Gesture Recognition using Webcam and CNN

Khalid Brelvi, Lucas Connell

Swanson School of Engineering, University of Pittsburgh
3700 O'Hara St, Pittsburgh, PA 15213
kfb23@pitt.edu, lzc6@pitt.edu

Abstract— **This paper presents two contrasting approaches to convolutional neural network (CNN) machine learning model development to accomplish the task of hand gesture recognition using a webcam: custom development, training, and tuning of a sequential neural network using PyTorch, and the use of MobileNet as a basis for transfer learning. Both models use respective custom created datasets representing 7 hand gestures: thumbs up, thumbs down, and numbers one through five. It is determined through implementation of both models that CNNs are sufficient for object detection and classification with some caveats that will be discussed. Khalid Brelvi researched and implemented the custom developed and trained sequential CNN, and Lucas Connell researched and implemented the MobileNet transfer learning network.**

I. Introduction

Hand gestures have played a pivotal role in human communication throughout all of history. Be it a friendly wave or a stern command, we can convey emotion and intent through a simple gesture. As we become more reliant on computers, we inevitably seek different ways to interface with the machines. Computers can already understand speech, but the field of gesture recognition is one that is ripe for exploration.

Unfortunately, gestures can vary wildly from person to person and moment to moment. Spoken language follows more rigid rules, gestures are often dependent on the context. For even more complexity, the slight turn of a hand or bend of a finger could throw off a traditional algorithm. Convolutional Neural Networks (CNNs) are uniquely positioned to dominate this field, with their unique ability to extract vital features in real time. This project aims to create a custom CNN for the purpose of a live gesture classification feed using a webcam.

II. Background

A. Applications

This new technology is useless if it does not have useful applications, but that fortunately is not the case. Research has been conducted on utilizing CNNs for the classification of sign language gestures, achieving an accuracy of over 99% on multiple datasets [1]. The architecture proposed by Damaneh et al. is far more complex, using similar architectures to our custom CNN multiple times in their proposal.

Another application being studied is for interacting with objects in video games. While the Faster R-CNN method can classify gestures easily, it has no way to account for time. Using the Two-Stream Faster R-CNN method, Muchtar et al. have managed to create a gesture recognizer that is 90-93% accurate when including the time domain [2].

B. State of the Art: MobileNet and ImageNet

The MobileNet CNN is a widely used network for object detection and classification tasks. Its efficient, lightweight architecture is suitable for use in a variety of scenarios, including deployment on mobile devices (hence the name). The characteristics of MobileNet that differentiate it from other networks include its use of depth-wise separable convolutions which reduce computational cost while maintaining accuracy, and the availability of the model pre-trained on the ImageNet dataset [3].

MobileNet's depthwise convolution layers provide two key optimizations: the reduction of parameters and reduction of multiply and accumulate operations (MACs). Mobilenet consists of 14 hidden CNN layers, with an input size of 224 x 244 x 3, a dense 1024 x 1000 layer, and a standard output softmax layer to classify 1000 unique classes (this will be modified for the specific gesture recognition task later). It has approximately 4.2 million parameters. In comparison, MobileNet's implementation with standard convolution layers instead of depthwise convolution layers has 29.3 million parameters and about 8.5x as many MACs (4866 vs 569) while providing less than 1%

additional accuracy (71.7% (Conv MobileNet) vs 70.6% (MobileNet) [3].

MobileNet's training on the ImageNet dataset makes it suitable for a wide range of image classification tasks due to the breadth and depth of the ImageNet dataset. Imagenet is a collection of images based on the WordNet dataset which is a compilation of over 100,000 nouns and phrases. The result is 21,000+ unique image classes with over 14 million images in total. The pre-trained version of MobileNetV2 implemented for the gesture recognition task is trained on the ILSVRC dataset - a subset ImageNet intended for competition use consisting of 1000 unique categories and over 1 million images [5]. In comparison to other high performing CNNs trained on the ImageNet dataset, MobileNet outperforms them mainly due to its high efficiency [3].

C. *ReLU CNN*

The Rectified Linear Unit (ReLU) activation function is foundational to using CNNs for image classification. It is defined as f(x)=max(0,x), meaning it is 0 for negative x and linear for positive x, as seen in Figure 1. The simplicity of this function allows for faster computation than other activation functions like sigmoid, which involve complex mathematics.

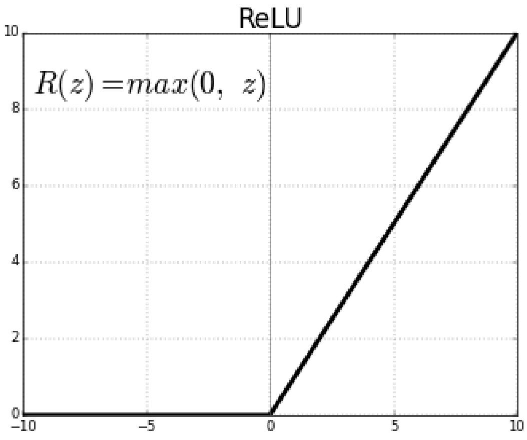


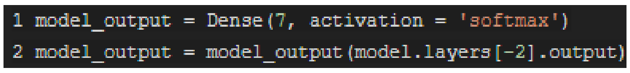
Figure 1: Graph of the ReLU activation function [4]

III. Design and Implementation

A. *MobileNet CNN*

The key underlying concept of the design of the modified, pre-trained MobileNetV2 model used to accomplish the gesture recognition task is transfer learning. Due to the readily available pre-trained MobileNetV2 model via the Keras Python package [6] it is an excellent candidate for transfer learning to accomplish new tasks. Hidden layers of the network can be frozen, and custom output layers can be appended to the model. By feeding new data to the model and training only the new, custom layers, the network can be adapted to recognize objects it was not previously trained on with a fraction of the time and resources required to train a model of this size. This process is what is known as transfer learning [7].

Transfer learning using the Keras MobileNetV2 model is implemented by creating a softmax output layer and inserting it as the last layer of the model in place of the default 1000 class output layer. This is accomplished through the following two pieces of python code:



```
1 model_output = Dense(7, activation = 'softmax')
2 model_output = model_output(model.layers[-2].output)
```

Figure 2: Python code for inserting custom output layer to Keras model

Line 1 creates the output layer, and line 2 inserts the previous layer's output as the input to the new output layer using the Python functional programming technique [7].

For the creation of the custom dataset, a custom Python script was used to capture images over a constant time interval using the laptop device's native webcam to capture 224 x 224 rgb images and label them based on their classification. 500 images of each gesture were collected resulting in 3500 images in total to be used for training and validation.

To train the model, the well-known Adam optimizer and cross entropy loss function were used. A 75/25 training-validation split was used to divide the randomly shuffled, labeled images from the dataset. A total of 50 training epochs were used in the final implementation, resulting in a training time of only ~4 minutes.

B. *ReLU CNN*

Our custom CNN was built using PyTorch in Visual Studio Code. Three convolutional layers were used with pooling, followed by three fully connected layers. This can be seen below in Figure 2. The final layer has seven output nodes, corresponding to the seven different gestures the model can classify.

Layer Name	Type	Filter Shape	Input Size
conv1	Conv2d	3x3x32	3x640x480
pool1	MaxPool2d	2x2	32x640x480
conv2	Conv2d	3x3x64	32x320x240
pool2	MaxPool2d	2x2	64x320x240
conv3	Conv2d	3x3x128	64x160x120
pool3	MaxPool2d	2x2	128x160x120
flatten	View	-	128x80x60
fc1	Linear	-	76,800
fc2	Linear	-	256
fc3	Linear	-	128

Figure 3: Table showing the layers used in the ReLU CNN

The same Python script as was used in the MobileNet CNN dataset creation was used to create this dataset, but the images collected were of size 640x480. 500 images of each of seven gestures were collected. These images display the hand and arm on a plain white background to minimize variance.

To train the model, the Adam optimizer and cross entropy loss function were used, similarly to the MobileNet CNN. The labeled images were randomly shuffled and 70/15/15 training - validation - test split was used to divide them. A total of 10 training epochs were used for a 1.5 hour training time.

In order to run the model in real time, it is saved as a .pth file after training then loaded into another Python script. This script runs the CNN on every image captured by the webcam.

IV. Results, Challenges, and Tricks

A. *MobileNet Network*

After 50 epochs, the following training and validation accuracy and loss graphs summarize the results of the model's training and validation:

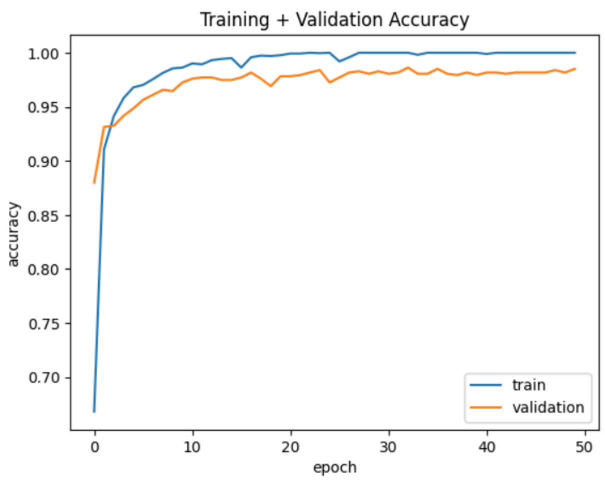


Figure 4: Training ands validation accuracy after 50 training epochs

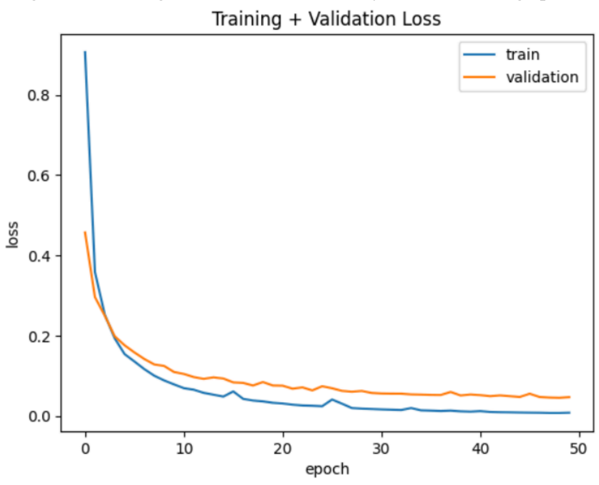


Figure 5: Training and validation loss after 50 training epochs

Training accuracy reaches 100% with validation accuracy reaching 98% on 875 images (total 857 correct predictions). Training loss approaches 0 with validation loss around 1%. While the model is able to (almost) always predict the correct images of the validation set, the model's real-time performance is not as high-performing.

Lack of real-time performance accuracy was observed, and can be attributed to multiple reasons.

The validation images are a random subset of the training images, all taken in similar lighting and background conditions.

Real-time images in different environments present a new challenge of handling the differences in lighting and background for which the model was not directly trained on.

However, the overall real-time performance of the model exceeds that of the custom created CNN model in the other approach presented in this paper.

This is because of the pre-trained MobileNet model's feature extraction capability on a wide range (1 million) of images which is transferred to the novel (to the model) hand gesture recognition task.

A number of challenges prevented a higher performing model. Computational power and lack of a robust, diverse dataset provided for a narrow scope of the model's potential performance. Training the model was done on Google Colab, a free VM-based Python Web IDE. Training consumed 12/16 GB on the model, reaching the computational limits of the machine which crashed multiple times before optimizing image processing. This training would not even have been possible on most student laptops. Due to the limitation of having to use Colab, managing the size of the data associated with 3500 images presented time and storage constraints. The images took over 2 hours to upload into Google Drive, which failed repeatedly due to storage limitations. It is apparent that neural networks demand a lot of computational power and network resources.

Despite the challenges of developing the transfer learning model, several good development practices also provided for a better development process. Using the Colab VM's free GPU access provided a free resource that is not available on most laptops. Colab also provides a free code snippet to embed image capture into the browser application which is used for demo purposes. Extensive Keras documentation made learning a new Python AI package much easier, and online tutorials accomplishing similar transfer learning and data handling tasks also streamlined the development process.

B. ReLU CNN

The following graphs were produced after ten epochs of training, showing training loss, validation loss, and validation accuracy.

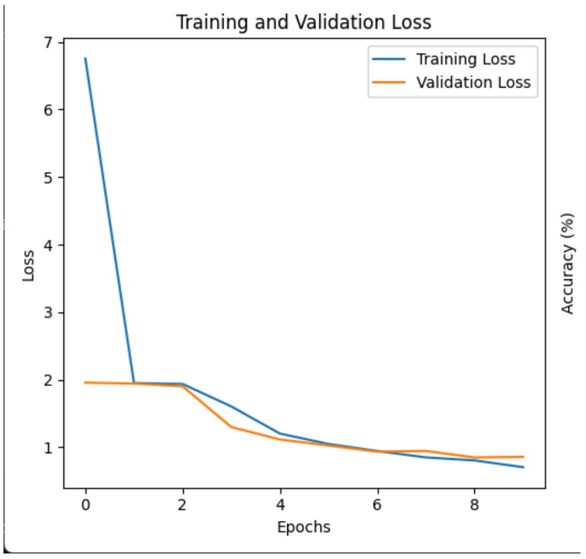


Figure 6: Training and validation loss after 10 training epochs

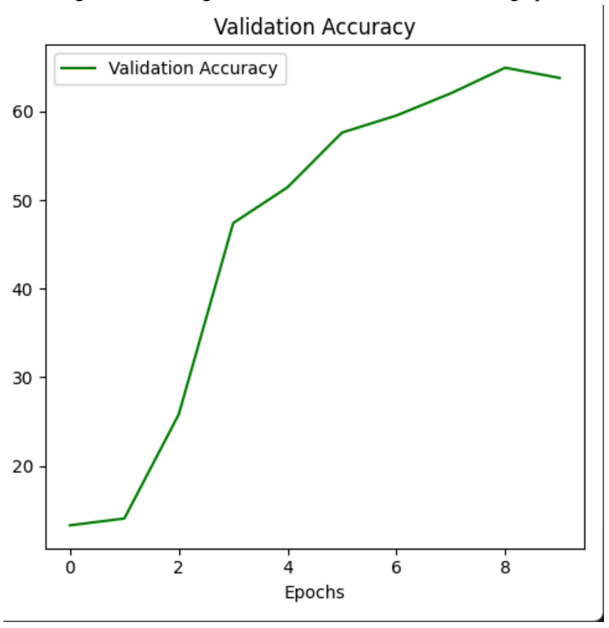


Figure 6: Validation accuracy after 10 training epochs

These charts show a peak validation accuracy of 64.9%, and uncharted is the 61% test accuracy. These numbers point to the model being relatively accurate in classifying the images, but real-time results are nowhere near as accurate. It rarely classifies the gesture correctly, as seen in Figure 7.

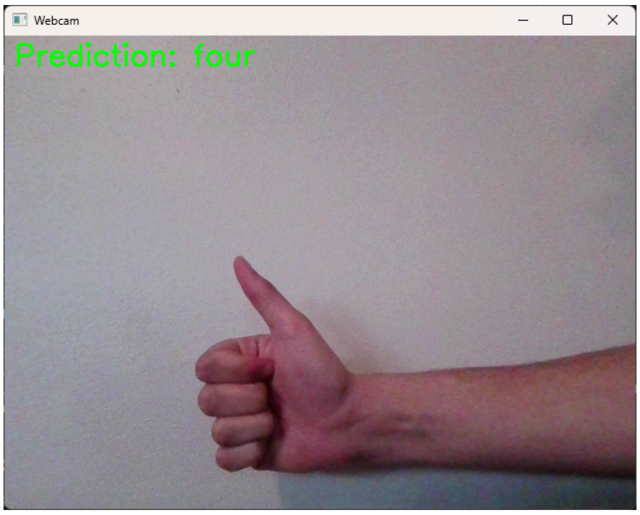


Figure 7: An inaccurate live gesture classification

There were a few challenges in creating a better model. For starters, excessive hours were spent trying to get a CNN with a Sigmoid activation function working. This led to a model that had a consistent 14% validation accuracy, or, as 1/7 shows us, random guessing. In addition, the first dataset created was not as good, attempting to show the gesture in a variety of lightings, positions and angles. Adding more preprocessing to separate the gesture from the background would help this, but this idea was never implemented. Finally, this model was trained using a 12th gen Intel i9 processor and later an NVIDIA 3070 GPU, leading to longer training times than necessary. Utilizing the computing power of Google Colab would have sped up training immensely, allowing for more extensive refining of the CNN's parameters.

IV. Conclusions/Future Work

CNNs are proven to be effective for hand gesture recognition due to their ease of implementation and tunability as well as the availability of mature CNN architectures. Transfer learning proved to be the more effective technique with better training and real-time performance results.

It is apparent that hand gesture recognition requires a more robust and common dataset which provides an opportunity for future work. As for the specific implementation of both approaches, both can be improved. For the custom-built CNN, a more complex architecture could be explored, and training could be done with a VM that has more resources than a local machine. Additional preprocessing would also be helpful, separating the gesture from the background. For the MobileNet architecture, a need for a more robust dataset would contribute to a much higher performing model. Additionally, real-time, continuous image capture and processing could be implemented which would make the overall application more deployable and easy to use.

References

[1] M. M. Damaneh, F. Mohanna, and P. Jafari, "Static hand gesture recognition in sign language based on convolutional neural network with feature extraction method using ORB descriptor and Gabor filter," *Expert Systems with Applications*, vol. 211, p. 118559, Jan. 2023, doi: https://doi.org/10.1016/j.eswa.2022.118559.

[2] R. A. Muchtar, R. Yuniarti and A. Komarudin, "Hand Gesture Recognition for Controlling Game Objects Using Two-Stream Faster Region Convolutional Neural Networks Methods," 2022 International Conference on Information Technology Research and Innovation (ICITRI), Jakarta, Indonesia, 2022, pp. 59-64, doi: 10.1109/ICITRI56423.2022.9970207. keywords: {Training;Technological innovation;Computational modeling;Symbols;Games;Gesture recognition;Streaming media;Games;Hand Gesture Recognition;Spatial and Temporal;Two-Stream Faster R-CNN;Video},

[3] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv.org, 2017. https://arxiv.org/abs/1704.04861

[4] S. SHARMA, "Activation Functions in Neural Networks," *Medium*, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[5] "ImageNet," Image-net.org, 2020. https://image-net.org/about (accessed Apr. 23, 2024).

[6] K. Team, "Keras documentation: MobileNet and MobileNetV2," keras.io. https://keras.io/api/applications/mobilenet/

[7] S. Mathôt , "Customizing MobileNetV2 through Transfer Learning," Python Tutorials, 2018. https://pythontutorials.eu/deep-learning/transfer-learning/ (accessed Apr. 23, 2024).