture, Google's third edition of the Inception Convolutional Neural Network series. We chose this network structure since it is well known for its state of the art accomplishments in object recognition tasks, demonstrated by its results in the ImageNet Recognition Challenge, which extends well to our intended use. As a 42-layer deep network with a relatively small 23 million

Figure 3.1: Training Data-Flow



parameters, the InceptionV3 is able to achieve classification results similar to that of other networks such as VGG16 and AlexNet which have 180 million and 60 million parameters respectively. With special optimizing features such as an auxiliary classifier, smaller factorized convolutions, and batch normalization layers, and label smoothing, the InceptionV3 model is optimal for situations where limited training time is available and predictions must be made within timing constraints. By using this pre-structured network, we were able to then add our own layers after the network in order to further promote learning. Specifically, we flattened the output of the InceptionV3 base model, added a dropout with 0.5 probability to reduce overfitting, added a fully connected layer in between the Inception-V3 network and the output layer, and generated probabilities over 29 classes with softmax activation for categorical outputs.

As we decided to adapt the InceptionV3 network, we also tested the same modification layers at the end to the VGG-16 and AlexNet architectures since those have also had successful applications in image recognition. After initial testing all of these structures and comparing both accuracy and efficiency, we found that the Inception-V3 network generated the highest validation accuracy in a moderate and manageable amount of time. In short, these constraint factors specific to our problem advised how we chose to design our system architectures.

D. Web Application

Building a native android app for our project was one option we considered but we decided to build a web app for our ASL interpreter as it would be more accessible from multiple devices than native apps.

We chose to build this app using the Django

framework because of the ease of use of python and the OpenCV api that is available for it. It also handles both the frontend and backend and reduces work. The app has a simple layout, showing a live feed of the user signing and a highlighted area where signs are interpreted in real time.

In figure 3.2, we show a high-level design of the data flow when a user is using the system. In order to send the video feed to the Django web-application, we used an IP Webcam object. The IP Webcam streams the video feed to an IP on the local network that is then picked up by the Django web-app and analyzed by the image processor. The result of image processing is then fed into the neural network classifier, giving a prediction of the displayed sign.

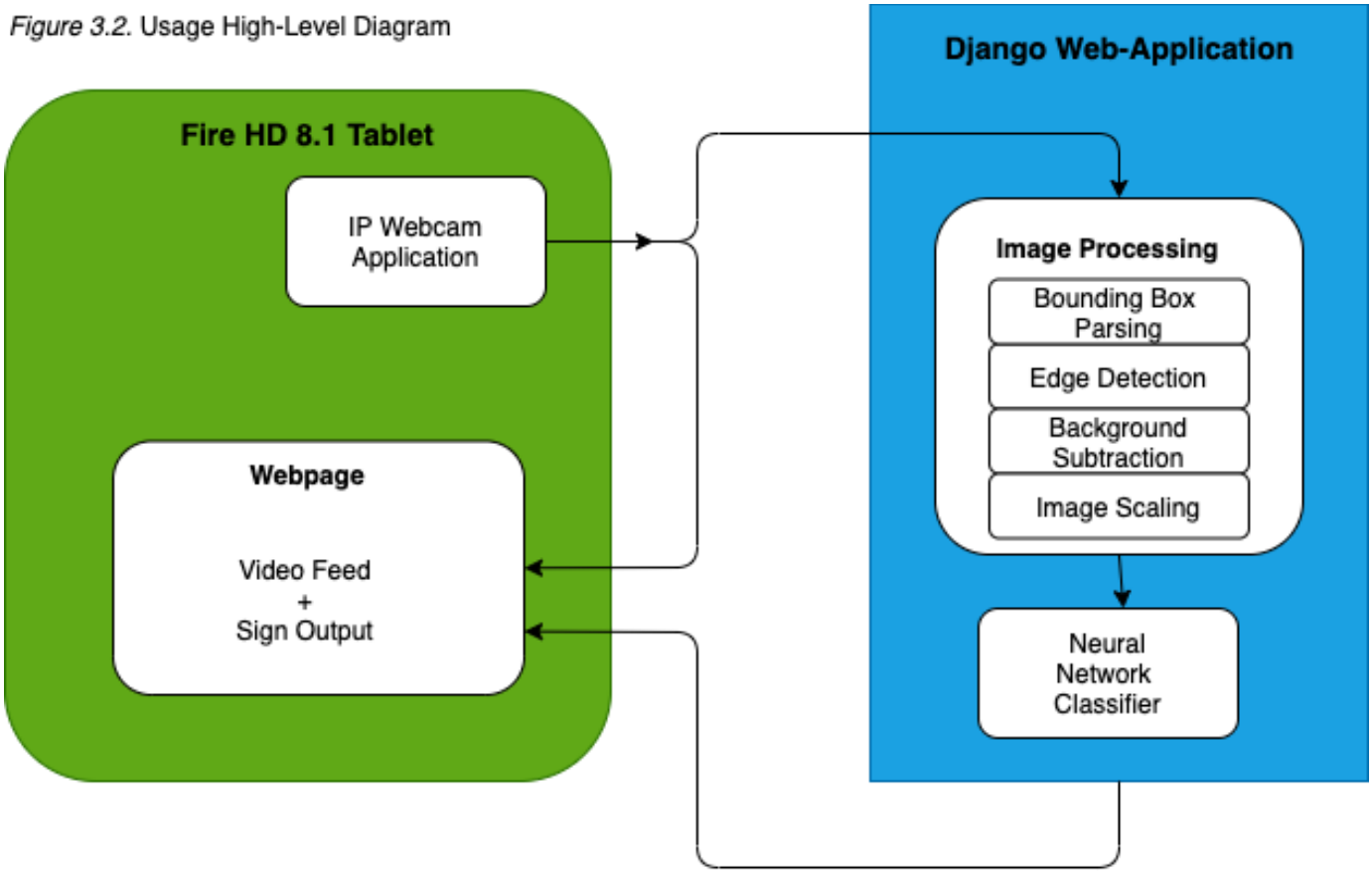
IV. Design Trade Studies

As the architectural choices to be made in our project are entirely software, they can be reconfigured and redesigned with much more ease and free of cost. This means we have more liberty to consider other design choices and evaluate them during the realization and production process and shift gears if we find that another approach is more optimal than the current one. The only substantial cost of making design chances would be time since training convolutional neural networks take time. Thus, we allocated a safely large portion of the budget towards AWS credits for GPU access to train networks with different architectures quickly and in parallel.

Initially, we had planned to develop a system that would be able to classify static signs along with dynamic signs, which are signs that require movement as opposed to a stationary picture. In terms of the model input, this requires multiple frames concatenated into a three dimensional array, as deep as the number of frames captured. This substantially increased the size of the input dimension, even after sampling at a lower framerate at the cost of accuracy.

Prior attempts at a dynamic sign interpreter

Figure 3.2. Usage High-Level Diagram



have generally been used for contexts such as gesture recognition for smart home controllers, or translation without real time requirements. Generally, special hardware has been involved in the form of gloves that send signals to the system in order to detect lower dimensional features such as fingertips and joints, filtered and refined. Being able to condense the 3-D vector into a 2-D vector seemed to be a common theme among these preprocessing stages, so we strived to create something similar.

Figure: Joint Detector Performing Correctly on Clear Image

Our idea for a hardware free feature extraction scheme involved extracting the joints of the hand on the screen, but there were a considerate number of problems we encountered with this approach. Firstly, the very time it took to preprocess the individual frames to make the network operate faster took up more response time than expected. Secondly, the joint detector was not very robust under settings where the background interfered. Lighting was a very critical and sensitive factor and often made the detection of the contours of the hand inconsistent. Furthermore, dynamic signs could take a lot of physical space so there can be no bounding box where the user is directed to sign within. This in itself is not problematic since the joint detection is able to identify the hand, but when another hand or the user's face showed up in the frame, which was usually the case, the detector would label joints elsewhere other than on the correct hand. We have tried using a hand detector that automatically draws contours around where it detects the hand to be in the frame, but this resulted in similar problems where either parts of the hand were cut out of the bounding region or other background features were picked up.

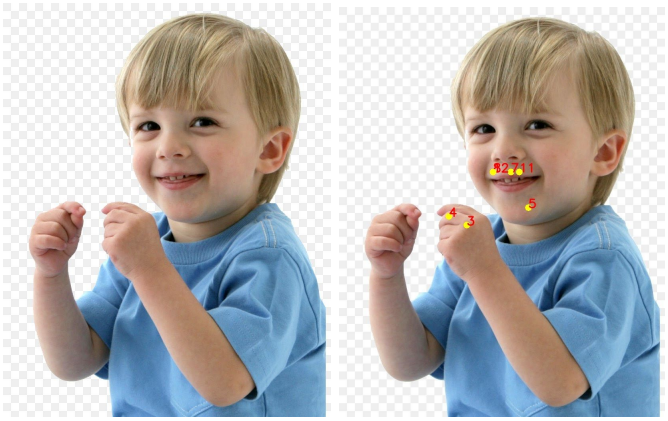


Figure: Joint Detector Performing Poorly on Complex Image without Bounding Region

There are a class of neural networks which involve CNNs optimized for 3d inputs such as the C3D and deep LSTMs, but these networks trained on video inputs have reported to take days to train on such large datasets, using high computational power resources. Of course, one training run is manageable but no design of a ConvNet works on the first try. After multiple runs of different training parameters and architectural modifications and

finding that certain feature extraction schemes do not work, those days easily add up to weeks of sheer training. While transfer learning and fine tuning could be used to improve the initial weights of the network and immensely speed up training, we were only able to find pretrained models for our 2D ConvNet models, such as the InceptionV3. Moreover, the Kaggle ASL dataset for static signs barely fit in the GPU instance that we were able to obtain from AWS. To fit a dataset of videos in the instance would have required trimming down the size of the dataset. Along with storage space, memory limitations in the AWS instance also turned out to be a recurrent issue. Due to the size of the weights and the size of the dataset, we were forced to use only 10% of the Kaggle ASL dataset, but thankfully this turned out to be sufficient for static signs. For a larger 3DCNN with video inputs, the memory usage would have skyrocketed, and we would have had to use a rather small dataset. To resolve this, we would require a much faster GPU instance with large storage space and memory but unfortunately AWS did not grant us access to a larger instance than our current m5.xlarge instance that we received upon request. So rather than to suffer losses due to data loss, we opted to aim to thoroughly explore and optimize our static sign detection system instead.

Regarding the preprocessing steps, another feature engineering plan we had was to use a binary pixel encoding. In more detail, this involves taking the edge filtered grayscale image and converting each pixel to either black or white by splitting on a brightness threshold. This would allow us to use integers, which capture 2^32 different values, to represent sets of 32 pixels, as opposed to one pixel in the grayscale image. Effectively, this reduces our input dimension by a factor of 32 and is easy to program, but performed poorly in an image classification setting. So while this did lend efficiency benefits, the accuracy losses were not worth the tradeoff.

The even more reductive feature encoding was the joint detection classifier which reduces every image to 22 integers. The disadvantage to this approach is that nuances in grayscale image would be lost. Additionally since edge detection will not perfectly reproduce the outline of the palm and the fingers, simplifying to black and white may cause the outline to fragment where the edge value is not as intense.

A final and obvious choice is removing the preprocessing in the static signs all together, and using the raw input to the neural network. While this is the most expensive computationally for the classification stage, the preprocessing stage is significantly reduced, so it is not unreasonable to think that the overall runtime cost may even out . However, after testing this, we realized that it would not achieve sufficient accuracy or efficiency, so we quickly forgoed this idea.

V. System Description

Our ASL Interpreter is trained on a data set of American sign language gestures using deep learning. Using an Amazon Fire Tablet as the platform for image data capture, OpenCV and edge detection and feature extraction algorithms for image data processing, the program will be run through a Django web app utilising the OpenCV api. The user interface is built into a webpage, and contains a live video of the user with a box designated as the “signing area”. The page also has a description of the recognizable signs and an area in which the read signs will be printed. Diagrams of the final design are available on Figure 3.1 and 3.2 in Section III.

A. Neural Network

We designed the neural network based on available data and our ability to parse it. We processed the training data so the layer size is reasonable and so the network can be trained

efficiently. Designing and tuning the neural network comes down to how the feature extraction works. If we were to just read in raw data from the normalizer, the input space would be 1280 * 720. Instead, by capturing a smaller subset of the video frame we can reduce input layer size and focus more on layering. Since this is the part that we can test most rigorously, we took a lot of time on testing, re-designing and re-tuning our layers.

Our neural network was implemented using the Tensorflow and Keras packages. The input is the preprocessed version of the picture frame from a video feed captured by the tablet camera. The output is one of the 29 signs in the alphabet, space, delete, or nothing, which either type a letter, add an underscore, delete the last letter, or do nothing, respectively. Our solution approach for the implementation approach is as follows. A InceptionV3 base model is followed by a Flatten layer, a Dropout layer with probability 0.5, a fully connected layer with 29 output units followed by a softmax activation function for multinomial classification:

$$y = argmax_i softmax(x)_i = argmax_i \frac{exp(x_i)}{\sum_j exp(x_j))}$$

$$(i \in \{1, 2, \ldots, 29\})$$

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure: InceptionV3 base model

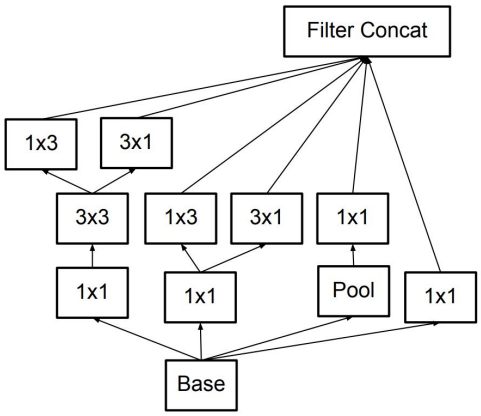


Figure: Inception module

where y is the output of the neural network and 29 = size of output space or how many signs we are detecting. Neural networks seek to minimize a loss function and ours will be categorical cross entropy. We used an Adam optimizer, which is a learning optimization algorithm that combines AdaGrad (separate learning rate per weight parameter) and RMSProp (changing learning rate per iteration) to speed up training performance and update neural network weights. The best learning rate we found for training the classifier is 0.0003, which we found most successful with 10 training epochs and a batch size of 32 on a trimmed Kaggle ASL dataset of about 10,000 images.

B. Computer Vision

How we handle the computer vision processing is as crucial to our project as the network is in producing consistent and accurate results with decent response time. Our solution approach to the computer vision portion of the project includes feature extraction and image processing. For feature extraction, we are trying to pick out the important bits of data from the overall image of a gesture to condense and input to our neural network.

For Image processing solution we are using our Fire Tablets' cameras to capture the image data and OpenCV to process the data. Through OpenCV we performed Background Subtraction by Moving

Averages which is where average values across frames are used to determine the moving foreground against the stationary background. Then, we Gaussian blur to smooth the image of minor blotches of colors that could be misconstrued as edges, and Canny edge detection to pick out the edges. Furthermore, we implemented a face detection and mask in the web app that is able to remove the face from the image, since background subtraction was not enough to remove the face. This feature enabled us to erase just the face from the image and would have allowed us to potentially use a dynamic bounding box detector to find a hand, but took much too long to operate on a single frame. Thus, we disabled the face removal and used a static bounding box, since this would not affect the user's usability for signing static hand signs.

While we originally planned to sample at a rate of 5 seconds, we realized throughout manual testing that static images would not need such a high sample rate. Furthermore, this simply would cause the user to mistype multiple characters at a time if they realized a letter was signed incorrectly too late. Thus, we made the delay happen . Due to the modularity of the OpenCV camera feed running with the detection in the background, we are able to give the illusion that the two components are running in parallel, without a pause in the frontend.

C. Web Application

Initially we had considered building a native android app for our project but soon came to the realization that this would be too large of an endeavor on top of the main portions of the project due to our lack of experience with android development Building a native app would also go against our goal of making the app easily accessible unless we have the expertise to develop and deploy it on most platforms. So we decided to build a web app for our ASL interpreter with the added advantage of it being more easily accessible from multiple devices than native apps.

We chose to build this app using the Django framework because of the ease of use of Python and the OpenCV api that is available for it. It handles both the frontend and backend and reduces work so you don't have to closely manage HTML, CSS, and JavaScript files while still having the option to do so if needed. The app has a simple layout, showing a live feed of the user signing, a highlighted bounding box area where the signs will be interpreted, and subtitle text at the bottom of the screen filled by the text of translated signs. The app works by displaying the live feed as a video stream from our fire tablet using an IP webcam. Each frame is preprocessed using background subtraction, gaussian blur and canny edge detection in order to reduce the neural network input size. Then a frame with a hand present in a static bounding box is sampled and fed to the neural network.

D. Physical Components

Due to our project being heavily software and signals driven we don't have a need for much hardware. However we wanted to show that the project could run on an easily accessed device and thus we decided on Amazon's Fire Tablets. They are affordable and lightweight and thus, perfect for our use case. We envisioned this application to be highly portable, so that users may run the system on their tablet or other portable device, which they attach to a wall in a room where they can periodically go to sign.

V. Project Management

A. Schedule

Our current full schedule can be found at the end of the document. In general, each team member had a task or a number of tasks to complete every week. At the start of the project we spent a lot of time learning, and since then we've followed a cycle of research and learning, and then implementation, and

then into integration. In the end, we integrated our individual portions and debugged them in time to finish our project.

B. Team Member Responsibilities

In the conception phase of our project, we maintained generalist roles as we were doing a lot of research and learning in the topic areas that our project covered. Once we had a general grasp of the concepts required, we started splitting tasks according to what areas we were most proficient at while still maintaining general tasks when needed.

Young had the most experience with machine learning and deep learning, so he took the lead on developing, training, and optimizing the neural network for recognizing ASL gestures. His secondary responsibility was to work on the feature extraction component of computer vision for the project. Aaron was taking on the role of developing the Django web app through which our ASL Interpreter will communicate with our neural network and his secondary responsibility was to lead the integration of the OpenCV api into the web app for computer vision. He also worked in tandem with Malcolm on computer vision.

Malcolm took on computer vision, working on OpenCV to enable the filtering of the noise out of videos for easier feature extraction and as his secondary responsibility, also worked on Amazon Web Services integration for the training of our neural network and hosting of our web app.

Oftentimes, we ended up working in pairs or in a team when tasks proved to be too big for one person to handle. Our main goal here was to maintain flexibility so no one person was overwhelmed with work.

C. Budget

Due to the nature of our project we did not have to purchase much hardware. We purchased three Fire 8 HD Plus tablets for each team member and $150 of AWS credits to use the AWS instances. The complete Bill of Materials and their associated costs is included at the end of the report.

D. Risk Management

Our primary risk management strategy dealt with research and experimentation after making implementation decisions. Since we were not buying many parts for this project and the main focus is on development, we were allowed the freedom to try as many approaches as we liked with our only limiting factor being time. So when assessing risk, we focused on the issues around design choices and our past experiences with various systems and technologies. We went about balancing the risks we took on in developing this project by weighing the benefits of learning new technologies with the speed and comfort of using technology we had experience with.

For example, one of the choices we made dealt with the using physical gloves versus writing an implementation plan that pulled sign information straight from a video feed. We looked at possibilities of using a physical glove with sensors as well as a glove with color targets on fingertips in order to provide extra parseable information for the neural network to learn. The other side of this choice was creating a system to fully parse the hand sign information from the video feed. This would require much more clever design and implementation choices, but ultimately also lead to a more challenging problem and worthwhile learning experience.

At the same time, this choice also helps our learning process, since the databases we had available were just raw image data. If we were to have added physical gloves, we would have to heavily augment our training data or completely recreate data. This is because we would need to train on the physical sensor data as well as the video data, and they would need to be synced. Therefore, it was more worthwhile to choose the approach that

made our problem more difficult without causing us to recreate all of the data ourselves.

VI. Related Work

We were really only able to find one product that was the same as the product that we are trying to produce. MotionSavvy, which emerged from the Leap Motion accelerator AXLR8R, built a tablet case that leveraged the power of the Leap Motion controller in order to translate American Sign Language into English and vice versa. This product was built entirely by a 6-person team of which all were deaf.

VII. Special Thanks to AWS

The making of this project would not have been possible without the help of the AWS m5.4xlarge instance, which significantly improved computation speed, storage space, and memory capacity compared to our local machines. It enabled us to train and retrain our models in under an hour, which allowed us to truly optimize them with time to spare. We used a total of around $42 worth of credits, mostly through the m5.4xlarge instance.

VIII. Summary

In summary, our system was able to meet our design specifications. It hits a limit of recognizing 29 signs at an accuracy of 95% for static signs. Possible next steps we could take include further attempts at a dynamic bounding box as well as recognizing dynamic signs.

A. Future work

Currently our project only implements a mix of 29 American Sign Language static signs, and that is what we planned to have produced at the end of this semester. However, thinking about the future of our project, it would be interesting to add more gestures and grammar to make it more useful in daily life.

B. Lessons Learned

We would say to other student groups that are attempting similar neural network based problems that it is important to start exploring data representation early into the project. Being able to abstract relevant information in a problem is important since it lets you more easily develop a neural network architecture as well as train the network. Designing the neural network itself is also likely going to fail the first time and keeping this assumption in mind is a good idea For training based projects, being able to train multiple implementations in parallel via AWS instances has been very time saving. Setting up AWS is also nontrivial and sometimes clunky if you have to transfer data in and out a lot, but ultimately it was our only option with our limited computational resources Otherwise we just recommend getting started early!

X. References

1. Li, Dongxu, et al. “Word-Level Deep Sign Language Recognition from Video: A New Large-Scale Dataset and Methods Comparison.” *ArXiv.org*, 21 Jan. 2020, arxiv.org/abs/1910.11006.

2. “ASLLRP DAI.” *DAI - ASLLVD*, www.bu.edu/asllrp/av/dai-asllvd.html.

3. 10-601 Panopto Recordings (Machine Learning), https://scs.hosted.panopto.com/Panopto/Pages/Sessions/List.aspx#folderID=%22eb329349-cb74-47a2-8e0d-abc000e95971%22

4. Akash. “ASL Alphabet.” *Kaggle*, 22 Apr. 2018, www.kaggle.com/grassknoted/asl-alphabet?select=asl_alphabet_train.

5. Akash. “ASL Alphabet.” *Kaggle*, 22 Apr. 2018, www.kaggle.com/grassknoted/asl-alphabet?select=asl_alphabet_train.

6. Gupta, Vikas. “Hand Keypoint Detection Using Deep Learning and OpenCV.” *Learn OpenCV*, 8 Oct. 2018, www.learnopencv.com/hand-keypoint-detection-using-deep-learning-and-opencv/.

Capstone Project Schedule

Item	Cost	Description	Status
Fire 8 HD Plus Tablet	$89.99	Purchased on Amazon	Arrived
Fire 8 HD Plus Tablet	$89.99	Purchased on Amazon	Arrived
Fire 8 HD Plus Tablet	$89.99	Purchased on Amazon	Arrived
Amazon Web Service Credits	$150	Purchased on AWS	Redeemed
Total	$419.97		

- Malcolm
- Aaron
- Young
- Malcolm + Aaron
- Malcolm + Young
- Young + Aaron
- Everyone :)