Fingers Touch Detection

Justin Matthews[1] and Luciano de Oliveira Neris[2]

[1]George Mason University
[2]Federal University of São Carlos

Summer Team Impact Project 2022

Abstract

Object detection is a form of machine learning that utilizes computer vision and deep learning techniques. It can be used to identify targeted objects in images and videos with precise accuracy. For detecting hand gestures, the method most commonly used is computer vision. When using computer vision, there are some factors to consider that could impact the results of the experiment. Due to this, a variety of different techniques were developed in order to mitigate the significance those factors have on the data. This paper will present one technique to use when creating a machine to detect hand gestures.

Keywords— Object Detection, Computer Vision, Deep Learning, Machine Learning

1 Introduction

Machine learning is "a field of study that gives computers the ability to learn without being explicitly programmed" [11]. The origin of machine learning begins with a psychologist named Frank Rosenblatt from Cornell University. He based the design of the machine on the human nervous system. The machine was named "perceptron" and its purpose was to recognize the letters of the alphabet [8]. As the field of machine learning grew, so did the amount of tasks that could be completed with it. For example, object detection is one of the many tasks that was further studied, tested, and deployed with the use of machine learning. Object detection is a visual recognition problem in computer vision with the goal being to find objects of certain target classes in a given image and assign each object a corresponding class label. It incorporates deep learning techniques due to the success of deep learning based image classification over the recent years [12]. Now that some of the history of machine learning is explained, let's begin to look at how it can be used to detect finger positions. Creating this machine will be beneficial as its model can be taken and implemented into different applications.

For the remainder of the paper the following will be explained: the first topic discussed will be some previous research done by others on finger detection, next the methods of how the model was created and tested will be provided, and following that will be the results of this whole process. Finally, the paper will end with the conclusion.

2 Related Works

There are many different ways that researchers can use machine learning for hand and finger detection. The most common way is with the use of computer vision. It is one of the most used methods as it provides contactless communication between humans and computers. The camera also has different configurations that can be used, such as monocular, fisheye, TOF, etc. There are downsides with this method however as it is dependent on several factors which include: lighting variation, background issues, the effect of occlusions, complex background, processing time traded against resolution and frame rate and foreground or background objects presenting the same skin color or otherwise appearing as hands [10]. An example of a way to use computer vision to create a machine that can detect different hand gestures is by taking a picture of a hand gesture and detecting the hand from the background. The palm and fingers are then segmented and based on the segmentation, the fingers are recognized. The hand gesture is then recognized by a rule classifier that is a mixture of different labeled photos of hand gestures. The only issue with this method however, is that objects that are similar in color to the skin of the hand could hinder performance [7]. Another method discovered for detecting hand gestures is using circular Hough transform. Hough transform is a feature extraction technique widely used in image analysis, computer vision, and digital image processing. It groups edge points into object candidates and extracts the edge features

of the images. After the hand image is converted to gray scale, the circular Hough transform of the image is computed. The location of the peaks of the transform are identified which correlate to the probable fingertip location of the image with the number of peaks identified correlating to the number of probable fingertips visible. The coordinates of the peaks are saved and the Hough lines are scanned to determine whether or not the line correlates to a finger. If the coordinates of the peaks align with the coordinates of the line, then those sets of coordinates are recognized as a finger. The total number of finger counts and their coordinates are provided as the output for this algorithm. This algorithm was tested with a database that contained 100 hand images and had an accuracy of 95 percent [6]. This paper will be discussing the process of creating a machine to detect hand gestures using computer vision, with the only difference being the technique used.

3 Materials and Methods

The goal of this project was to make a program that can detect hand gestures, specifically the hand fingering the numbers zero to five, using machine learning. To do this, there are a couple of programs required. This section will describe the functionality of each of the tools used over the course of this project.

3.1 Tools

Visual Studio Code (VS Code): The integrated development environment (IDE) that provides developers with the tools that they need for a quick code-build-debug cycle [1]. For this project, VS Code served as a platform that allowed for easy editing of the code in certain files.

Figure 1: Visual Studio Code Logo

Source: https://code.visualstudio.com/opengraphimg/opengraph-blog.png

Anaconda Prompt: A program that is very similar to a terminal or command prompt (cmd) [3]. Simply put, it is a black screen used to type in commands by the user. This project used Anaconda Prompt as a way to run the Python scripts that were created.

Figure 2: Screenshot of a new Anaconda Prompt window

Tensorflow: An open-source (free) platform for machine learning [4]. From Tensorflow, the application programming interface (API), Keras, was used to save and load the data of the model that was created and trained.

Figure 3: Tensorflow Logo

Source: https://www.tensorflow.org/extras/tensorflow_brand_guidelines.pdf

Python: A high-level, interpreted, object-oriented programming language [5]. Python was the programming language used thorughout the entire project.

Figure 4: Python Logo

Source: https://www.python.org/community/logos/

OpenCV:An open source computer vision and machine learning software library [2]. With OpenCV, the webcam was accessed and streamed a real time video in a new window.

Figure 5: OpenCV Logo

Source: https://opencv.org/resources/media-kit/

3.2 Setup

Before any coding was started, research on the different architectures and frameworks of machine learning was conducted. A table was constructed that listed the advantages and disadvantages of each tool. The selection of the architectures and frameworks for the project was dependant not only on the advantages and disadvantages list, but also by the following three criteria: how simple it is to install on a windows machine, how easy it is to learn for beginners, and how well it integrates with other required tools needed for machine learning. A list of the different dependencies was also created which simply consisted of the name and definition of each dependency. After the research concluded, the tools that were chosen were Pytorch (this was later changed to Tensorflow and Keras but the reason will be discussed in a later section), Python, and OpenCV. Pytorch was chosen over frameworks like Tensorflow for two reasons. The first reason was because it has windows support, unlike Tensorflow, and the second reason is because it is easy to code with it using Python.

Frameworks	Pros	Cons
Tensorflow	• Open source platform • Data visualization – Better visualization of data using a graphical approach – Easy debugging of nodes using TensorBoard • Keras friendly – Allows for some high-level functionality sections in the code – Keras provides system-specific functionality such as pipelining, estimators, and eager execution • Scalable • Compatible – Compatible with many languages like C++, Javascript, Python, C#, Ruby, and Swift • Parallelism – User can choose to run its code on either of the architecture based on the modeling rule (GPU if not specified) – Uses different distribution strategies in GPU and CPU systems • Graphical Support – Allows building neural networks with the help of graphs that represent operations as nodes. – Acts in multiple domains such as image recognition, voice detection, motion detection, time series, etc • Training is faster • Well documented	• Frequent updates – Updates every 2-3 months – Updates increases the overhead for a user to install it and bind it with existing system • Inconsistent – Provides homonyms that share similar names but different implementations ∗ Ex: tf.nn.conv2d, tf.nn.convolution, tf.layers.conv2d all have varying meanings, which makes it inconsistent with its usability • Architectural limitation – TPU (Tensor Processing Unit) only allows the execution of a model not to train it • Dependency – Reduces the length of code and makes it easier for a user to access, however, every code needs to be executed using any platform for its support which increases the dependency for the execution • Symbolic loops – Lags at providing the symbolic loops for indefinite sequences – Referred to as a low-level API • Slow speed – Low speed with respect to its competitors – Less usability in comparison to other frameworks • Support for Windows – Does not provide much features for the Windows Operating System users – Wide range of features for the Linux users – Windows users can download TensorFlow using the anaconda prompt or using the pip packages • Steep learning curve

Keras	• User-friendly and Fast Deployment – Very easy to create neural network models – Good for implementing deep learning algorithms and natural language processing – Good collection of Keras functions to do data processing – Provides multiple layers including support for Convolution and Recurrent Layers • Quality Documentation and Large Community Support – One of the best documentations ∗ Introduces each function in a very organized and sequential way – Great community support ∗ Lots of community codes on various open-source platforms ∗ Many developers and Data Science enthusiasts prefer Keras for competing in Data Science challenges ∗ Many researchers publish their code and tutorials to the general public • Multiple Backend and Modularity – Provides multiple backend support ∗ Tensorflow, Theano, and CNTK are the most common backends – Can train the Keras model on one backend and test its results on another • Pretrained models – Proved some deep learning models with their pre-trained weights ∗ Can use those models directly for making predictions or feature extraction – Available models: ∗ Xception ∗ VGG16 ∗ VGG19 ∗ ResNet, ResNetV2 ∗ Inception V3 ∗ InceptionResNetV2 ∗ MobileNet ∗ MobileNetV2 ∗ DenseNet ∗ NASNet • Multiple GPU Support – Allows users to train their models on a single GPU or use multiple GPUs – Built-in support for data parallelism – Can process a very large amount of data • One of the fasted growing libraries for deep	• Problems in low-level API – Possibility to get low-level backend errors continuously ∗ Errors occur because an operation was performed that Keras was not designed for – Does not allow to modify much of its backend – Error logs are difficult to debug • Need improvement in some features – Data-preprocessing tools are not that much satisfying when compared to other packages like scikit-learn – Not so good to build some basic machine learning algorithms like clustering and PCM (principal component analysis) – Does not have features of dynamic chart creation • Slower than its backend – Sometimes it is slow on GPU and takes longer time in computation compared with its backends ∗ Sacrifice speed for its user-friendliness

Pytorch	• Easy to learn
	  – Same structure as traditional programming
	  – Brilliantly documented with the developer community continuously working to improve
	  – Easy to learn for the programmer and non-programmer
	• Developers Productivity
	  – It has an interface with python and with different powerful APIs
	  – Can be implemented in Windows or Linux OS
	  – Knowledge developers can improve their productivity as most of the tasks from PyTorch can be automated
	• Easy to debug
	  – Can use debugging tools like pdb and ipdb tools of python
	  – Develops a computational graph at runtime, so programmers can use Pythons IDE PyCharm for debugging
	• Data parallelism
	  – Can distribute computational tasks among multiple CPUs or GPU
	  * Possible by using the data parallelism feature (torch.nn.DataParallel)
	  * Wraps any module and helps do parallel processing
	• Useful Libraries
	  – Large community of developers and researchers who built tools and libraries to extend PyTorch
	  – Community helps in developing computer vision, reinforcement learning, and NLP for research and production purposes.
	  – Popular libraries:
	  * PyTorch
	  * BoTorch
	  * Allen NLP
	• Supports cloud platforms
	• Supports Python and C++
		• Was released in 2016, so it is new compared to others and has fewer users and is not widely known
		• Absence of monitoring and visualization tools like a tensorboard
		• Developer community is small compared to other frameworks
		• Lacks model serving in production (possibility to change in the future)
		• Not as extensive as TensorFlow
		  – The development of actual applications requires conversion of the PyTorch code into another framework such as Caffe2 to deploy applications to servers, workstations, and mobile devices

Caffe	• Open sourced (free) • Library is targeted at developers who want to experience deep learning first hand • Offers resources that promise to be expanded as the community develops • One doesn't need a steep learning part and can start exploring deep learning using the existing mode right away • Do not need to know code to use Caffe • Good to do computer vision tasks • Chose Python for its API • Good for feedfoward networks and image processing • Good for fine tuning existing networks • Train models without any code • Python interface is pretty useful • Widely supported libraries for CNNs and computer vision	• Steep learning curve • Doesn't fare well with recurrent neural networks and sequence modeling • Not intended for other deep learning applications such as text, sound, or time series data • Need to write C++/CUDA for new GPU layers • Not good for recurrent networks, or generative adversarial network • Cumbersome for big networks (GoogLeNet, ResNet) • Not extensible • No commercial support • Slow development

Table 1: Framework Pros and Cons

The next step taken was installing the necessary equipment. The first tool installed was Python (for this project Python 3.10.5 was used). Next, the app "Windows Terminal" was downloaded from the Microsoft Store, which was used to access the Command Prompt (cmd). The reason this app was chosen over just normally opening the cmd was because the app allows you to have multiple tabs open at once. This means that a person can work with two separate directories at the same time easily.Though this application would get replaced by a later one, it was used initially to install pip and check to see if the installation of Python and pip was successful. After "Windows Terminal" was installed, the pip package was installed. The pip package allows users to install other dependencies with ease and was necessary for the development of this project. Next, was installing an application called "Anaconda Prompt". This program is the one that will now replace cmd. Anaconda Prompt is essentially cmd, but it allows users to create environments to run their programs. It also organizes it so that any dependencies or imports installed to the environment are only available to that certain environment. Finally, Pytorch was installed using a command provided by the website. Once Pytorch was installed, a new environment was created and the coding of the project began.

All Python scripts were created and edited with the use of Visual Studio Code and ran using Anaconda Prompt. In total, there are five Python scripts that make up the entirety of this project. The first script is called "buildDataset.py". As the name suggests, this code will build your image dataset for training the machine learning model. To run the script, the user will have to type python buildDataset.py *folder name to store images* *number of images* into the Anaconda Prompt. The code then accesses the computer's webcam and puts the videostream in a new window. By pressing the 'a' key on the keyboard, screenshots of the video stream will begin to be taken. A new folder will be created into a directory specified in the script and the images will be stored in that folder. The screenshot process can be paused by pressing the 'a' key again and if the user wants to quit early, they can press the 'q' key. The next script to run is titled "newNeuralNetwork.py". This script is what sets up, trains, and saves the model. Originally, training was done on the CPU (central processing unit) which had a runtime of approximately 50 minutes. It was decided that this runtime was too long and the CUDA GPU (graphics processing unit) was installed in order to help lower the runtime during the training process. The CUDA GPU is a popular GPU programming model introduced by NVIDIA for parallel computing [9]. With CUDA, the runtime went from 50 minutes to 8 minutes. The next two scripts are optional to run. The first one is named "getConfusionMatrix&Accuracy.py" and as the name suggests, when run it will print out the confusion matrix of the model and the overall accuracy. The second one is called "test.py" which will use the saved model to predict the label of a given image. In order to successfully run this script, the user will have to type python test.py *image directory path*. The last Python script of the project is called "detection_V2". This script accesses the computer's webcam and displays the videostream in a new window. A rectangle is drawn in the videostream along with text. Depending on the hand gesture that is held in the rectangle, the model will make a prediction which will change the text. The options for hand gestures are the following: "Zero", "One", "Two", "Three", "Four", and "Five" (This can be seen in Figure 6).

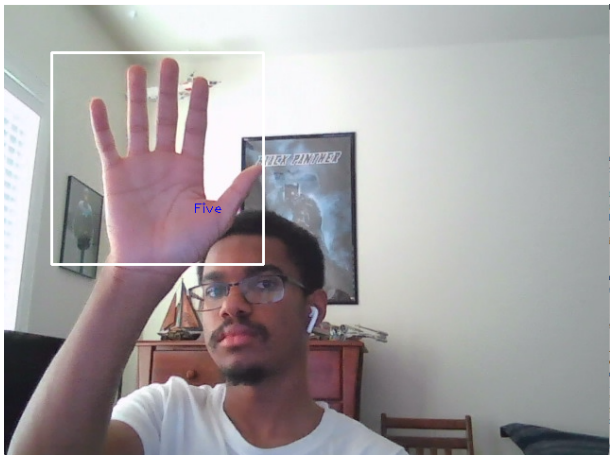

Figure 6: Example of the Detection

4 Results

This section will show the results of a run-through of all of my python scripts, except buildDataset.py, and what the expected output would look like. For this run there are six folders (0-5 and named respectively) and each contain 500 images of specific hand gestures. A total of ten epochs were conducted and each took roughly 40 seconds with a batch size of 32.

4.1 Trained Model Results

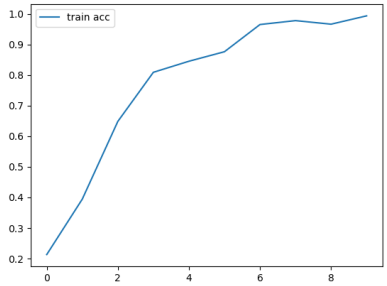

Figure 7: Accuracy of model during training

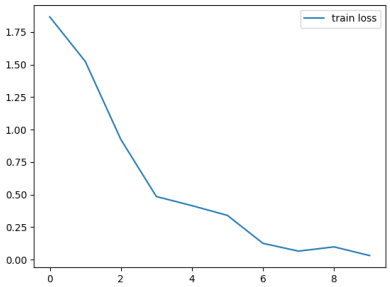

Figure 8: Data loss during training

For Figures 6, the x-axis is the epoch, which is a pass of the training dataset through the algorithm, and the y-axis is the accuracy during that epoch. For Figure 7, the x-axis is also the epoch while the y-axis is the data lost during that epoch.

4.2 Inference Results

```
[[ 66   0   0   0   0   0]
 [  0 611   0   0   0   0]
 [  0   0 118   0   0   0]
 [  0   0   0  89   0   0]
 [  0   0   0   0 324   0]
 [  0   0   0   0   0 342]]
              precision    recall  f1-score   support

        Five       1.00      1.00      1.00        66
        Four       1.00      1.00      1.00       611
         One       1.00      1.00      1.00       118
       Three       1.00      1.00      1.00        89
         Two       1.00      1.00      1.00       324
        Zero       1.00      1.00      1.00       342

    accuracy                           1.00      1550
   macro avg       1.00      1.00      1.00      1550
weighted avg       1.00      1.00      1.00      1550
```

Figure 9: Confusion Matrix and accuracy of the model

When printing Figure 8, the runtime was approximately 4:20 minutes. Note that this runtime will increase or decrease depending on the amount of images in the dataset.

```
1/1 [==============================] - 3s 3s/step
Predicted: Four
```

Figure 10: Prediction from test.py

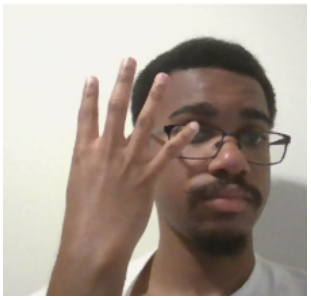

Figure 11: Image used for test.py

5 Conclusion

There were some challenges that arose when building this project. The first major issue that was faced was when OpenCV couldn't access the laptop webcam. Originally, this project was going to be done with an Ubuntu virtual machine in the terminal. However, when trying to access the webcam using OpenCV, error messages kept appearing. That was because the /dev/video0 file was missing. There were many attempts to resolve this error such as using the built in Ubuntu command "cheese" and researching how to create and/or install the /dev/video0 file. Eventually, it was decided to switch to Windows Command Prompt where the webcam was then able to be accessed. The next major issue came up when training the machine learning model. As stated early, when the training was done on the CPU, the runtime was approximately 50 minutes. The location that the training takes place was then swapped from the CPU to the CUDA GPU. However, when that switch happened, Pytorch was unable to connect to the GPU in the working environment. To try and resolve this problem, Pytorch was uninstalled and reinstalled, downgraded, and upgraded, but none of these methods worked. Tensorflow was then installed and was successfully able to connect to the GPU. The last major issue that was encountered happened when testing the final product. When displaying the hand in the region of interest (ROI), the model kept making the wrong predictions. It was discovered that this issue was happening due to there not being enough images of the hand gesture in different positions and angles.After overcoming all of the obstacles during this project, the goal was met. A program that can detect hand gestures, specifically the hand fingering the numbers zero to five, using machine learning was created and functioning.

References

[1] Visual studio code FAQ, 2021. Accessed August-03-2022.

[2] About, n.d. Accessed August-03-2022.

[3] How to get the anaconda prompt on macos, n.d. Accessed August-03-2022.

[4] Tensorflow, n.d. Accessed August-03-2022.

[5] What is python? executive summary, n.d. Accessed August-03-2022.

[6] A. Biswas. Finger detection for hand gesture recognition using circular hour transform. In *Advances in Communication, Devices and Networking*, pages 651–660, 2018.

[7] Z. Chen, J.-T. Kim, J. Liang, J. Zhang, and Y.-B. Yuan. Real-time hand gesture recognition using finger segmentation. *The Scientific World Journal*, 2014.

[8] A.L. Fradkov. Early history of machine learning. *IFAC-PapersOnLine*, 55(13):1385–1390, 2022.

[9] T. Kalaiselvi, P. Sriramakrishnan, and K. Somasundaram. Survey of using gpu cuda programming model in medical image analysis. *Informatics in Medicine Unlocked*, 9:133–144, 2017.

[10] M. Oudah, A. Al-Naji, and J. Chahl. Hand gesture recognition based on computer vision: A review of techniques. *Journal of Imaging*, 6(8), 2020.

[11] U.S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley. A brief survey of machine learning methods and their sensors and iot applications. IEEE, 2017.

[12] Wu, Sahoo, and Hoi. Recent advances in deep learning for object detection. arXiv, 2019.