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*—Image classification is a field in which a lot of research has been done in the last few years. These days models are available that can identify anything from general house objects to specific species of plants and trees. Our project is inspired by such classification models. We are building an application that is capable of real-time hand gesture recognition. Our application not only detects a hand gesture but also maps it to a control on the user's computer. The application is built to be very lightweight and run in real-time without using any external devices such as powerful GPUs. The application is built to run on a computer with a webcam, and the model inference process also runs on the 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 lightweight own 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. Real-time Hand Gesture Recognition System

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 Fig. 1. This means our dataset consists of 6,000 images, with 75% as training and 25% as test data. Fig. 2 shows a further breakdown of these numbers for clarity.

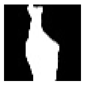
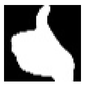
Class # (in our dataset)	Class name	Example	Class # (in original dataset)
0	OK sign		0
1	L sign		10
2	Victory sign		2
3	Fingers crossed		13
4	Thumbs up		19

Fig. 1. Classes Chosen for Our Dataset

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

Fig. 2. Image Counts in Our Dataset

A. Image Pipeline

We build 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 greyscale we made use of. Using OpenCV we first build a bounding box on the top right corner of the screen as seen in Fig. 3. The user is expected to place their hand gesture in this bounding box and only the image from the bounding box is captured by our program. This helps eliminate various objects in the frame of the image.

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. 4.

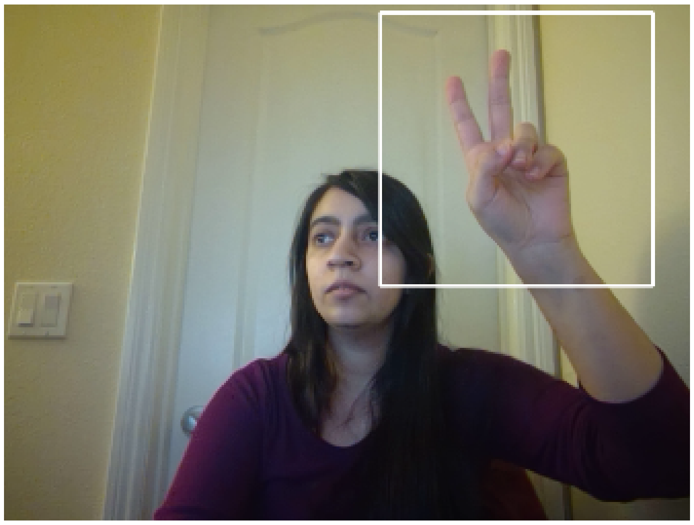


Fig. 3. Webcam Image Showing a Frame in the Top-Right Corner as the Bounding Box for the Hand Gesture

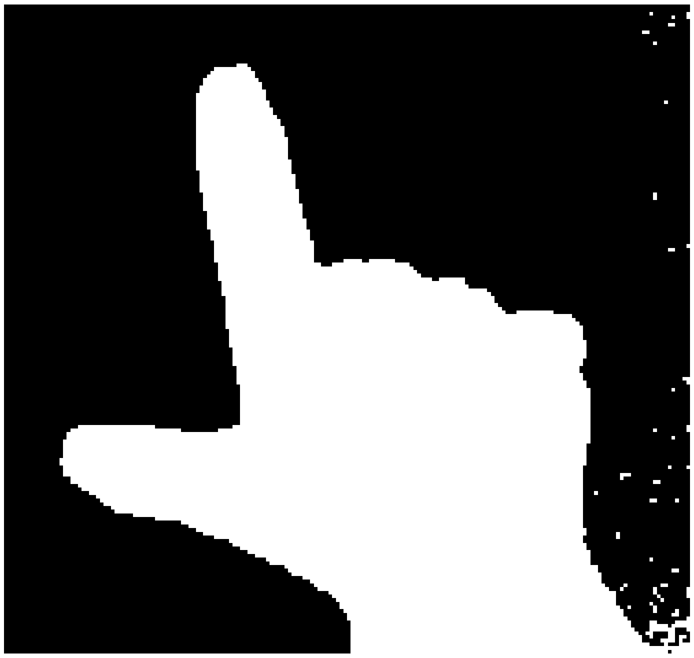


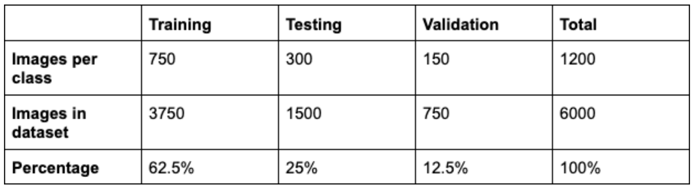
Fig. 4. Image From the Webcam After Being Processed

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 of our dataset. This final image is then fed into our model to carry out the task of classifying the hand gesture.

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 further partitioned the data, as indicated in Fig. 5.

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



	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%

Fig. 5. Image Counts in Our Dataset for the YOLO Model

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. 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.

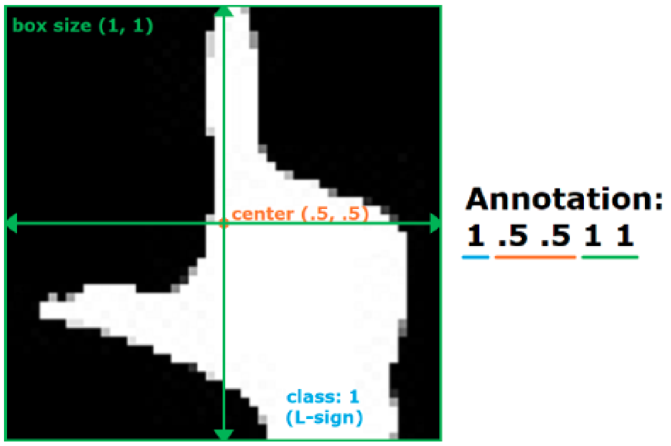


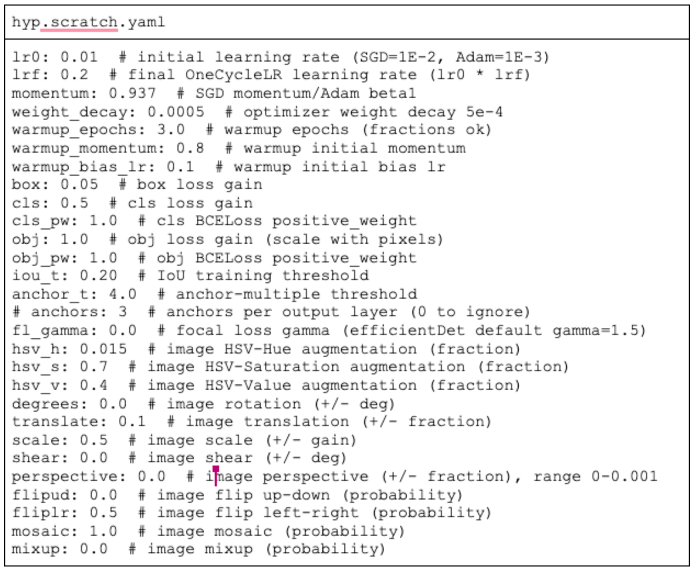
Fig. 6. Example of YOLO V5 Annotation on an Image From Our Dataset

To train the model, we used the basic set of hyperparameters for Yolo v5 as shown in Figure 4. The batch size was 32 and the number of epochs was 100.

C. Classification Model: CNN

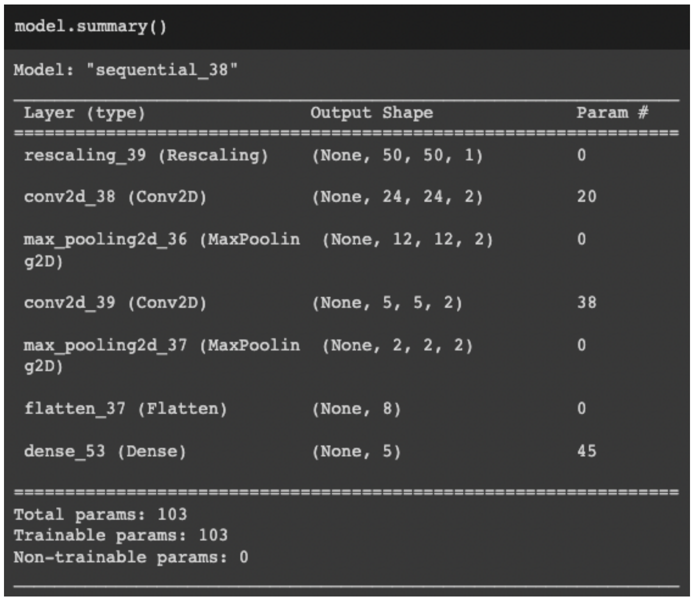
We also explored building our own CNN to classify the gestures. We built models with different architectures using Tensorflow and Keras as the API for it. While building and testing models, we had to ensure that our model achieved the right tradeoff between speed and accuracy. While bigger models lead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

Fig. 8 and 9 show the two different CNN models that we built and tested on our 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 two dense layers. The larger model also had 20



```
hyp.scratch.yaml

lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
```

Fig. 7. Copy of "hyp.scratch.yaml", the Training Configuration Covering Hyperparameters for Our YOLO V5 Model



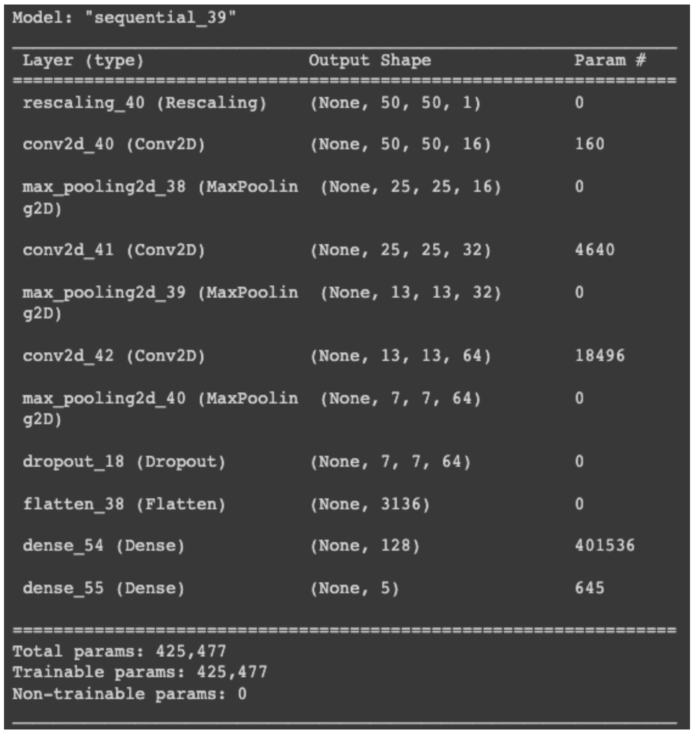
```
model.summary()

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. 8. Smaller CNN Model

Both models were trained using the Adam optimizer and CategoricalCrossentropy loss function which is the standard loss function that is used when multiple classification categories are present. A batch size of 16 was used and the models were trained for 15 epochs. We found out that since our dataset was relatively simple, training for 15 epochs gave us the desired results.

IV. Results

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



```
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. 9. Larger CNN Model

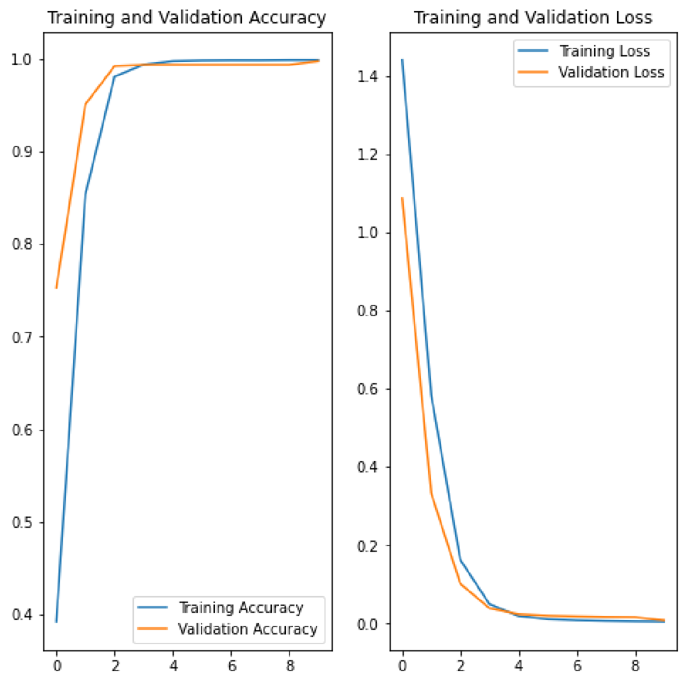


Fig. 10. Training Accuracy and Loss of the Smaller CNN Model

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.

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 96% after training for 15 epochs. Fig. 10 and 11 show the training curves of the models.

In terms of inference speed, the smaller model is able to classify an image within 100-150 milliseconds whereas the larger model takes 150-200 milliseconds. Both speeds work for our purposes of real-time recognition. Therefore, we can make use of the larger more accurate model since we can make 5 predictions per second with it.

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 accuracy also occurs because our test and training images in the dataset are very similar. Because there is not much noise or variance in our dataset, the model performs very well on the test data, but not very well on images from our webcams, as it does not have a generalized understanding of the hand gesture.

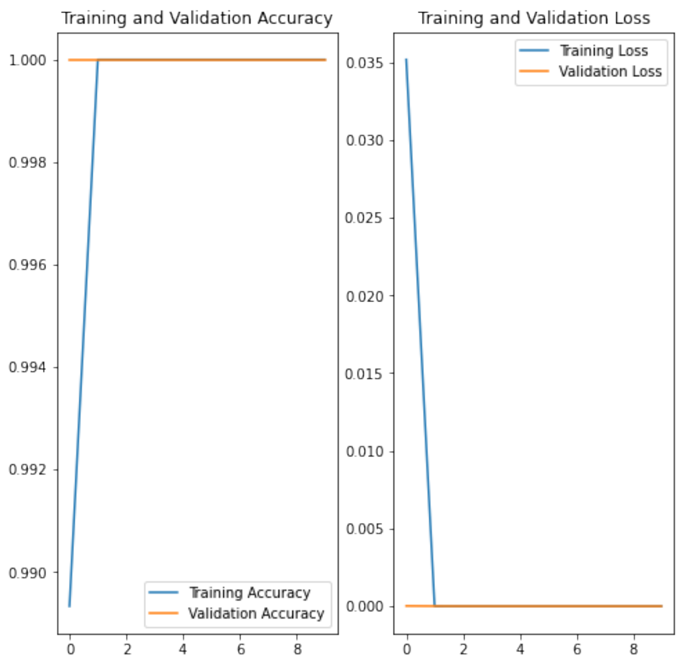


Fig. 11. Training Accuracy and Loss of the Larger CNN Model

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 classify the hand gestures correctly, with high accuracy and a prediction time well within the limits of ”real-time.” We were also able to build our pipeline to capture images from the webcam, process them, and pass them to the trained model for predictions.

However, the performance of our CNN model on our webcam images could be improved. To get clean images, we had to manually adjust the lighting, ensure the background was empty, and even wear gloves in some cases. Ideally, we would like our model to be able to classify gestures correctly even with a messier input. To solve this problem, we would want to augment the dataset with images from our webcam. Data augmentation can help introduce noise and variance in our dataset. Therefore, the model would be able to learn a more generalized representation of the hand gestures and improve its classification.

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.