# Real-time American Sign Language Detection System

Bill Yu
billyusfg@gmail.com

Hollis Holmes
hollis.holmes@ucalgary.ca

Kevin Amado
kevin.amadorueda@ucalgary.ca

Tyson Trail
tyson.trail@ucalgary.ca

Vladyslav Timofyeyev
vladyslav.timofyeyev@ucalgary.ca

***Abstract*—This paper presents an approach to American Sign Language (ASL) recognition using deep learning and transfer learning techniques. We trained a neural network on a large data-set of American Sign Language images, leveraging the pre-trained weights from a state-of-the-art image classification model. The resulting model achieved high accuracy in classifying various sign language gestures. We then deployed this model in a real-time sign language detection system using a camera feed. The system successfully recognized sign language gestures in real-time with low latency, and achieved high accuracy in identifying signs from multiple individuals. This approach has significant potential for enhancing accessibility for people who use sign language as their primary mode of communication.**

***Index Terms*—Deep Learning, Transfer Learning, American Sign Language**

## I. Code repository

The source code for this project can be found at:

https://github.com/kamadorueda/2023-winter-enel-645-final-project

## II. Introduction

Sign language is a critical mode of communication for individuals who are deaf or hard of hearing. However, interpreting sign language can be challenging for those who do not have proficiency in it. Therefore, there is a growing need for automated sign language recognition systems that can translate sign language into text or speech, thereby facilitating communication for them. In recent years, deep learning models have shown remarkable success in image classification tasks and have been used to recognize various sign language gestures. However, the large amount of labeled data required to train these models can be a significant barrier. Transfer learning is a technique that leverages the pre-trained weights of a neural network and can mitigate this problem by allowing for the transfer of knowledge from a pre-trained model to a new one. In this paper, we present a novel approach to sign language recognition using transfer learning and deep learning techniques. We trained a neural network on a large dataset of sign language images, and utilized a pre-trained state-of-the-art image classification model to accelerate the training process. We then deployed the resulting model in a real-time sign language detection system using a camera feed. This system was able to accurately recognize sign language gestures from multiple individuals in real-time with low latency. Our approach has the potential to significantly enhance accessibility for individuals who use sign language as their primary mode of communication, and could also have applications in other domains, such as robotics and human-computer interaction.

## III. Related work

There are several researchers and organizations that have worked on using transfer learning and deep learning to train a neural network for sign language recognition, as well as implementing it for real-time detection. A notatable example of this is a research paper by K. Bantupalli and Y. Xie [1]. They proposed a system for American Sign Language (ASL) recognition using deep learning and computer vision techniques. The system was presented at the 2018 IEEE International Conference on Big Data (Big Data) in Seattle, WA, USA. The paper describes the use of a convolutional neural network to extract features from sign language images and classify them using a softmax classifier. The proposed system achieved a high recognition accuracy of 97.8% on an ASL dataset.

In this study,The paper proposes a vision-based application to aid communication between signers and non-signers of sign language. The authors propose an ensemble of two models that can recognize gestures in sign language. The dataset used is one of the American Sign Language, which consists of 100 different signs performed five times by a single signer in varying lighting conditions and speed of signing. For the purpose of consistency, the background in all the videos is the same. The proposed model uses a CNN model named Inception to extract spatial features from the video stream for Sign Language Recognition (SLR). Then, by using a LSTM model, the RNN extracts temporal features from the video sequences via two methods: Using the outputs from the Softmax and the Pool layer of the CNN respectively.

Another example is the Sign Language Recognition system developed by Ali Farhadi, David Forsyth, and Ryan White in 2015 [2]. To evaluate their proposed method, the authors used the American Sign Language Lexicon Video Dataset, which contains 1,000 signs from the American Sign Language lexicon, each performed by 20 signers, for a total of 20,000 sign instances. The authors used the first 900 signs for training and the remaining 100 signs for testing. They also evaluated their method on a set of 200 signs not included in the training data.

For their method, the authors used pre-trained CNN models, which were trained on a large-scale dataset for image recognition. They then fine-tuned the pre-trained models on the sign language dataset by replacing the last fully connected layer with a new layer with the number of outputs corresponding to the number of sign classes in the dataset. The authors used the fine-tuned models to extract features from the sign language videos, which were then fed into a support vector machine (SVM) classifier to predict the sign.

The authors compared their transfer learning approach with traditional approaches that involve training a model from scratch on the sign language dataset. They showed that their method outperformed the traditional approaches in terms of recognition accuracy. They also showed that their method was able to generalize to signs not included in the training data, demonstrating the potential of transfer learning for sign language recognition.

## IV. Materials and methods

### A. Methodology

In this section we discuss the specific steps that we did in order to build our real time American Sign Language detection system.

*1) Data-set:* For our data-set, we selected ASL Alphabet by Akash Nagaraj [3]. This training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING. Additionally, each class is perfectly balanced (it contains the same number of images). Loading some images from the data-set in a Jupyter Notebook yields Fig. 1.

*2) Data-set augmentation:* Even though the original model contains a wide variety of images in all classes, in real life, signs won't be perfect, in Fig. 2 we augment the data using realistic movements that could be found in practice like zooming, translation and rotation. Most signs will be well aligned though, is not reasonable to expect a sign that's too far from the standard pose, so 5% is a reasonable compromise. This technique help us avoid over-fitting, increases the flexibility of the model, generalization, and tolerance to variations in the data-set. Some examples of this transformations can be found in Fig. 3.

*3) Pre-trained model:* For our base model, we selected EfficientNetB2 [4] with pre-trained weights on Imagenet. It provides a reasonable trade-off between number of parameters (9.2M) accuracy, top-5 accuracy (94.9%) and time per inference step.

*4) Transfer learning:* For this and future sections, we followed the recommendations in "Transfer learning and fine-tuning" tutorial by François Chollet (MIT License) [2].

*5) Load the base model:* We loaded the EfficientNetB2 model with weights pre-trained on Imagenet, but without the top layer (the prediction layer) and froze the layers; we don't want to loose the features the model already contains during

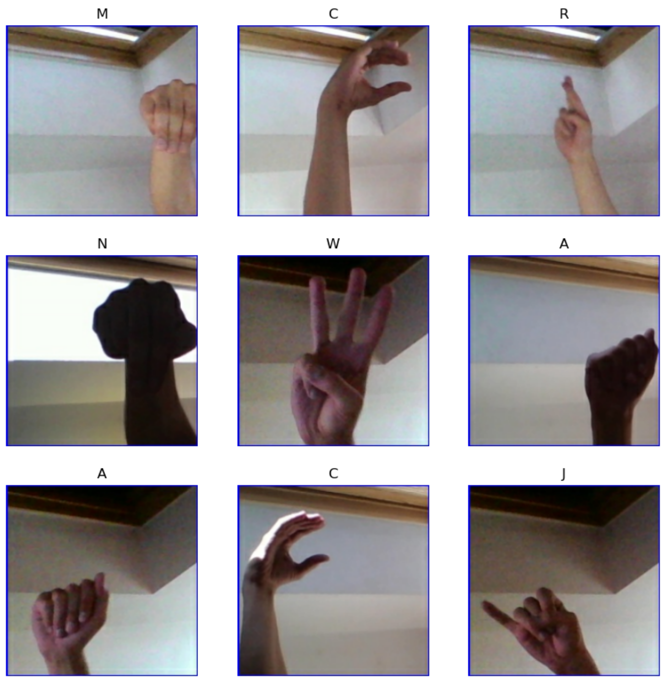
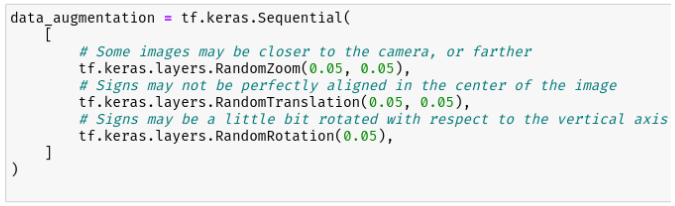


Fig. 1. American Sign Language images and their corresponding labels.



```
data_augmentation = tf.keras.Sequential(
    [
        # Some images may be closer to the camera, or farther
        tf.keras.layers.RandomZoom(0.05, 0.05),
        # Signs may not be perfectly aligned in the center of the image
        tf.keras.layers.RandomTranslation(0.05, 0.05),
        # Signs may be a little bit rotated with respect to the vertical axis
        tf.keras.layers.RandomRotation(0.05),
    ]
)
```

Fig. 2. Data augmentation steps.

our training. At this point we have 7,768,569 parameters (0 trainable, 7,768,569 non-trainable).

*6) Assemble our custom model:* Here we add the missing layers to the base model to complement it and adapt it to our problem domain. The overall architecture is:

1) Input layer (224x224x3).
2) Data Augmentation (random zoom, translation and rotation).
3) Prepossessing using the efficientnet function.
4) Base Model (in no-training mode).
5) Pooling layer.
6) Dropout layer (prevent over-fitting).
7) Dense layer (prediction).

Whose number of parameters can be seen in Fig. 4

*7) First Training Cycle:* The goal of this training cycle is to tune the parameters in the top layer of our model (the prediction layer, which is a dense layer).

In order to keep training times as small as possible we configured an EarlyStopping callback that monitors the loss function in the validation data-set. If it does not improve for 3 continuous epochs, the training is stopped.

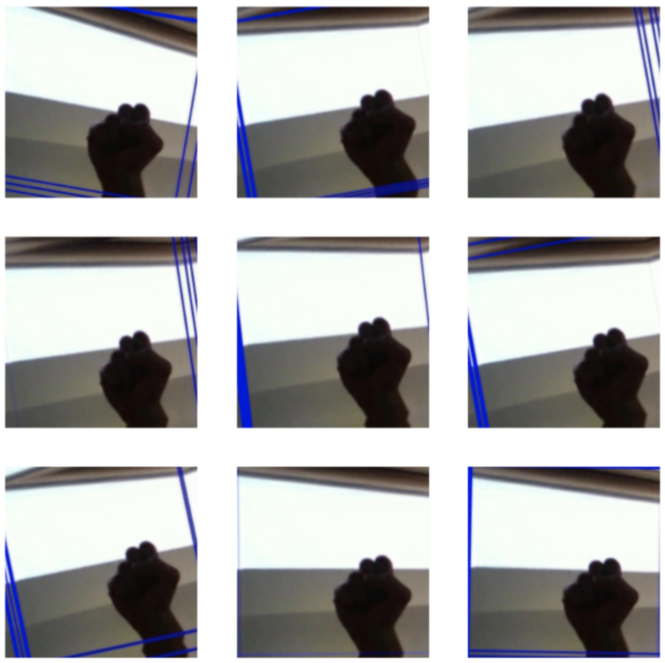
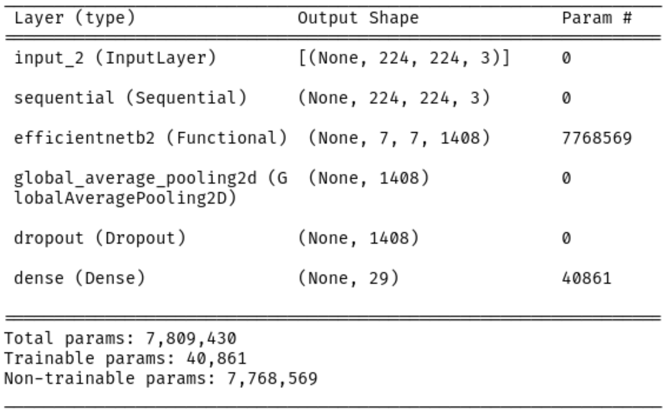


Fig. 3. Data augmentation examples.



```
Layer (type)                 Output Shape              Param #
================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0

sequential (Sequential)      (None, 224, 224, 3)       0

efficientnetb2 (Functional)   (None, 7, 7, 1408)       7768569

global_average_pooling2d (G   (None, 1408)             0
lobalAveragePooling2D)

dropout (Dropout)            (None, 1408)              0

dense (Dense)                (None, 29)                40861

================================================================
Total params: 7,809,430
Trainable params: 40,861
Non-trainable params: 7,768,569
________________________________________________________________
```

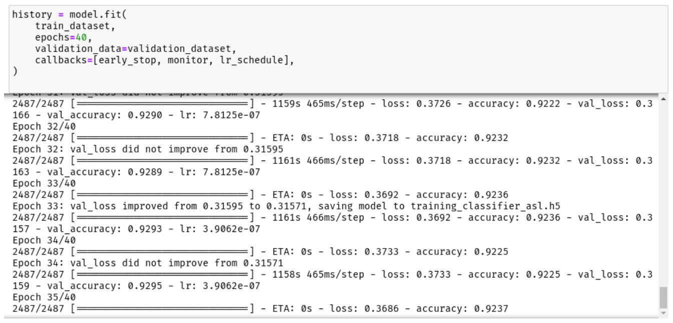
Fig. 4. Our model architecture.

Since we expect training to run over a 12 hours period, we also added a ModelCheckpoint callback that saves the best model after each epoch. This gives us the opportunity to save the best model on the go and not having to wait until the whole model is trained to see the results. It also serves as a ”backup” in case the training interrupts for some reason (for instance a system failure).

Additionally, we added a learning rate scheduler callback, which cuts in half the learning rate every four epochs.

Then we simply compile the model using an Adam optimizer and a CategoricalCrossentropy loss function, and fit it using 40 epochs as seen in Fig. 5.

Training was done both in our local laptops and in TALC (see SLURM file in the repository), but ultimately we run it for 12 hours on an EC2 c5.4xlarge AWS instance to avoid the



```
history = model.fit(
    train_dataset,
    epochs=40,
    validation_data=validation_dataset,
    callbacks=[early_stop, monitor, lr_schedule],
)
```

```
2487/2487 [==============================] - 1159s 465ms/step - loss: 0.3726 - accuracy: 0.9222 - val_loss: 0.3
166 - val_accuracy: 0.9290 - lr: 7.8125e-07
Epoch 32/40
2487/2487 [==============================] - ETA: 0s - loss: 0.3718 - accuracy: 0.9232
Epoch 32: val_loss did not improve from 0.31595
2487/2487 [==============================] - 1161s 466ms/step - loss: 0.3718 - accuracy: 0.9232 - val_loss: 0.3
163 - val_accuracy: 0.9289 - lr: 7.8125e-07
Epoch 33/40
2487/2487 [==============================] - ETA: 0s - loss: 0.3692 - accuracy: 0.9236
Epoch 33: val_loss improved from 0.31595 to 0.31571, saving model to training_classifier_asl.h5
2487/2487 [==============================] - 1161s 466ms/step - loss: 0.3692 - accuracy: 0.9236 - val_loss: 0.3
157 - val_accuracy: 0.9293 - lr: 3.9062e-07
Epoch 34/40
2487/2487 [==============================] - ETA: 0s - loss: 0.3733 - accuracy: 0.9225
Epoch 34: val_loss did not improve from 0.31571
2487/2487 [==============================] - 1158s 465ms/step - loss: 0.3733 - accuracy: 0.9225 - val_loss: 0.3
159 - val_accuracy: 0.9295 - lr: 3.9062e-07
Epoch 35/40
2487/2487 [==============================] - ETA: 0s - loss: 0.3686 - accuracy: 0.9237
```

Fig. 5. Epoch 35, 92% training accuracy, 93% validation accuracy.

wait times and limitations of TALC (TALC’s GPU nodes were busy most of the time when we tried running it).

The model converged pretty well as shown in Fig. 6.

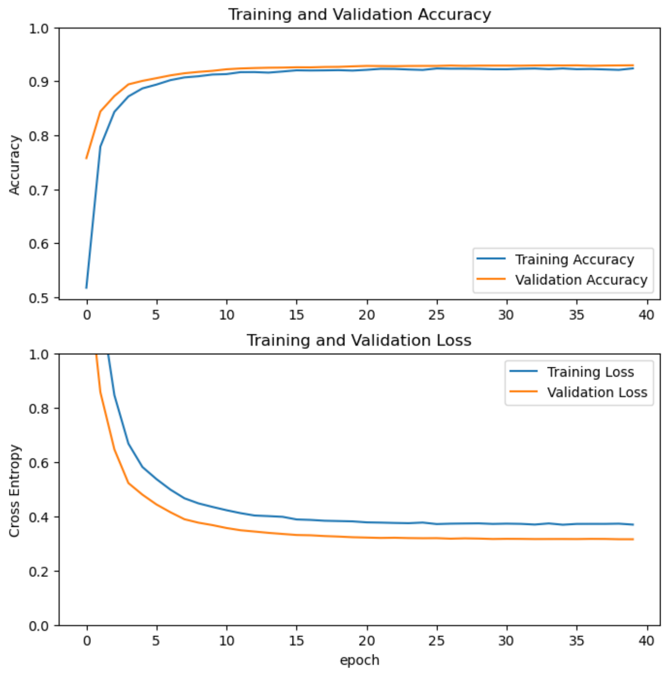
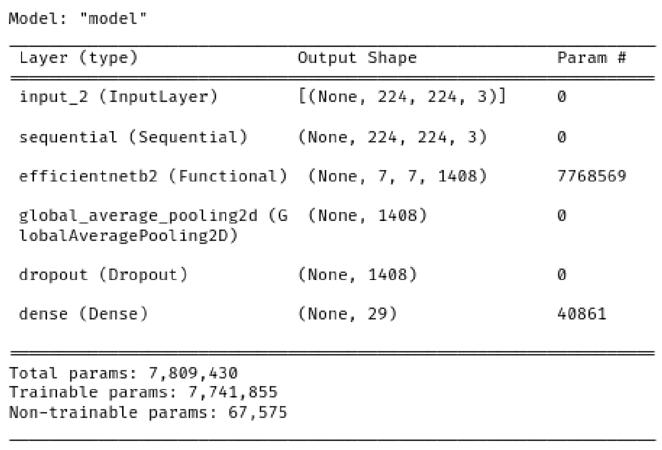


Fig. 6. Training and validation accuracy and loss over time.

*8) Second Training Cycle:* In this cycle our goal is to fine tune the model. So all we do is load the model, set all layers as trainable (unfreeze the model), compiling using the same Adam optimizer but with a very very small learning rate, the same loss function and callbacks used for the first training cycle, and training for 3 epochs only.

Fig. 7 contains our model architecture at this point. As you can see, most parameters are now trainable, this to allow the model to adjust its features to the current problem domain.

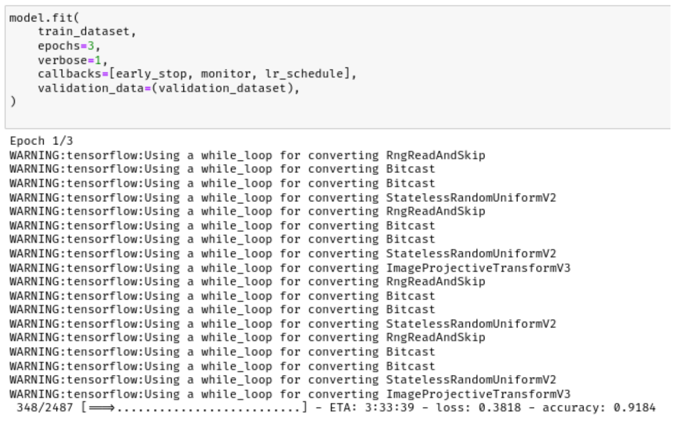
Unfortunately, as seen in Fig. 8, having so many parameters is a very slow process (4 hours per epoch), so we stopped it and decided that the 92% accuracy of our model with transfer learning is good enough. In any case, we believe fine-tuning the model has the potential for improving the model’s



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential (Sequential)     (None, 224, 224, 3)       0

 efficientnetb2 (Functional)  (None, 7, 7, 1408)       7768569

 global_average_pooling2d (G  (None, 1408)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1408)              0

 dense (Dense)               (None, 29)                40861

=================================================================
Total params: 7,809,430
Trainable params: 7,741,855
Non-trainable params: 67,575
_________________________________________________________________
```

Fig. 7. Our model architecture for fine-tuning.

accuracy, but we'll leave this experiment for a day when resources are not a constraint.



```
model.fit(
    train_dataset,
    epochs=3,
    verbose=1,
    callbacks=[early_stop, monitor, lr_schedule],
    validation_data=(validation_dataset),
)
```

```
Epoch 1/3
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
 348/2487 [==>.........................] - ETA: 3:33:39 - loss: 0.3818 - accuracy: 0.9184
```

Fig. 8. American Sign Language alphabet.

*9) Live American Sign Language detection system :* In this section, our goal is to create a software application that connects to a real-time camera and detects the American Sign Language made by a person in the camera.

For reference, Fig. 9 contains the American Sign Language alphabet. Please notice letter C and O; the difference between them is just if the fingers touch.

Fig. 10 shows letter C being detected with 92% accuracy by our model and Fig. 11 shows letter O being detected with 100% accuracy by our model.

In order to accomplish this goal we created a custom script using the OpenCV and MediaPipe. This script can be found in the code repository for this project.

In essence the script perform the following steps:

1) It loads our fully trained model.
2) It uses OpenCV to capture the live camera feed and take screenshots at regular intervals.
3) It uses MediaPipe to recognize the portion of the screenshot that contains "hands" and extracts it into 224x224px images that are feed into our model.

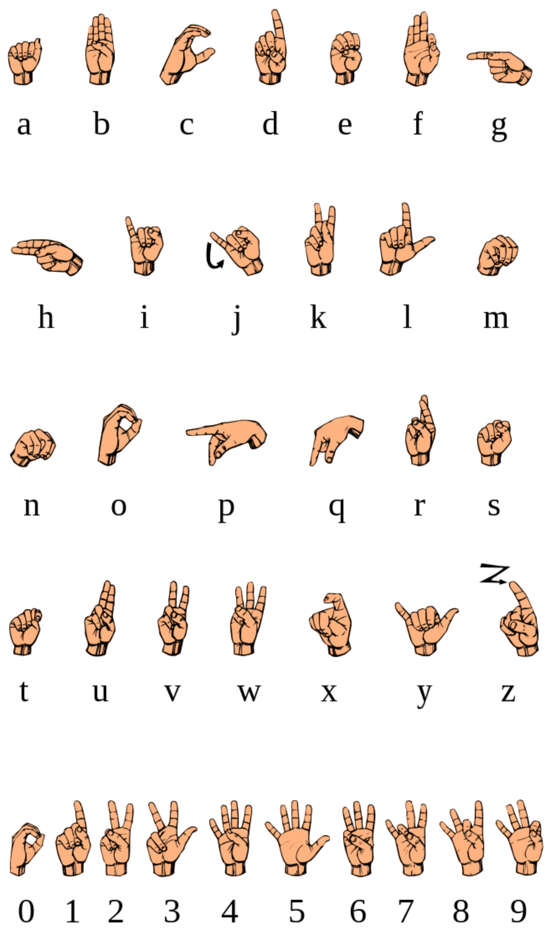


Fig. 9. American Sign Language alphabet.

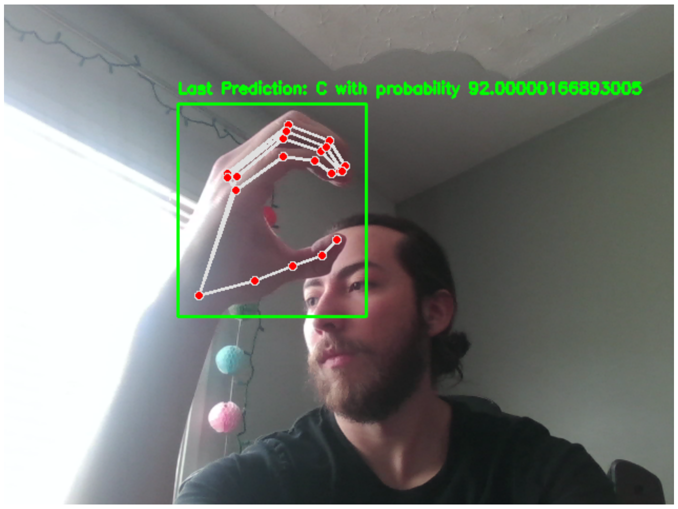


Fig. 10. Our system detecting letter C.

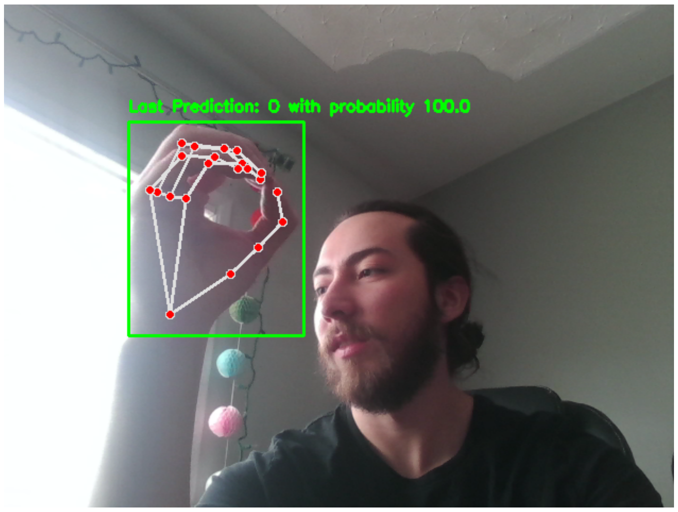


Fig. 11. Our system detecting letter O.

4) Our model performs the inference.
5) We use OpenCV to render the inference result back in the live camera feed.

### *B. Strengths and limitations of the methodology*

Our approach of using transfer learning with a pre-trained EfficientNetB2 model on Imagenet dataset to recognize American Sign Language has several strengths. Firstly, by using transfer learning, the model is able to learn high-level features from a large dataset and apply them to a different domain, which allows for better generalization. Data augmentation techniques such as zooming, translation, and rotation can also help increase the model's flexibility, generalization, and tolerance to variations in the data set, reducing overfitting. This is very important. Despite the fact that images in the training dataset contains a wide variety of images from real scenarios, it is beneficial to make the data set more diverse, so the model is able to recognise signs from many different scenarios which may present themselves. Additionally, early stopping and model checkpoint callbacks enable the model to save the best version of itself, reducing the risk of losing valuable training progress in case of any system failures. The learning rate scheduler also allows the model to converge faster by adjusting the learning rate at specific intervals.

However, there are also several weaknesses to this approach. Firstly, EfficientNetB2 is still a large model with 9.2M parameters, which can require a significant amount of computational resources and training time. This makes it nearly impossible to run on low-end devices such as mobile phones or Raspberry Pi. Secondly, despite the use of data augmentation techniques, the model may still not be robust to significant variations in real-world signs that were not captured in the data set, leading to false positives or false negatives. Thirdly, since the data set only contains images, the model may not be able to capture the temporal information of the sign language, which is crucial for understanding the context and meaning of a sentence. Lastly, the real-time American Sign Language detection system may require high frame rates to work accurately, which may pose challenges for low-end cameras or computing devices. Additionally, the system may require a reliable internet connection to transmit the data from the camera to the server and back, which may not always be feasible in all settings.

## V. Results and discussions

1) As shown in Fig. 5, the model we trained using transfer learning achieved 92% training accuracy and 93% validation accuracy over the 87000 images in our data-set.
   Discussion: Although the results were excellent in the data-set we used, we still believe there is room for improvement, for instance, most images in our data-set were from the same person (hand size, skin tone, fingers length). A more robust model would be one trained on different hands since the variation of hands sizes in real life is very big. Moreover, some people even lack fingers, and our model is completely unaware of that.
2) As shown in Fig. 8, the model we trained didn't really benefit much from fine-tuning.
   Discussion: In our methodology, we only ran it for 3 epochs. An interesting test would be running it for more epochs. We were limited in the amount of computing resources we could dedicate since our model had around 8 million parameters and each epoch was around 4 hours of time.
3) We proved that the model is converging very well overtime as seen in Fig. 6.
   Discussion: The ModelCheckpoint callback was a huge thing in our approach because it allowed us to save the model as it was being trained, which gave us a lot of peace of mind that we were going to have a model after each epoch even if the system crashed or something happened to the node running the computation. Fortunately, this didn't happen, but still, it was very useful because we were able to import the best model after each epoch into our live camera detection system and see if it was actually improving or not, instead of waiting the full 12 hours that the transfer learning session took.
   Discussion: The EarlyStopping callback never was triggered, it was supposed to stop after 3 consecutive epochs were the validation loss didn't improve, but the most it got was to 2 consecutive epochs, so ultimately the model was trained for the full 40 epochs (around 12 hours). We could have been more aggressive with the stop condition.
4) The model statistics are good, but we also showed that our model has applications in real life in Fig. 10 and Fig. 11, where we see that the predictions in a real camera match closely the ASL alphabet in Fig. 9.
   Discussion: Our model detects static signs, but some signs like the letter Z requires a hand movement, which our model is unable to see. An interesting research topic would be to see how this movements could be detected.
5) The confusion matrix of the fully trained model in Fig. 12 shows that overall our model classifies signs

correctly, and in a minority of cases it confuses some sign with others.

Discussion: The cases were signs were miss-classified do not show any apparent relationship. Common sense would say that the model could confuse for instance letter S with T since they are almost the same (their signs differ in just a slight displacement of the thumb). But interestingly, this is not the case, and our model gets confused between signs that are very different. Our thoughts on this could be checking the dropout layer, adding (or increasing) a couple more convolution layers, and using a data-set where hands are bigger within the image so that details are more apparent (our data-set has small hands with respect to the image size, so there is a lot of background noise that the model needs to learn to ignore).

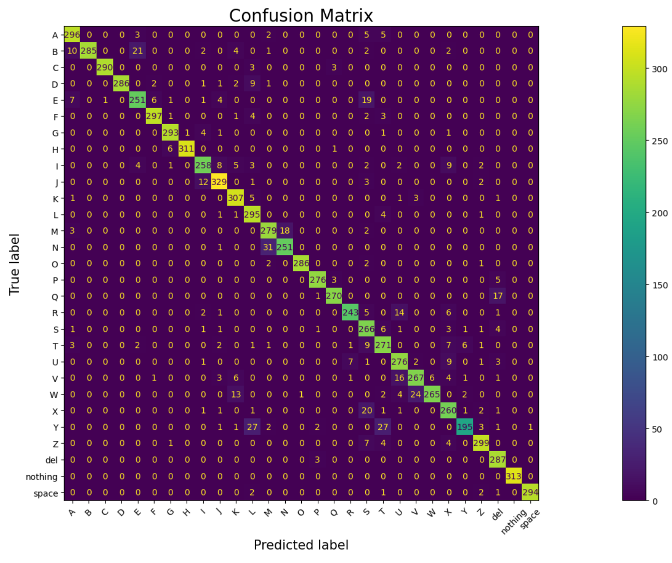


Fig. 12. Confusion matrix of the fully trained model.

## VI. Conclusion

This paper presents a successful approach to American Sign Language (ASL) recognition using deep learning and transfer learning techniques. We successfully trained a neural network on a large data-set of ASL images and utilized pre-trained weights from a state-of-the-art image classification model and the resulting model achieved high accuracy in classifying various sign language gestures.

Moreover, we were able to deploy this model in a real-time sign language detection system using a camera feed. The system was able to recognize sign language gestures in real-time with low latency and achieved high accuracy in identifying signs from multiple individuals.

The paper's results indicate significant potential for enhancing accessibility for people who use sign language as their primary mode of communication, allowing them to communicate more efficiently with others who may not know sign language.

Overall, we demonstrated the effectiveness of deep learning and transfer learning techniques in ASL recognition and highlights the potential for technology to improve accessibility and communication for people with disabilities.

## VII. Team work Rating

### A. Bill Yu

1) *Score (1 to 3):* 3

2) *Participation:* Dataset Selection, Related Work, Strengths and limitations of methodology, References, Presentation slides

### B. Hollis Holmes

1) *Score (1 to 3):* 3

2) *Participation:* Live camera script, data import using image_dataset_from_directory to reduce RAM usage, data splitting, prepossessing, training, some plotting.

### C. Kevin Amado

1) *Score (1 to 3):* 3

2) *Participation:* In source code: Data augmentation, MLOps in AWS, repository etiquette and code formatting, tried to fix a bug that was causing our model to have 3% accuracy but failed in the attempt (Hollis found the bug).

In the report: abstract, index, introduction, methodology, data-set, data-set augmentation, pre-trained model, base-model, model arch, transfer learning, fine-tuning, live asl detection, results and discussions, conclusions, references, figures.

### D. Tyson Trail

1) *Score (1 to 3):* 3

2) *Participation:* Dataset selection, data importing, formatting, and pre-processing, transfer learning model selection, training, and testing, plotting images and results.

### E. Vladyslav Timofyeyev

1) *Score (1 to 3):* 3

2) *Participation:* MLOps and setting the code up to be ran in TALC, troubleshooting data importing, troubleshooting memory and GPU issues.

## References

[1] American Sign Language. (2023, March 31). In Wikipedia. https://en.wikipedia.org/wiki/American_Sign_Language

[2] https://github.com/tensorflow/docs, /blob/master/site/en/tutorials/images/transfer_learning.ipynb, & François Chollet. (2017) Transfer learning and fine-tuning.

[3] https://www.kaggle.com/grassknoted/aslalphabet, & Akash Nagaraj. (2018). ASL Alphabet [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/29550

[4] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ArXiv. /abs/1905.11946

[5] Wadhawan, A., & Kumar, P. (2020). Deep learning-based sign language recognition system for static signs. Neural computing and applications, 32, 7957-7968.

[6] Bantupalli, K., & Xie, Y. (2018, December). American sign language recognition using deep learning and computer vision. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 4896-4899). IEEE.

[7] Taskiran, M., Killioglu, M., & Kahraman, N. (2018, July). A real-time system for recognition of American sign language by using deep learning. In 2018 41st international conference on telecommunications and signal processing (TSP) (pp. 1-5). IEEE.

[8] Gattupalli, S., Ghaderi, A., & Athitsos, V. (2016, June). Evaluation of deep learning based pose estimation for sign language recognition. In Proceedings of the 9th ACM international conference on pervasive technologies related to assistive environments (pp. 1-7).

[9] Farhadi, A., Forsyth, D., & White, R. (2007, June). Transfer learning in sign language. In 2007 IEEE conference on computer vision and pattern recognition (pp. 1-8). IEEE.

[10] Jiang, X., Hu, B., Chandra Satapathy, S., Wang, S. H., & Zhang, Y. D. (2020). Fingerspelling identification for Chinese sign language via AlexNet-based transfer learning and Adam optimizer. Scientific Programming, 2020, 1-13.

[11] Rathi, D. (2018). Optimization of transfer learning for sign language recognition targeting mobile platform. arXiv preprint arXiv:1805.06618.

[12] Sarhan, N., & Frintrop, S. (2020, October). Transfer learning for videos: from action recognition to sign language recognition. In 2020 IEEE International Conference on Image Processing (ICIP) (pp. 1811-1815). IEEE.

[13] Garcia, B., & Viesca, S. A. (2016). Real-time American sign language recognition with convolutional neural networks. Convolutional Neural Networks for Visual Recognition, 2(225-232), 8.

[14] Cayamcela, M. E. M., & Lim, W. (2019, February). Fine-tuning a pre-trained convolutional neural network model to translate American sign language in real-time. In 2019 International Conference on Computing, Networking and Communications (ICNC) (pp. 100-104). IEEE.

[15] Papastratis, I., Chatzikonstantinou, C., Konstantinidis, D., Dimitropoulos, K., & Daras, P. (2021). Artificial intelligence technologies for sign language. Sensors, 21(17), 5843.

[16] Adithya, V., Vinod, P. R., & Gopalakrishnan, U. (2013, April). Artificial neural network based method for Indian sign language recognition. In 2013 IEEE conference on information & communication technologies (pp. 1080-1085). Ieee.

[17] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141.