Hand Gesture Recognition Using Artificial Intelligence

Talal Siddiqui, Visheshank Mishra
Northeastern University
360 Huntington Ave, Boston, MA 021153
siddiqui.t@northeastern.edu, mishra.vis@northeastern.edu
https://github.com/talals1/cs5100-final-project

Abstract

For our final project in CS5100: Foundations of Artificial Intelligence, we chose to research and implement a system that can recognize hand gestures via a webcam. To do this, we utilized several techniques in artificial intelligence and machine learning such as various classifiers, image segmentation, background subtraction, and more. Our code is located at https://github.com/talals1/cs5100-final-project.

Introduction

Humans have been interacting with technology in a myriad of ways over the past century. From levers, switches, buttons, and now touch displays, the way we interact has been a constant focus in how we design the machines we use in our everyday lives. Currently, it seems like the next evolution of human-computer interaction is focused on interacting with computers without physical touch, such as talking or gesturing instead of manually clicking a button or turning a switch. There are many products already used by many that emphasize wireless control, such as voice assistants like Siri by Apple or Alexa by Amazon. As well, many have begun to simple systems of gesture control, but many of these systems are just marketing gimmicks with no real value.

For our project, we attempted to figure out the ways we could utilize gestures in a way that was actually useful and easy to use. To do this, we trained several types of classifier models using Python, Scikit-Learn, and several other libraries. Then, we created a program that uses the video feed from your webcam to detect your hand gesture and then perform appropriate behavior based on it.

Background

In order to be able to make a computer understand that a certain type of image corresponds to a certain value, we had to utilize things called *classifiers*. Classifiers can be defined as mathematical models or algorithms that can be used to classify a certain input to a certain output. There are several types of classifiers in the world of artificial intelligence (AI), and the ones we used are called *K-Nearest Neighors*, *LinearSVC*, and the *Random Forest Classifier*. Classifiers belong to a subgroup of machine learning called *supervised learning*, which is where you have data that has labels assigned to it and the model trains itself by learning how certain samples correspond to certain labels.

For gesture recognition, the that a system will be predicting off of is going to be a person in a real life situation. This means that though the user's hand will be in the image, there will also be a lot of other unimportant things in both the foreground and background of the image. This includes things like the body of the user, lights, their room, and basically anything else in the room. The hand is just a small part of the picture, and finding the hand can be a tricky adventure. In order to extract out certain parts or features of an image, researchers use a technique called *image segmentation*. Image segmentation is where you apply filters and masks to an image in order to highlight specific objects present in the image. For example, you could use image segmentation in order to highlight all the humans in a picture.

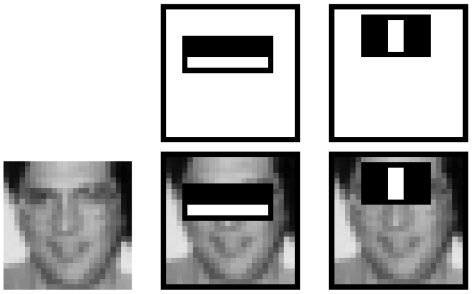
In our case, we needed to use image segmentation in order to focus on the hand in the image. What we hand to do was to find a specific way to segment each frame from the user's webcam so that only the gesturing hand was the only thing being shown in the image. Along with that, we also played around with *background subtraction*, which is where after you segment an image, you get rid of everything else in the image in order to end up with the piece of the image necessary to make predictions from.

As you can see, training a model was the easy part for us. The main challenge lay in figuring out the right combinations of filters and masks that would transform the input from the camera into an acceptable piece of data that our classifier model could use to make a prediction.

Related Work

There has been a lot of work done in image and gesture recognition over these past decades. The early days of facial recognition could probably be thought of as the first generation of gesture recognition since a lot of systems were designed to guess what feeling is being portrayed on a particular face.

One of the earliest widespread uses of object recognition in live video was pioneered by Paul Viola and Michael Jones who created the Viola-Jones object recognition.



Above is a diagram of the Viola-Jones method, taken from their seminal paper ”Robust Real-time Object Detection”. The VJ method utilizes something called ”features”, which they describe as patterns they can use to detect whether or not a face is present in a particular area of a image. In the diagram above, we can see the features specific to a nose and to a upper cheek are being used to detect the mans face. This system was very widely used in cameras to detect faces for many years. Our approach is similar to this method as we also utilize a black and white representation of our image to identity an object (a hand gesture in our case). But unlike the VJ method, we do not scan over the entire image to detect our object since it seemed a bit complex for us.

The paper ”A review of hand gesture and sign language recognition techniques” by Cheok, Omar, and Jawad goes over many techniques employed by various papers in detecting hand gestures. We particularly were interested in Cheok et al’s description of using Hidden Markov Models (HMMs) in detecting hand gestures. But, we did not have much time to fully dive into HMMs and how they would be implemented to detect hand gestures for our project. Using HMMs may be an interesting way in detecting hand gestures, but the paper listed various ways that HMMs limitations cause its accuracy with hand gestures to hover at 90% and below.

Data Set Description

We first began by finding a suitable dataset to use. There really isn’t a whole lot of hand gesture datasets on the internet but we did first 2 on Kaggle that proved useful to us. The first one by someone named Arya R., which we term the ”Simple Hand Set”, was a collection of 50 by 50 pixel images that was quite simple since it did not have much data and because the images in the dataset were simple white and black pixels (no in-between values).

The second dataset, which we term the ”Complex Hand Set”, is a collection of 150 by 150 pixel images that was taken via a Leap Motion camera by an group called ”Gti”. The complex hand set images were much more detailed and lifelike, which the simple hand dataset had images similar to pixel art since they were quite low resolution. In the end, we decided on using the complex data set since the models trained using the simple data set did not have a high accuracy when predicting off of the frames from our webcam, but more on that later.

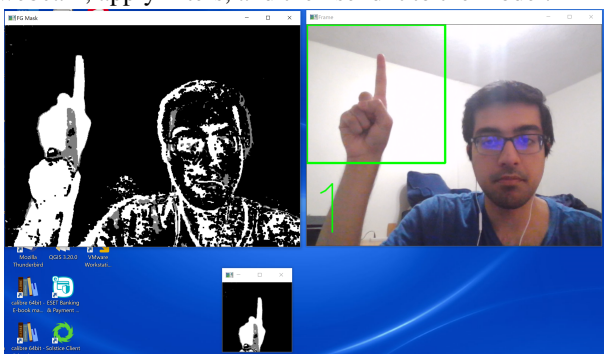
The data set has the following gestures:

- 1 = Palm
- 2 = L (right angle using index and thumb)
- 3 = Fist
- 4 = Fist at an angle
- 5 = Thumb
- 6 = Index finger
- 7 = OK sign
- 8 = Palm at an angle
- 9 = C (made using rounded index and thumb)
- 10 = Fingers pointing down

Approach

With out data sets in hand, we utilized Scikit-Learn’s build in models for our classifiers. We used premade models instead of coding our own from scratch since the premade one are already very optimized, which means that nothing we could’ve made would’ve matched the performance of Sklearn’s models.

After we had trained a few models using the different classifiers previously mentioned, we put them to the test by making them predict live images from a webcam. To do this, we utilized the *OpenCV* library in order to capture image from a webcam, apply filters, and then send it to the model.

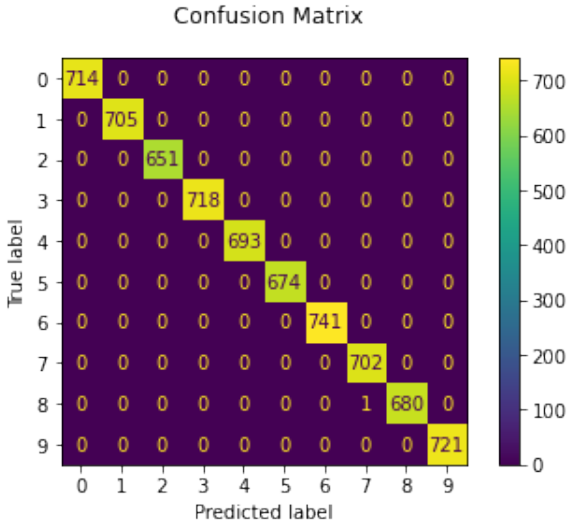


Above is an image of the system in action. On the left is the full frame from the webcam after we processed it using a OpenCV’s built-in image segmentation mask. The window on the right is what the user sees. The green box is there to tell the user where to put their hand gesture. We had considered making it so that the user could gesture anywhere in the image, but that proved to be too difficult since segmenting the image would run into too many artifacts and noise. Under the green box is a green number representing the models prediction. As you can see, the model is able to correctly predict that the user is gesture the number 1 to the camera.

Underneath the big 2 windows lies a small window in the center. The small window is what will be sent to the model for prediction. Its basically a cut out of the green square from the top right window. Before the frame is sent to the model, we flatten the matrix so it it becomes one contiguous array of data.

Experiment and Results

After training the model, the results seemed to be a bit confusing. When we had trained the model, we kept getting 100% accuracy on the our classification matrix. As well, our confusion matrices seemed really good, as in we had little to none mis-classifications.



Because of this, we decided to experiment with several different classifiers to see whether we weren't using the data and the models correctly or if we just had really good luck. We were afraid of a phenomenon called *overfitting*, which is where a machine-learning system becomes so familiar with a given data set that it is 100% accurate on the test and train sets. Overfitting is NOT a good thing since it is a signal that the model isn't really generalizing the features it detects in the data set and that it would not fair particularly well in the real world. Unfortunately, we did encounter something similar to that.

```
Saving model...
Testing model...
Accuracy: 1.000
Classification report for classifier RandomForestClassifier():
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       714
           1       1.00      1.00      1.00       705
           2       1.00      1.00      1.00       651
           3       1.00      1.00      1.00       718
           4       1.00      1.00      1.00       693
           5       1.00      1.00      1.00       674
           6       1.00      1.00      1.00       741
           7       1.00      1.00      1.00       702
           8       1.00      1.00      1.00       681
           9       1.00      1.00      1.00       721

    accuracy                           1.00      7000
   macro avg       1.00      1.00      1.00      7000
weighted avg       1.00      1.00      1.00      7000
```

As you can see above, the model seems to have 100% accuracy after being tested on the testing data. However, when the model was utilized using live webcam feed, it performed quite poorly. It often would mis-classify gestures and would generally have a hard time getting its bearings straight. Since this was on real world data from our webcam, we were not able to collect much data about this. It would simplify mis-classify whenever we showed a gesture in the green box in the frame.

```
Testing model...
Accuracy: 0.999
Classification report for classifier KNeighborsClassifier(n_neighbors=4):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       719
           1       1.00      1.00      1.00       711
           2       1.00      1.00      1.00       677
           3       1.00      1.00      1.00       709
           4       1.00      1.00      1.00       714
           5       1.00      1.00      1.00       686
           6       1.00      1.00      1.00       724
           7       1.00      1.00      1.00       671
           8       1.00      1.00      1.00       699
           9       1.00      1.00      1.00       690

    accuracy                           1.00      7000
   macro avg       1.00      1.00      1.00      7000
weighted avg       1.00      1.00      1.00      7000
```

Above is the classification report for the K-Nearest Neighbors classifier. It too scored an almost perfect score, and had the same confusion matrix.

We believe that this sharp difference between the models performance in the testing data and via the webcam is mainly due to the way we had performed image segmentation and background subtraction. These 2 techniques are very hard to get right, and when you have a poor webcam, things do not get much easier. We often encountered a lot of noise in our images and had to spend a lot of time in making sure that only the designated hand region itself was extracted and sent to the model.

Overall the best gestures it could predict from the camera were 0, 1, and 2. Everything else wasn't quite impressive since they kept getting mis-classified by the model.

Conclusion

In conclusion, we believe we have created a foundation to a promising system. Though our models may have been overfitted, we believe that if we had better image segmentation, our model could have worked much better.

Throughout this project, we learned so much about how much work really goes into image classification. Originally, we had assumed that image classification was just loading in images and throwing them at a neural network and calling it a day. But in reality, image classification has many parts that must work perfectly in conjunction with one another in order to get the best prediction from the model. As well, we learned that there is a long history of image classification, which we had assumed was a more recent phenomenon due to the perceived notion that this is the only time in history where computers were able to do anything insanely complex like identifying parts of an image.

We plan to look further into this project by enumerating all the possible ways that we could improve and fixing the parts that did not go exactly to plan. Hopefully, we plan to complete our vision of making a *useful* hand gesture recognition system which could be used by anyone with a webcam.

Future Scope

Deep neural networks can be leveraged for our classification tasks, given enough training data these models can perform exceptionally well. Some of the models are AlexNet, VGG, ResNet, SqueezeNet, DenseNet Inception, GoogLeNet, ShuffleNet, MobileNet, MobileNet, ResNeXt, Wide ResNet, MNASNet, EfficientNet, RegNet, VisionTransformer, ConvNeXt. All of these have their own pros and cons, specifically in terms of memory, compute time, and performance.

Depending on the above-mentioned constraints, a suitable neural network can be trained for our classification task, upon which real-time classifications can be done in a certain fixed time interval, the output of this can be fed to another parallelly running process which can interpret the actions/operations to be performed via system calls of an OS. An example of this can be, A sequence of pre-defined hand gestures can activate volume control, followed by hand gestures depicting the a value of 0, 1, 2 to 10, which can correspond to the volume levels of 0%, 10%, 20% to 100%.

References

Arya, R. (2021, September 4). Hand gesture recognition dataset. Kaggle. Retrieved from https://www.kaggle.com/aryarishabh/hand-gesture-recognition-dataset

Gti. (2018, July 30). Hand gesture recognition database. Kaggle. Retrieved n.d., from https://www.kaggle.com/datasets/gti-upm/leapgestrecog

cheok, ming jin, omar, zaid, Jaward, M. H. (2017, July 31). A review of hand gesture and sign language recognition techniques. Retrieved n.d., from https://www.researchgate.net/profile/Zaid-Omar-2/publication/318989820_A_review_of_hand_gesture_and_sign_language_recognition_techniques/links/59d5be76aca2725954c680de/A-review-of-hand-gesture-and-sign-language-recognition-techniques.pdf

Fang, Y., Wang, K., Cheng, J., Lu, H. (2007). A real-time hand gesture recognition method - IA. Retrieved from http://www.nlpr.ia.ac.cn/2007papers/gjhy/gh45.pdf

FastAI. (2022, January 20). Using the Learning Rate Finder (beginner). walkwithfastai. Retrieved n.d., from https://walkwithfastai.com/lr_finder

Gogul Ilango. (2017, April 25). Hand gesture recognition using python and opencv - part 2. Gogul Ilango. Retrieved n.d., from https://gogul.dev/software/hand-gesture-recognition-p2

Gogul Ilango. (2017, April 6). Hand gesture recognition using python and opencv - part 1. Gogul Ilango. Retrieved n.d., from https://gogul.dev/software/hand-gesture-recognition-p1

Huang1, H., Chong2, Y., Nie2, C., Pan2, S. (2019, June 1). IOPscience. Journal of Physics: Conference Series. Retrieved n.d., from https://iopscience.iop.org/article/10.1088/1742-6596/1213/2/022001

Kostas, M. (2019, February 19). Understanding FASTAI's fit_one_cycle method. IconOf.com. Retrieved n.d., from https://iconof.com/1cycle-learning-rate-policy/

Nausheen. (2021, February 11). Compute Performance Metrics-F1 score, precision, accuracy for CNN in fastai. Medium. Retrieved from https://towardsdev.com/compute-performance-metrics-f1-score-precision-accuracy-for-cnn-in-fastai-959d86b6f8ad

Synced. (2017, April 29). How to train a very large and deep model on one GPU? Medium. Retrieved n.d., from https://medium.com/syncedreview/how-to-train-a-very-large-and-deep-model-on-one-gpu-7b7edfe2d072

Viola, P., Jones, M. (2001). Rapid object detection using a boosted cascade of Simple features. CiteSeerX. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.6807

Vision.transform. fastai. (2021, January 5). Retrieved n.d., from https://fastai1.fast.ai/vision.transform.html