Real-Time Hand Gesture Recognition

Rounak Salim, Anmol Dhoor, Jayam Thaker
San Jose State University
San Jose, U.S.A.
rounak.salim@sjsu.edu, anmol.dhoor@sjsu.edu, jayam.thaker@sjsu.edu

***Abstract*—The last few years have turned out multitudes of research on image classification. One can find models to identify anything, from general household objects to specific species of trees and plants. Our project is inspired by such classification models to identify different hand gestures. This could potentially be used to control computer applications or even one day understand sign language. Our application is made to be lightweight, running in real-time without using any external devices such as powerful GPUs. It uses a computer's webcam to capture images of the user and recognizes their hand gestures, making several predictions per second. We gathered a dataset covering 5 gestures to improve our results, consisting of 16,000 images in total. Finally, we were able to build a real-time system capable of identifying any of the five hand gestures using just the computer's webcam and CPU.**

I. Introduction

Image classification is an upcoming field that has received much attention over the years. Along with classification, we have powerful models that also carry out the task of object detection [1]. While most of these classification or object detection models offer great accuracy, they are also very big models that require high computational power, even for the task of inference.

Inspired by futuristic sci-fi movies, we decided to build an application that allows users to control their computers using hand gestures in real-time. We found this to be a good application of the various image classification capabilities present these days. While classifying images is not a hard task these days with the availability of pre-trained models, doing so quickly with a CPU can get challenging with large models. Since our project requires only hand gestures to be classified, we decided that pre-training an existing fast model such as Yolo [1] or building our own lightweight model is the path forward.

With such an application, people could control different aspects of their computers with just their gestures, such as controlling the volume, switching through different applications, going back and forth through slides in a presentation, etc. Besides this, we have also discovered that datasets for sign language are very similar to the hand gesture dataset we used, and therefore, a similar application and model could also be used for the real-time translation of sign language.

We decided to build the application as a Python script since it offers ease of use, a vast array of libraries to extract images from the webcam, and is also the default choice for building machine learning models. Our project consists of two major components. The first is to collect images in real-time from the computer's webcam and run any pre-processing steps required for the images. The second component is centered around building and training a model to carry out the task of hand gesture classification with good inference times.

The following sections will discuss the details of our approach to building this system and the challenges associated with them. Section 2 consists of a review of relevant and existing work being done in this area. Section 3 will discuss the technical details of building the real-time hand gesture recognition system. Section 4 will contain the results of our work, followed by the conclusion in section 5.

II. Literature Review

A lot of work has been done in the area of image classification. Some of the best performing models in terms of accuracy and speed are [1], [2], and [3]. These models generally work by segmenting the images into multiple different sections and then carrying out the task of detection and classification on each of these segments. While this works well for general images with a lot of different objects present in them, a system to recognize hand gestures such as ours doesn't require segmentation since the image only contains the hand gesture. Therefore, such pre-trained models can be used along with transfer learning for our dataset. However, if we were to build a network from scratch, we wouldn't need to perform segmentation and then detection and classification as our task is to classify an image containing just a single point of interest, the hand gesture.

Other work done by Waseem Rawat and Zenghui Wang [4] show that deeper CNNs generally perform better as long as the architecture avoids overfitting to the training data. However, they also show that deeper and larger architectures require high computational resources and take longer for inference. Therefore, we'll have to focus on the tradeoff between accuracy and inference speed while experimenting with our CNN models for the real-time detection system.

Data can often be the limiting factor in image classification systems. Jason Wang and Luis Perez show in [5] that data augmentation can be a highly effective strategy for improving the performance of a CNN with limited data. Various techniques such as cropping the image, rotating the image, and introducing Gaussian noise in the image can augment the dataset and help the model learn better in a generalized manner. We can explore using similar data augmentation techniques for our models as our dataset is also limited in size. Besides being limited in size, the dataset consists of very

similar images, and therefore augmentation will help with the generalization of the model as well.

III. Approach and Implementation

As the starting point for our project, we decided to work with a Hand Gesture Recognition Dataset from Kaggle [6]. The dataset contains 24,000 greyscale images of 20 different types of hand gestures. We elected to limit our scope to 5 hand gestures, as shown in Table 1. This means our dataset consists of 6,000 images, with 75% as training and 25% as test data. Table 2 shows a further breakdown of these numbers for clarity.

TABLE I
Classes Chosen for Our Dataset

Class # (in our dataset)	Class name	Example	Class # (in original dataset)
0	OK sign		0
1	L sign		10
2	Victory sign		2
3	Fingers crossed sign		13
4	Thumbs up		19

TABLE II
Image Counts in Our Dataset

	Training	Testing	Total
Images per class	900	300	1200
Images in dataset	4500	1500	6000
Percentage	75%	25%	100%

A. Image Pipeline

We built the image pipeline of our system using Python. We were able to make use of the popular OpenCV library [7] to capture images from webcams. OpenCV's library also offers useful functions to crop and convert images to grayscale.

Using OpenCV, we first build a bounding box in the corner of the screen, as seen in Fig. 1. The user is expected to place their hand gesture in this bounding box, and only the portion within the bounding box is captured by our program. This helps eliminate various objects surrounding the user.

After capturing the image, our program runs a simple background elimination algorithm in which it uses certain threshold values to grayscale the image so that the gesture is in white and the background is black to match the images from our dataset. An example of this can be seen in Fig. 2.

Once we have the final grayscale image of the hand gesture, we crop and resize the image so that it's 50x50 pixels to match the image size from the Kaggle dataset. This final image is then fed into our model to carry out the task of classifying the hand gesture. The results of the prediction, the name of the class and the confidence, are printed on the screen for the user to see.

Fig. 1. Webcam Image Showing a Frame in the Top-Right Corner as the Bounding Box for the Hand Gesture

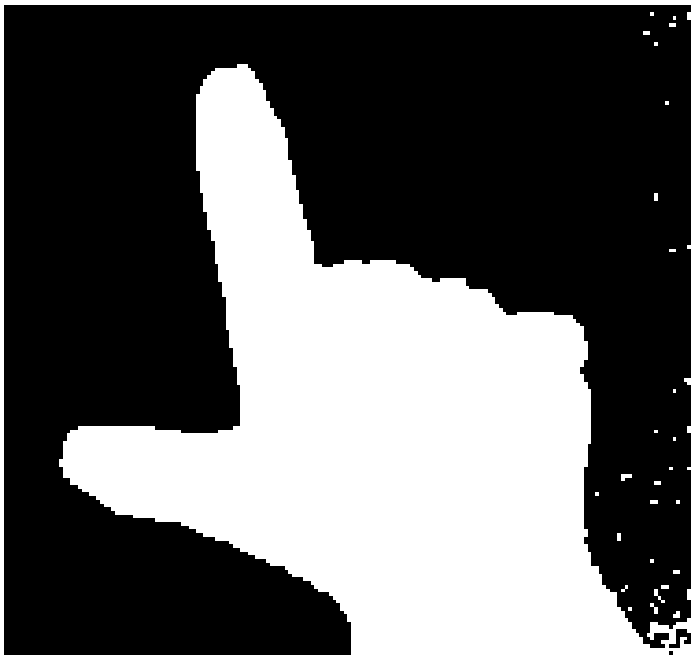

Fig. 2. Image From the Webcam After Being Processed

B. Classification Model: Yolo v5

Since Yolo is known as a real-time, lightweight model, we used it as the basis for our first approach. We used the chosen portion of the Kaggle dataset and further partitioned the data, as indicated in Table 3.

To train this model, we needed to create annotations for the data–information that would specify the location of the true bounding box and the true classification of each object in the

TABLE III
IMAGE COUNTS IN OUR DATASET FOR THE YOLO MODEL

	Training	Testing	Validation	Total
Images per class	750	300	150	1200
Images in dataset	3750	1500	750	6000
Percentage	62.5%	25%	12.5%	100%

image. Yolo v5's annotation format includes 3 parts: the class number, an x- and y- coordinate for the center of the bounding box, and an x- and y- coordinate for the size of the box. All coordinates are on a [0,1] scale.

The images in our dataset made this very simple. Each image holds only one object, which spans the entirety of the picture. Fig. 3 has an example of the annotation. Because each one is nearly the same (differing only in class numbers and file names), we automated the process of writing these annotations with a Python script.

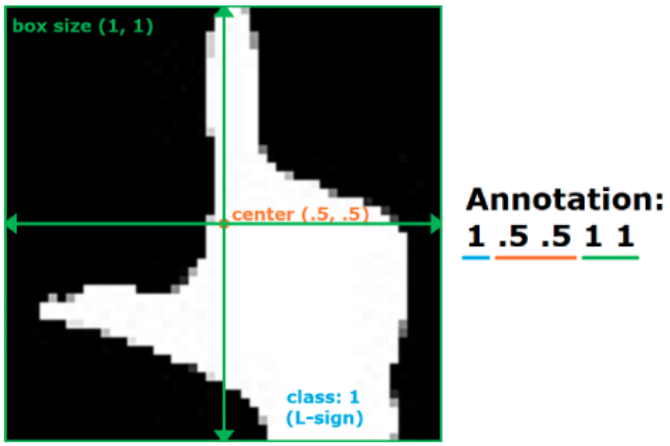

Fig. 3. Example of YOLO V5 Annotation on an Image From Our Dataset

To train the model, we used a basic set of hyperparameters for Yolo v5, defined in its official "hyp.scratch.yaml" configuration file. The batch size was 32 and the number of epochs was 100.

C. Classification Model: CNN

We also explored building our own CNN to classify the gestures. We built models with different architectures using Tensorflow and Keras. While building and testing models, we had to ensure that our model achieved the right trade-off between speed and accuracy. While bigger models lead to better results, they come at the expense of speed. Since our system needs to operate in real-time, we had to ensure that the model can classify an image within 500 milliseconds (2 inferences per second).

Fig. 5 and 6 show the two different CNN models that we built and tested on the chosen portion of the Kaggle dataset. While the smaller model consisted of just two convolutional and max-pooling layers and one dense layer, the larger one consisted of three convolutional and max-pooling layers and

```
model.summary()

Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_39 (Rescaling)    (None, 50, 50, 1)         0

 conv2d_38 (Conv2D)          (None, 24, 24, 2)         20

 max_pooling2d_36 (MaxPoolin  (None, 12, 12, 2)        0
 g2D)

 conv2d_39 (Conv2D)          (None, 5, 5, 2)           38

 max_pooling2d_37 (MaxPoolin  (None, 2, 2, 2)          0
 g2D)

 flatten_37 (Flatten)        (None, 8)                 0

 dense_53 (Dense)            (None, 5)                 45

=================================================================
Total params: 103
Trainable params: 103
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4. Smaller CNN Model

```
Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_40 (Rescaling)    (None, 50, 50, 1)         0

 conv2d_40 (Conv2D)          (None, 50, 50, 16)        160

 max_pooling2d_38 (MaxPoolin  (None, 25, 25, 16)       0
 g2D)

 conv2d_41 (Conv2D)          (None, 25, 25, 32)        4640

 max_pooling2d_39 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_42 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_40 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 dropout_18 (Dropout)        (None, 7, 7, 64)          0

 flatten_38 (Flatten)        (None, 3136)              0

 dense_54 (Dense)            (None, 128)               401536

 dense_55 (Dense)            (None, 5)                 645

=================================================================
Total params: 425,477
Trainable params: 425,477
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. Larger CNN Model

two dense layers. The larger model also had 15% dropout applied to it to help avoid overfitting. It can be observed that the smaller model contains significantly fewer parameters (103) than the larger model (425,477).

Both models were trained using the Adam optimizer and CategoricalCrossentropy loss function, which is the standard loss function used when handling multiple classification categories. We used a batch size of 16 and trained the models for 15 epochs. We found that since the dataset was relatively simple, training for 15 epochs was enough to give us the desired results.

D. Custom Dataset

While our CNN model performed well on the Kaggle dataset, we realized that our results on the webcam images were limited by the dataset itself. The images were too clean, making it difficult to re-create similar images when processing the captures from our webcam. In an attempt to solve this issue, we captured images of our own to create a custom dataset.

Each of our three team members captured about 500 images for each class. By splitting up the load in this way, we were able to gather data with varying environments and amounts of noise. We further augmented the data by copying and flipping images to ensure the model did not depend on a hand gesture facing a certain way–for example, a victory sign using one's left vs. right hand.

In total, our custom dataset is 16,000 images (broken down in Table 4). This is over 2.5x larger than the corresponding portion of the Kaggle dataset, which is 6,000 images. Fig. 6 compares images from each dataset, showing the improvement in variance and noise, which will allow the model to create a more generalized understanding of the hand gestures.

TABLE IV
IMAGE COUNTS IN OUR CUSTOM DATASET

	Training	**Testing**	**Total**
Images per class	2400	800	3200
Images in dataset	12000	4000	16000
Percentage	75%	25%	100%

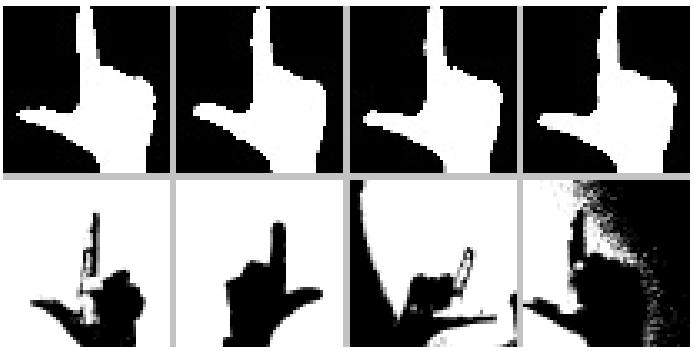

Fig. 6. Comparison of Images From Previous Dataset (Top Row) vs. Our Custom Dataset (Bottom Row)

IV. RESULTS

A. Yolo + Kaggle Dataset

Our initial attempt to apply transfer learning to a trained YoloV5 model did not give us the results we expected. We were only able to achieve a validation accuracy of 10% after training the model for 100 epochs. Further research on this topic led us to realize that we were conflating object detection and object classification. Yolo is designed to perform well on object detection problems, first deciding which parts of the image are objects vs. background, then classifying the objects. It does not perform well on bounded images like the ones in our dataset. This is why we switched to a different classification model.

B. CNN + Kaggle Dataset

Training our own CNN led to better results on our dataset, as the dataset is limited to only five different gestures, and such a classification task is relatively simple for a CNN. Both our smaller and larger models were able to achieve high validation accuracy of 100% after training for 15 epochs. Fig. 7 and 8 show the training curves of these models.

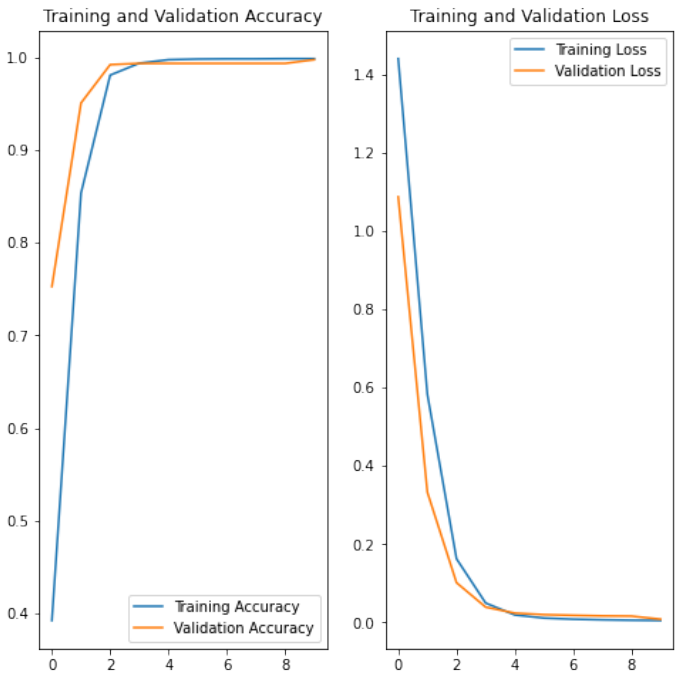

Fig. 7. Training Accuracy and Loss of the Smaller CNN Model

Our goal for inference speed was (at most) 500 milliseconds. The smaller model is able to classify an image within 100-150 ms, whereas the larger model takes 150-200 ms. Both these speeds are easily within 500 ms, fulfilling our purposes of real-time recognition. Therefore, we can use the larger and more accurate model, making 5 predictions per second.

We observed that the larger CNN model is able to learn a good representation of the dataset after just a single epoch as the model is quite dense. However, such a high validation

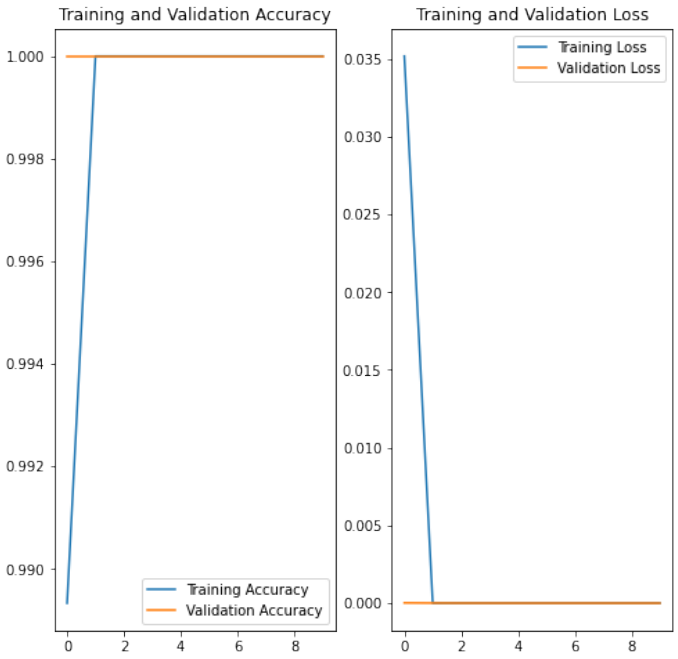

Fig. 8. Training Accuracy and Loss of the Larger CNN Model

accuracy also occurs because the test and training images in the Kaggle dataset are very similar. As previously discussed, there is not much noise or variance in this dataset. The model performs very well on the test data, but it is fitted to an idealized representation of the hand gesture. To get the model to recognize the right gestures from our webcams, we had to specifically tailor the environment and lighting, wear gloves, etc. Fig. 9 shows how difficult it can be to get an accurate prediction with this model.

Fig. 9. Example of Incorrect Classification Despite a Relatively Clear Image

C. CNN + Custom Dataset

The larger model's performance for webcam images greatly improved when we trained it instead on our custom dataset. It is less demanding about the environmental factors and demonstrably understands the difference between different gestures (while before, it was consistently showing a bias toward certain classes). Fig. 10 shows the training curves of the larger model trained on our custom dataset.

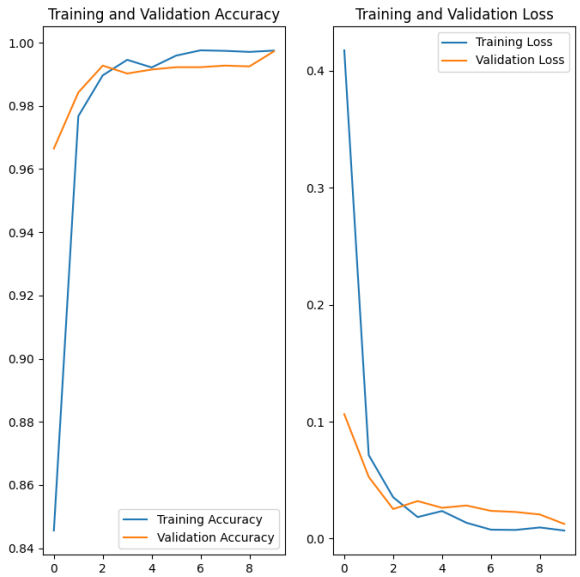

Fig. 10. Training Accuracy and Loss of the Larger Model Trained on Custom Dataset

As we can see, the model's accuracy increases steadily and isn't achieving 100% validation accuracy (gets close to 99% accuracy), which we encountered when training the same model on the Kaggle dataset. This indicates that this model does a much better job of learning from the dataset and doesn't have overfitting issues.

This model also performs prediction in 200 milliseconds and does a better job of successfully identifying each of the five hand gestures we trained on when looking at webcam images. These attributes made it the perfect model for our final real-time system. Fig. 11 shows successful predictions for each gesture, and a real-time demo of this can be found in the project repository (Note that it may take a few seconds for the GIF to load).

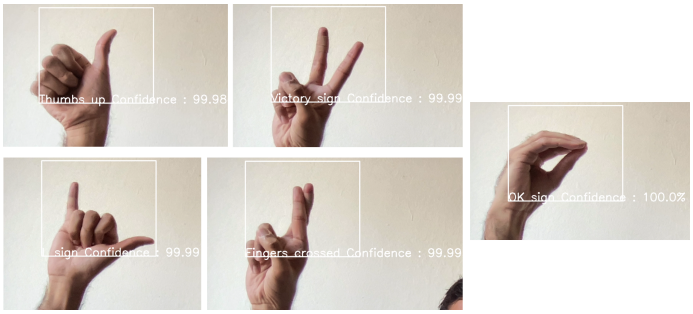

Fig. 11. Final Model Shows Accurate Predictions for All Five Gestures

V. Conclusion

Ultimately, we were able to achieve our goal of hand gesture classification in real-time. Our initial attempt with the Yolo model was fast but had insufficient accuracy. Once we switched to building a CNN with Tensorflow and Keras, we were able to achieve high accuracy and a prediction time of 200 ms.

When we moved on to classifying gestures from webcam images, our initial performance was rather poor. This led us to gather 16,000 images for a custom dataset with noisier and less idealistic pictures. The larger CNN model trained on the custom dataset led to the results we required from a model for our real-time recognition system. Our final system is able to capture images from the webcam in real-time, process them, and pass them to the model for inference which successfully classifies the gestures in real-time and presents the information to the user.

VI. Resources

- Project GitHub Repo: https://github.com/rounaksalim95/hand-gesture-recognition
- Project Demo Video: https://drive.google.com/file/d/1Qd90guLLBQ4pd93oSrekDvFHRVs_HxQk/view?usp=sharing
- Project Demo GIF: https://github.com/rounaksalim95/hand-gesture-recognition#demo

References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[2] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[4] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[5] Jason Wang, Luis Perez, et al. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*, 11:1–8, 2017.

[6] Kaggle. Hand gesture recognition dataset, 2022. [Online; accessed 22-May-2022].

[7] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3:2, 2000.