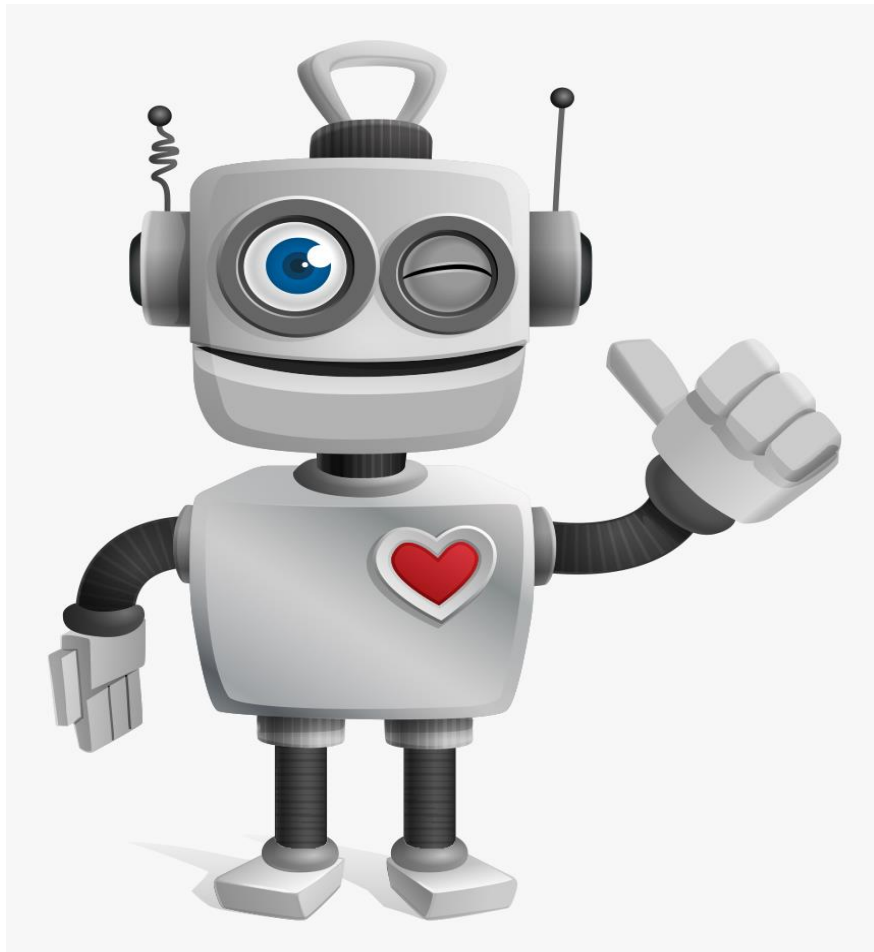


Hand Gesture Detection with Convolutional Neural Networks:

A New Mode of Human-Machine Interaction

By: Sam Elalouf, Roshni Gehani, Sneha Jadhav, Priti Thakur



Contents

Introduction	3
Problem Identification	5
Data Sources	9
Gesture Selection	10
Data Cleaning and Augmentation	11
Architecture Outline	15
Analytics & Model Selection	17
ETL	22
Webcam to Spotify Pipeline	27
Summary & Conclusion	28
References	28

Introduction

Image identification is becoming a crucial step in most modern world problem-solving systems. Categorization methods are used in image analysis and computer vision. Images transcend language, making them crucial for more global forms of communication.

In image analysis, we reduce images to their key components which can include tasks such as finding objects, detecting edges and boundaries, removing noise or distortion, and segmentation.

Image analysis is a broad term that covers a range of techniques that generally fit into these subcategories:

- ❖ **Image Enhancement:** Preparing images for display or analysis.
- ❖ **Image Segmentation:** Isolating regions and objects of interest.
- ❖ **Noise Removal:** Using morphological filtering or deep learning.
- ❖ **Image Augmentation:** Converting images to silhouettes, landmarks, etc.

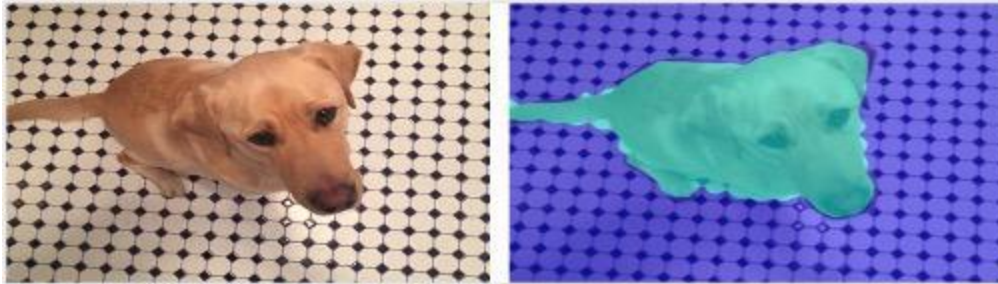
Image Enhancement:

Images can be enhanced by removing haze



Image Segmentation:

Images can also be segmented and color coordinated to identify objects such as a dog.



Noise Removal:

Noise can be removed from images as well, making them clearer.



Image Augmentation:

Augmentation involves converting images to silhouettes, landmarks, etc. for analysis.

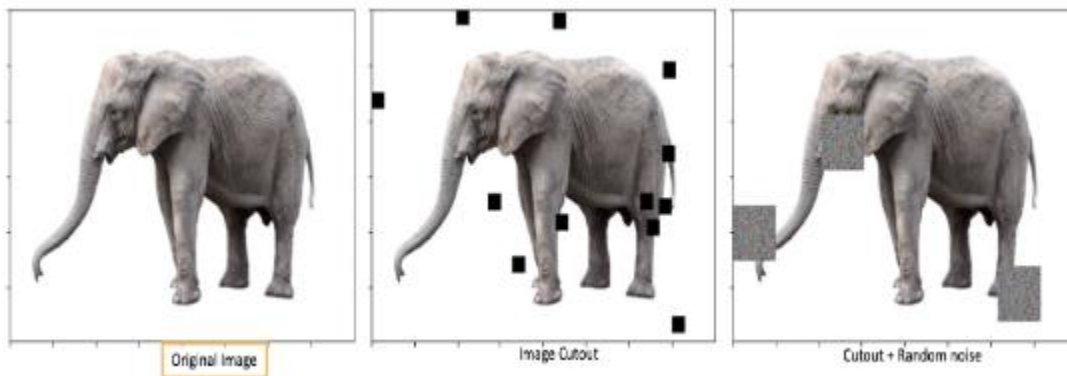


Image analysis or imagery analysis is when we analyze meaningful information in images. Digital images can be analyzed as a composition of pixels framed using RGB values.

Here's the VP of research at Gartner on the importance of image analytics:

“We do expect multimedia posts to become the predominant type of post on social media. Even the text that accompanies those posts is getting shorter and shorter... It becomes increasingly important for companies to be able to understand what's going on in those images.”

– Jenny Sussin, VP of Research at Gartner

Problem Identification

Gestures are a crucial part of communication, and for some individuals gesture is a crucial part of language itself in the form of sign language.

What is Gesture Recognition?

Gesture recognition is technology that uses sensors or cameras to read and interpret hand movements as commands.

Considering the Example of BMW:

In the automotive industry, this technology can allow one to interact with the vehicle — usually to control the infotainment system — without touching any buttons or screens.

The first gesture recognition system for automotive applications, introduced in the BMW 7 Series in 2015, was developed by Aptiv and can recognize hand gestures that control music/audio and incoming calls, etc.

With the rise of augmented reality and virtual reality, and the expansion of human-computer interaction, we will see more and more need for gesture recognition systems for everything from cars to music players.

How does it work?

The following gives an in depth description of how the BMW technology works:

A gesture recognition system starts with a camera pointed at a specific three-dimensional zone within the vehicle, capturing frame-by-frame images of hand positions and motions. This camera is typically mounted in the roof module or other vantage point that is unlikely to be obstructed. The system illuminates the area with infrared LEDs or lasers for a clear image even when there is not much natural light.

Those images are analyzed in real time by computer vision and machine learning technologies, which translate the hand motions into commands, based on a predetermined library of signs or gestures.

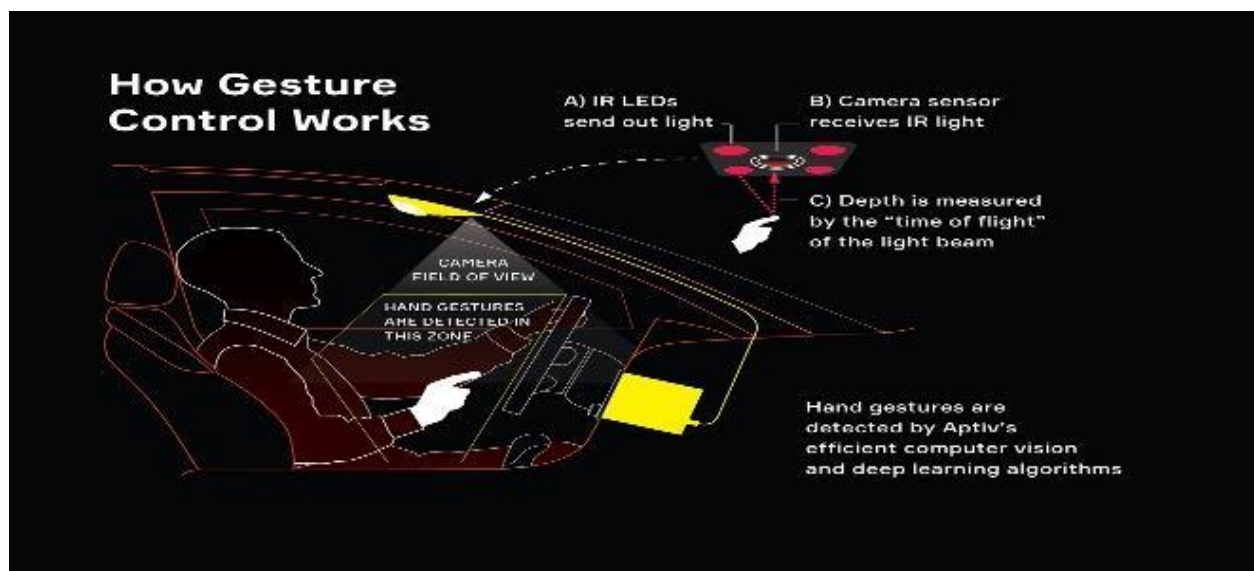
Commands generated by the gesture recognition software become just another type of input, similar to turning a dial, pressing a button or touching a screen. Additionally, as the quantity and quality of cabin cameras improves, other passengers in the vehicle could eventually get in on the act.



The bigger picture

Gesture recognition systems can improve safety as well as comfort “since drivers do not have to take their attention off the road as much as they would with touch controls — and the simple convenience of being able to control vehicle functions with deliberate gestures rather than a potentially complex menu scheme.”

Gesture recognition is an important feature of this system, but it is only one aspect of a more broad and all encompassing “camera-based interior sensing platform that can interpret the voice commands, eye movements and body movements of the driver, while giving the vehicle the capability to recognize the driver, adjust settings to the driver’s preferences, monitor the driver’s attentiveness and mood, take over braking functions in an emergency, and switch back and forth between autonomous and human driving modes.”

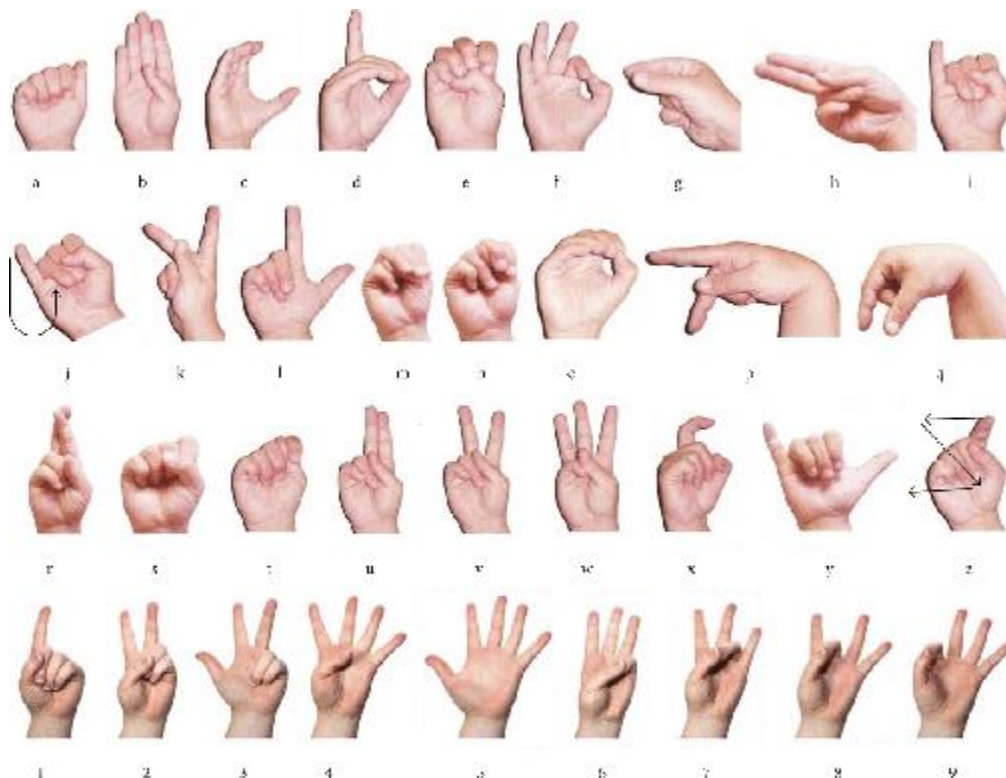


Gestural Control in BMWs

Another Example of hand gestures used in weddings:

Imagine that you're a wedding planner and hosting a wedding function for one of your customers. Everyone's having a great time, music's playing, and the function is very noisy. Suddenly, it's time for a ring ceremony! It's too loud to use Alexa, and rather than hunting for your phone or a remote control, what if you could simply raise an open hand while in mid-conversation, your smart home device would recognize that gesture, and turn off the music? And with the same gesture, you could dim the lights — just in time to see their couple dance performances and other dance performances as well. We will be developing a tool that will make this a reality.

From hot rods to fancy weddings to bar jukeboxes, gesture recognition could enable a whole new form of contactless human-machine interaction.



Data Sources

For our project we both augment existing datasets and create our own unique dataset. Our augmentation process is partially based on the “Hand Gestures” dataset from Kaggle. The Kaggle dataset includes 3000 color images of a set of gestures that are relevant for communication (the “okay” sign, pointing up, etc.). We combined this with subsets from the “Gesture Recognition” dataset from Kaggle which contains 20,000 images of a set of gestures from various people, subsets from the “Sign Language Gesture Images Dataset” which has 55,000 images, and the “Hand Signs/Gestures Dataset” which includes 2,000 images.

Alongside this we supplemented the dataset with our own original data which include roughly 4,000 images of our own hands and our friends’ hands. These will be taken using our webcams which we will also use to demo our model once it is trained. We believe this is useful because it not only adds original data but also will include a variety of different skin tones and hand structures to see if our augmentation process helps eliminate irrelevant differences.

We believe that using original photos, taken from multiple webcams, results in a more dynamic and ultimately more accurate model.

Our approach is strengthened by selecting a set of gestures common to multiple established data sets as well as our sophisticated approach to data augmentation.



Sample image from “Hand Gestures” dataset.



Sample image from “Gesture Recognition” dataset.

Gesture Selection

Many of the general gesture recognition datasets have inconsistent sets of individual gestures in them (because the gestures in one set aren't always included in another). As a result, we selected a set of gestures that commonly appear across data sets.

Our model will be trained on seven gestures that are common to gesture recognition datasets. While there might be a more intuitive set of gestures, our set is valuable because it extends the datasets commonly used for gesture recognition while also capturing a subset of distinct and potentially meaningful gestures:

1. Index up
2. L shape (index and thumb)
3. C shape
4. Thumbs up / Hitchhiker
5. Thumbs Down
6. Okay sign
7. Open Palm

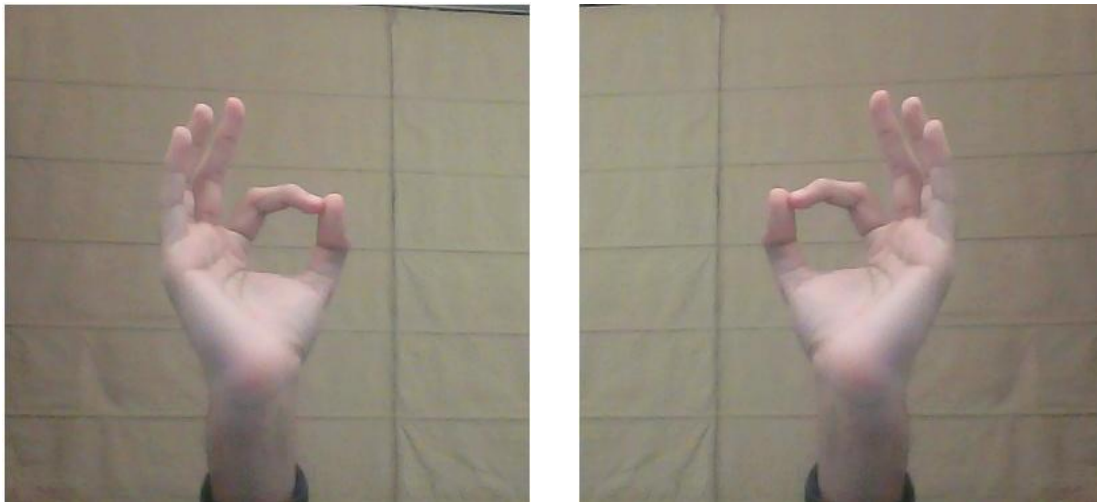
Aside from being useful due to their commonality across data sets, these are all useful gestures because they can be reasonably mapped to commands for human computer interaction. For example, “index up” could raise volume while the “L shape” could lower it, while “C shape” could close the current application. Again, these are only examples and each gesture could be mapped to any given command. We also hope that our augmentation will lead to a situation where users could use either their left or right hand for generating input to the model. It is also worth noting that none of our datasets included a thumbs down gesture, so we just inverted the thumbs up image vertically and used that as our training data.



Examples of each of our seven gestures from the “Hand Gestures” dataset.

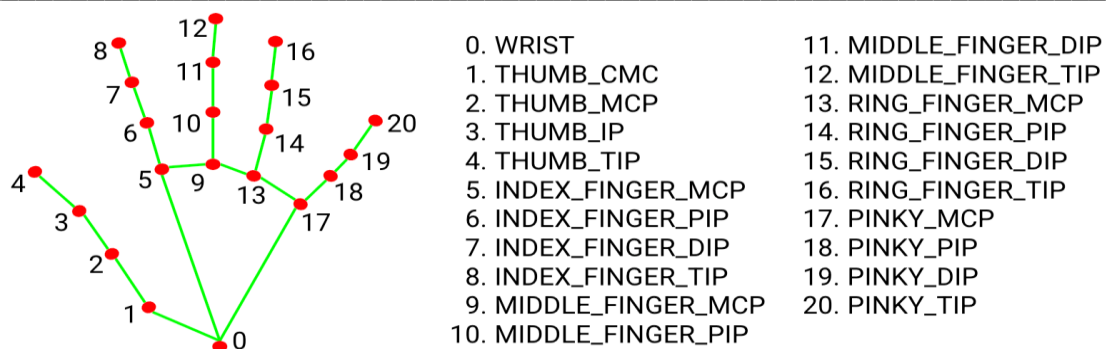
Data Cleaning and Augmentation

For our data augmentation we use OpenCV to develop a landmark-based dataset of our seven gestures. But before we apply landmarks we also augment and expand the data by making horizontally mirrored versions of each image. This resulted in a doubling of the number of images we were able to train our model on.



Examples of horizontally mirrored images from “The Hand Gestures” dataset.

We used OpenCV to then augment our newly compiled dataset and map landmarks onto each image. OpenCV maps hands using a set of twenty one landmarks that track points on the wrist, knuckles, and fingers.



Landmarks used by OpenCV

The landmark detection from OpenCV does two important things:

- 1) It detects whether there is a hand and then identifies the boundaries of the hand.
- 2) It maps landmarks onto any hands detected in the image.

As a result, this helps us both in segmentation and augmentation. It first helps us segment the hand out from the rest of the image then maps the landmarks onto the hand. Once the landmarks are mapped we see what looks like a constellation or skeleton of the hand.



Examples of landmark “constellations” or “skeletons”

We took the images from the Kaggle datasets as well as our original data and mapped landmarks onto each hand image. This results in a dataset of thousands of hand “skeletons” that we then used to train our model.

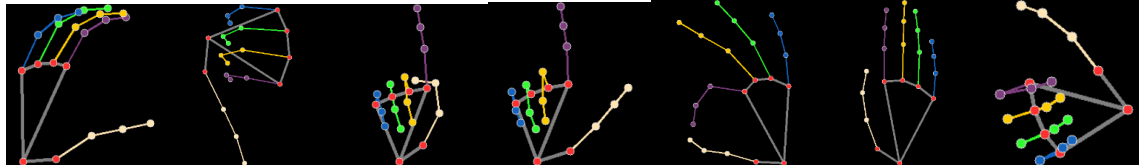
This skeleton approach is superior to using images of hands themselves because it helps remove a number of features that a model might mistake as relevant. For example, changing gesture images into landmark skeletons removes the variation present in skin color. Other datasets have handled this in other ways, for example changing all images into black and white or into silhouettes. We believe our landmark-based approach will prove better than these since it avoids variation in body fat and other features not removed through silhouette or black and white approaches.



Five Approaches to Data Augmentation (Raw, B&W, Silhouette, and Landmark)

Many approaches to gesture landmarks either use black and white landmarks or convert aspects of the landmarks to quantitative data. Our landmark approach pairs color-

coordinated landmarks with a model that is sensitive to RGB shifts in images. As a result, we capture meaningful information using colored landmarks while avoiding the pitfalls of colored images, overlayed landmarks, black and white images or landmarks, and silhouettes.



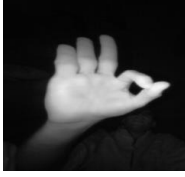


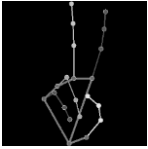
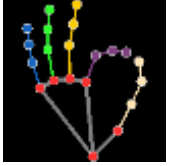
Examples of Each Gesture After Being Landmarked, Blacked Out, and Segmented

Our data augmentation methods help our model focus on the important aspects of each gesture while ignoring potentially irrelevant information like race, gender, or weight. It is also trained to identify gestures from either the left or right hand, making it more dynamic.

We further supplemented our data with thousands of images of “irrelevant gestures” that we want to train our classifier to ignore. This was technically necessary because we used a softmax activation layer. In layperson’s terms this means that our model would tie any input to one of the classes it was trained on. So without a baseline for empty frames and irrelevant gestures, it would just treat any input as one of our seven gestures, which would lead to utter chaos.

Beside being technically necessary because of our use of a softmax activation layer, this is also helpful because some irrelevant gestures might look more like a given trained gesture than they do a blank/black image. As a result we tried to include a number of common but irrelevant gestures as well as somewhat candid shots of hands in front of screens or people scratching their chins, etc. This was one of the biggest surprises and we are glad we were able to handle it without changing our model, which we outline in a later section.

Overview of the Comparative Strengths of Our Augmentation Process

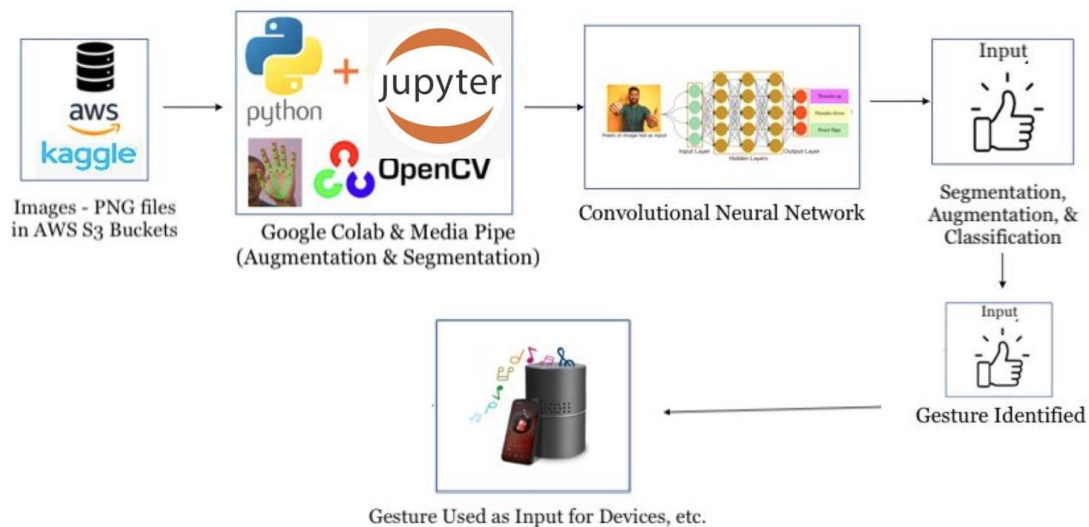
	<ul style="list-style-type: none"> • Black and White Images lack nuance and are bad for some models like the VGG-16, which requires an RGB input. • They also include the thickness of the fingers, which is also not relevant.
	<ul style="list-style-type: none"> • Color pictures are better, but can include unhelpful distractions like skin color as well as including the thickness of the fingers, etc. • As a result, training a model on color images would likely require a lot of data to handle variation.
	<ul style="list-style-type: none"> • Silhouettes avoid skin color confusion but still include thickness, which is irrelevant. • They also risk confusion between fingers in certain poses.
	<ul style="list-style-type: none"> • Black and white landmark methods avoid many of the pitfalls faced by other methods of augmentation but they are not well suited to models that require RGB inputs and miss out on potentially relevant information color can provide. • The standard minimal start example for webcam based landmark application in OpenCV is a form of this.
	<ul style="list-style-type: none"> • Color coordinated landmarks avoid including irrelevant information about skin tone and thickness but still get value from RGB models by coding relevant information (like which finger is which) according to color.

We think our method of augmentation allows for a dynamic model that can be trained on a small data set. It also manages to avoid the pitfalls of the alternatives while also retaining their virtues and even including new ones (such as the color-coordinated labelling of different fingers to help models discriminate between gestures.)

Architecture Outline

Our architecture involves data augmentation, segmentation, analytics, and classification. Each of these is elaborated in other sections, but a high level overview can be helpful. Our architecture begins with our data (both original and from Kaggle), then moves on to jupyter notebook where we use OpenCV and python for segmentation and augmentation of the original data.

After the data has been prepared for use in training we use it to train a convolutional neural network as a gesture classifier (more specifically a gesture landmark classifier). Convolutional neural networks tend to do well when it comes to image classification and we will specifically be using the VGG-16 pre-trained model using Keras & TensorFlow.

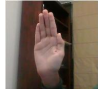


Architecture Outline Diagram

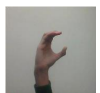
After we have trained and evaluated the model, the next thing is just to see how it performs on new images and in real life demos, in other words, on new input. As an image is taken for input into the model it is first segmented then landmarked using OpenCV so that it has the “skeletal” structure that our model is used to. Once the gesture is identified, then it can be used as the input for devices such as music players, laptops, etc.

We believe that part of the value of our gesture recognition system is how dynamic it is but for demo purposes we will be using the Spotify-CLI to connect our classifier to devices. The Spotify-CLI allows users to control devices that are connected to spotify.

Our demo will allow a user to control the basic functionality of their wifi-connected music player by issuing gestures to a webcam. This provides basic functionality but more functionality could be added for new gestures. The commands are as follows:



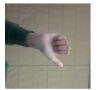
Palm = Pause



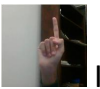
"C" Shape = Play/Continue



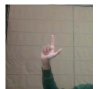
Thumbs Up = Volume Up 20



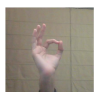
Thumbs Down = Volume Down 20



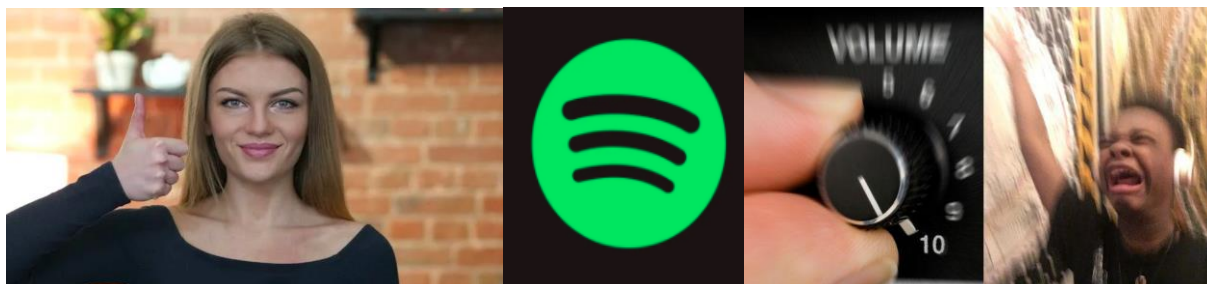
Index Up = Next Song



"L" Shape = Last Song



"Ok" Sign = Current Status



Our Demo (Dramatization)

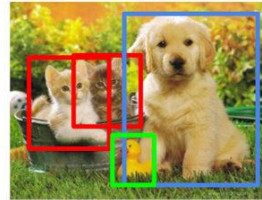
Analytics & Model Selection

Classification



CAT

Object Detection



CAT, DOG, DUCK

Examples of Classification and Object Detection

Image classification, localization, segmentation, and object detection are some of the key problems in the field of computer vision. Among these, Image classification (recognition) is one of the most fundamental tasks in computer vision. It refers to the task of assigning a label to an image.

Deep learning, a subset of Machine Learning, is highly relevant for areas like computer vision and speech recognition. Deep learning is in fact a new name for an approach to artificial intelligence using neural networks, which have been going in and out of fashion for years. Neural networks are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. For the purpose of our project we have chosen a Convolutional Neural Network (CNN).¹ CNNs tend to do well at classifying images. We will be using a specialized deep neural network model for handling image data.

¹ <https://www.ibm.com/cloud/learn/neural-networks>

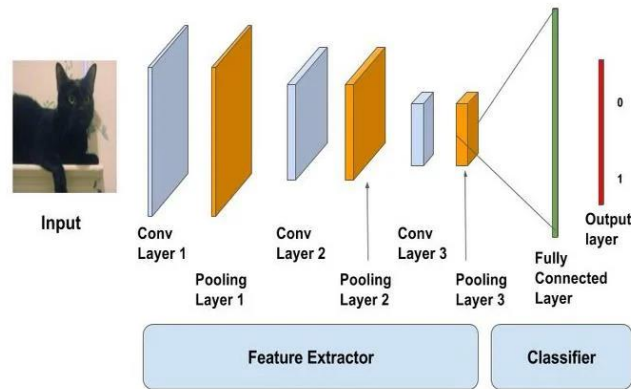


Diagram of a CNN

To use any CNN model we have to first train it by using large sets of labeled data and neural networks that contain multiple learning layers. But training any image classification model from scratch requires setting millions of parameters, a ton of labeled training data and a vast amount of compute resources (hundreds of GPU hours). While not as effective as training a custom model from scratch, using a pre-trained model allows you to shortcut this process by working with thousands of images vs. millions of labeled images and build a customized model fairly quickly (within an hour on a machine without a GPU). Because of the following constraints, we will be using a pre-trained VGG-16 model for image processing.

OPTION A: BUILD YOUR OWN CNN



Low Dimensional Image



Simple CNN



Adequate

OPTION B: PRE-TRAINED MODELS



More Complex Images



Pre-Trained Model



Predict Various Classes

Understanding transfer learning and pre-trained models

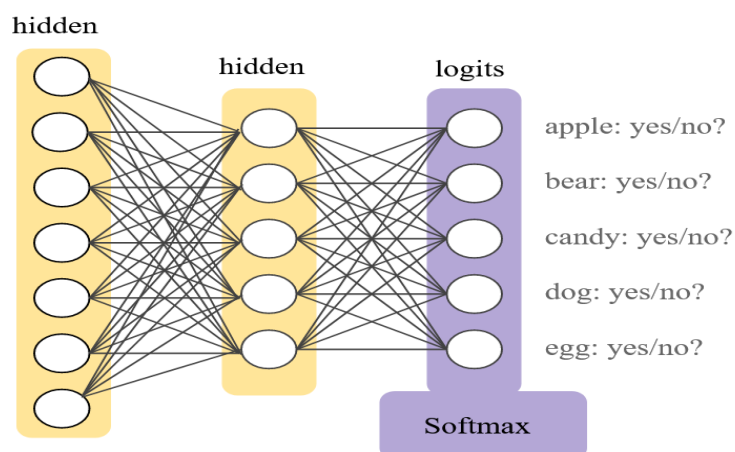
Let us understand what this VGG-16 model is. VGG is a convolutional neural network with a specific architecture that was proposed in the paper — “Very Deep Convolutional

Networks for Large-Scale Image Recognition” by a group of researchers from the Oxford Visual Geometry Group, or VGG for short. The VGG group that developed this CNN won the image classification task challenge they developed it for.

The group made their models and learned weights available online. The VGG model weights can be downloaded and used in the same model architecture using a range of different deep learning libraries, including Keras. This way of using pre-trained models with minor modification on wholly new predictive modeling tasks, harnessing the state-of-the-art feature extraction capabilities of proven models is “transfer learning.”

VGG has two different architectures: VGG-16 which contains 16 layers and VGG-19 which contains 19 layers. For our project, we used the VGG-16 pre-trained classifier model that mainly contains three different parts: convolution, pooling, and fully connected layers.

We added a classification head with a dense layer and softmax activation to the pretrained model. This classifier attempts to classify gestures using high level features of the images. The softmax layer ensures that we get a single prediction at the end by normalizing the output of our neural network to a probability distribution over our predicted output classes. This is why we had to include a set of irrelevant gestures in the training data. Had we not included irrelevant gestures, each irrelevant gesture would be tagged as the trained gesture it most closely resembles.



An Outline of How Softmax Activation Layers Function

As a result, we had to adapt our data set to our model since it has a softmax activation layer. Once we have our pre-trained model up and running we use it for the image classification task that handles the input for our music control system. Because the model was trained on augmented data, and the input is similarly augmented, it ends up being extremely dynamic, lightweight, and accurate. The below points outline the steps of our analytics process:

1. Install required libraries
2. Download model
3. Read model
4. Add layers and fit the model
5. Evaluate accuracy
6. Get the camera frame
7. Detect the hand in frame
8. Classify gesture using pre-trained model
9. Recognize, label, and use gesture as input

This pipeline takes us from training to input to basic functionality for controlling a music player.

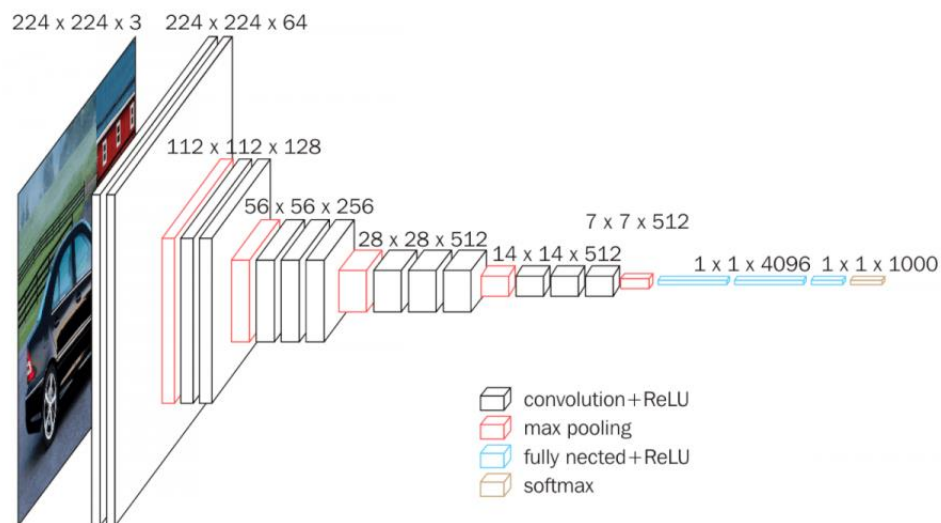


Diagram of the VGG-16 Architecture

Our model's validation accuracy was 95.78% and our model consistently performs well in live tests using totally novel images from a webcam. Besides our validation accuracy, we tested the model for precision (95.9%), recall (93.4%), and its F score (93.9%). We ran seven epochs on the data until we noticed a minimization of validation loss. After validation we saved the model so it could be used in the webcam-to-spotify pipeline we developed. This pipeline also segments and augments any new input to mirror the color-coordinated landmark format our model is familiar with from the training data.

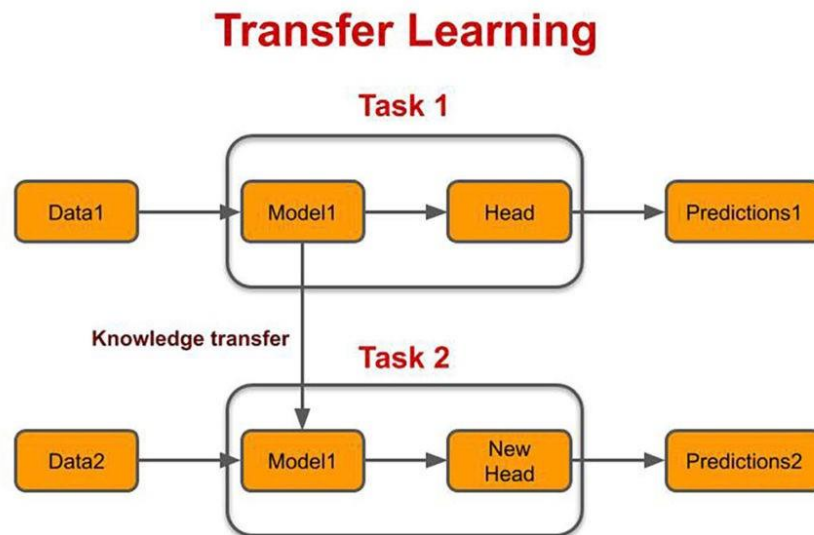
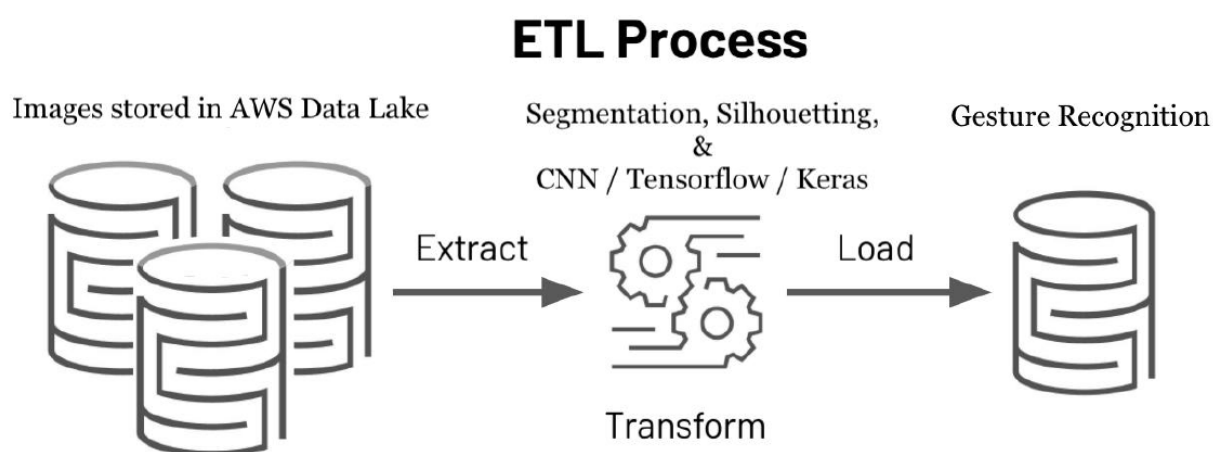


Diagram of the Transfer Learning Process

Our transfer learning approach appears to have been successful. The head we added to our model seems to have successfully transferred its past training base to our gesture dataset and made our version of the VGG-16 an effective gesture classification tool

ETL

ETL is an acronym for Extract, Transform, and Load. ETL is a technique for transferring data from numerous sources into a data warehouse. Data is extracted from the source system into the staging area in the first step of the extraction process. The data retrieved from the source is cleansed and transformed in the transformation process. The final stage in the ETL process is to load data into the destination data warehouse.



Extract:

We have collected our data from three different data sources. The datasets were collected from Kaggle. The Kaggle dataset contains 3000 color photos of a series of communication gestures (the "okay" sign, pointing up, and so on). We coupled this with sections of Kaggle's "Gesture Recognition" dataset, which has 20,000 photographs of diverse people's gestures, and the "Hand Signs/Gestures Dataset," which comprises 2,000 images, as well as a subset of the third set which has 55,000 images. The third dataset was created with our own unique data, which included over 4,000 photos of our own hands and the hands of our friends. These will be captured with our webcams, which will also be used to demonstrate our trained model. This is beneficial since it not only adds new data, but it also includes a variety of various skin tones and hand forms to determine if our augmentation technique can assist in reducing irrelevant differences.

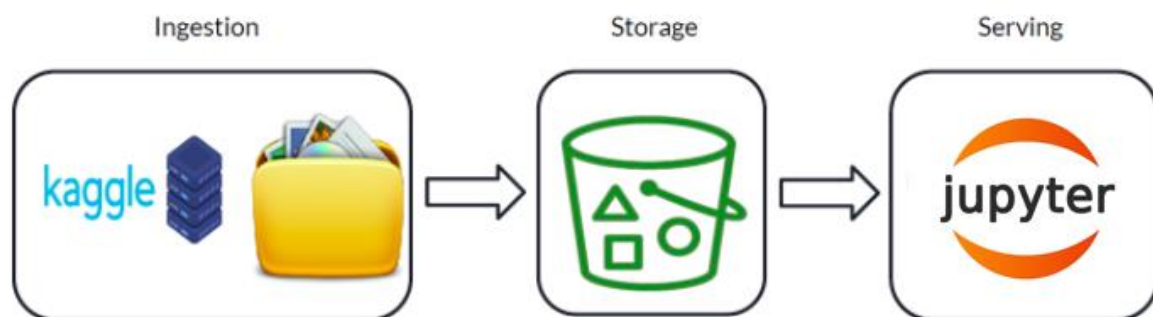
Transform:

We used OpenCV to add landmarks to each image and update our newly built dataset. OpenCV tracks points on the wrist, knuckles, and fingers using a collection of twenty-one landmarks. Along with that, we have restructured all the images to 224x224 pixels.

Load:

We used Amazon S3 buckets to store the data related to our project. S3, also known as Simple Storage Service, is the most popular and commonly utilized storage in AWS. S3 is a globally accessible service that may be accessed through the AWS Management console, a user-friendly web interface. We need to take a closer look at a few basic principles to gain a better understanding. The data is stored in Amazon S3 as objects within buckets. An object can be made up of files or metadata about those files. There are a few phases to storing an object in S3, and when you upload a file, you may define rights for both the bucket and the file.

The things are contained in buckets. You can make as many buckets as you want. Users can create up to 100 buckets per AWS account by default. You can, however, visit AWS Service Limits to increase your Amazon S3 bucket limit.

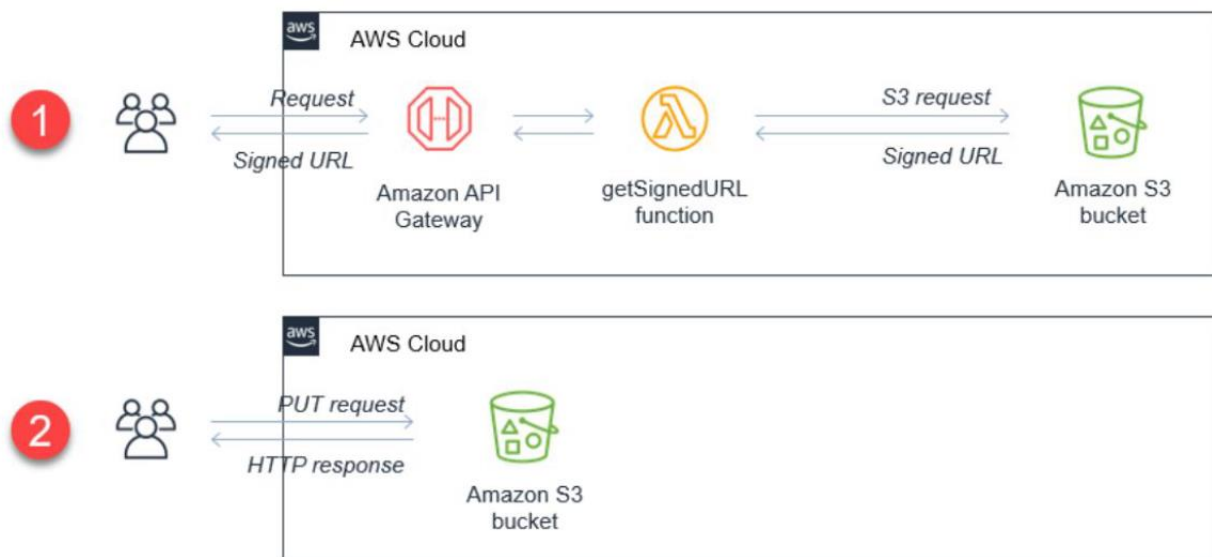


Above diagram represents the architecture of the data being used for our project. Data ingestion is carried out from Kaggle and data stored on a local computer. These images are loaded into amazon S3 bucket and are served for further processing using python code.

Loading and Serving the data using Amazon S3 bucket using Boto3:

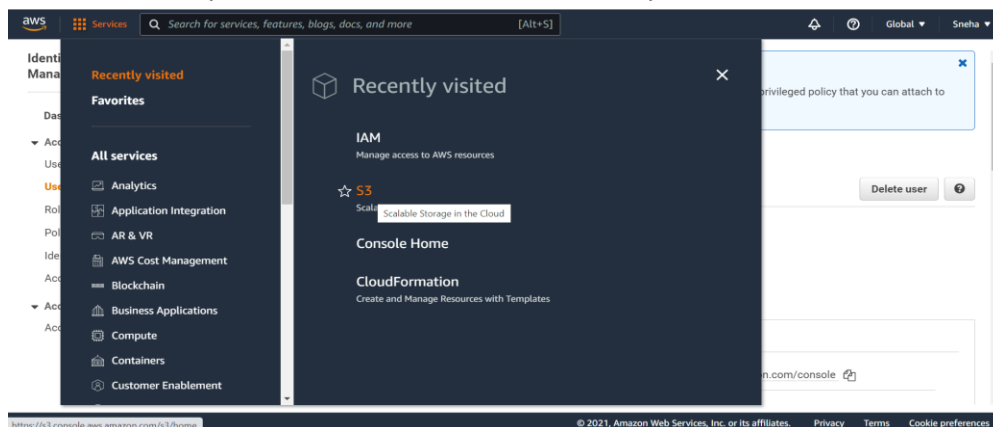
Using AWS services such as the AWS SDK or CLI, we can simply send and pull data from S3. SDKs are available in a variety of computer languages, including Python, which is where boto3 comes in.

The Amazon Web Services (AWS) SDK for Python is boto3. Python programmers can use it to construct, configure, and manage AWS services like EC2, S3, and others. It has an object-oriented API that is simple to use, as well as low-level access to AWS services.



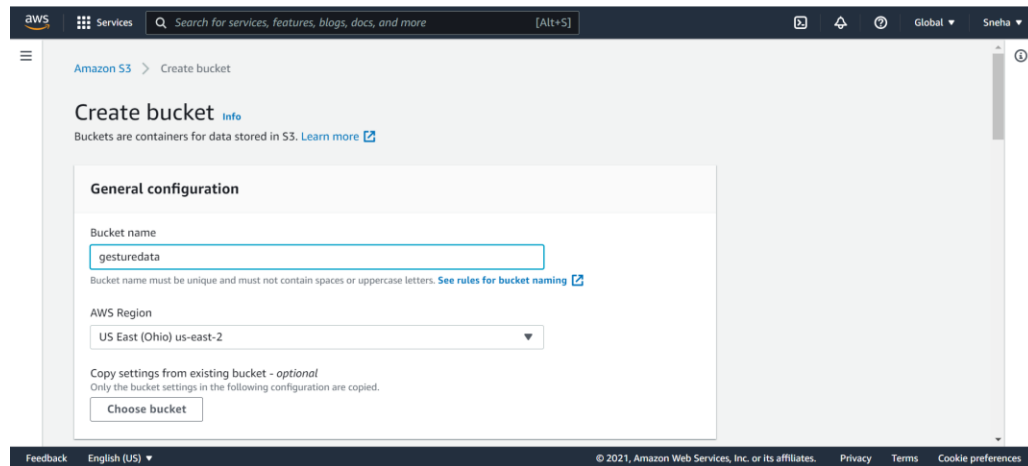
Step1: Upload data using Amazon console

We selected the S3 option to load the data. Next step is to create the bucket.



- Create S3 bucket

Using the S3 console, we created the bucket named gesture data as below.



- Upload data into bucket

We have uploaded all the augmented images into Amazon S3 bucket via console.

Step2: Generate the Secret Tokens

After the above steps, we require to have secret credentials provided by AWS in order to use boto3:

AWS_ACCESS_KEY_ID

AWS_SECRET_ACCESS_KEY

- Create the User
- Save the Access key and Secret key

After we have set up credentials, keys, etc. we can begin reading the data set into our buckets. This requires a number of steps.

Step3: Read the dataset into Python using Boto3:

- Creating the low-level functional client
- Creating the high-level object-oriented interface
- Fetch the list of existing buckets
- Make the bucket objects public
- Read the presigned public URLs for each image
- Save the images into an array for further processing in the analytics pipeline or webcam-to spotify pipeline.

This is how we integrated the AWS SDK (boto3) into our project to take advantage of S3's capability and easily solve data storage problems. Not only can boto3 be used to access and use file storage services, but it can also be used to access and use a variety of AWS services from python code.



Webcam to Spotify Pipeline

We chose to demo our model using Spotify, the online music streaming application. Instead of using an API (Application Programming Interface) we used a CLI (Command Line Interface) and then just structured our Python code to map the output of our model to the input of the CLI.

We connect our model, our webcam, our augmentation methods, and our CLI command executor all in a single script. This script reads frames from the webcam and maps landmarks using color coordinated points and lines onto the frame (which appears otherwise black to mirror our augmentation method). Then our model predicts each of the gestures until it recognizes a command to send to Spotify via the CLI.

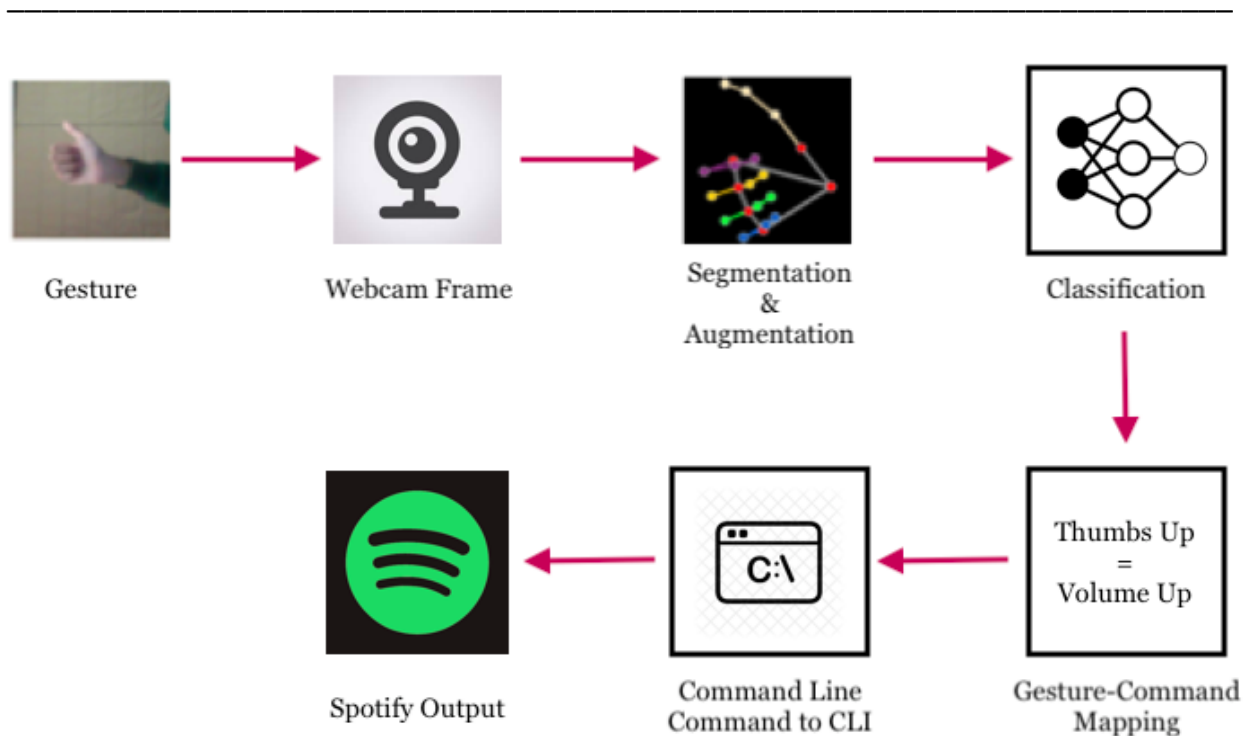


Diagram of Our Webcam to Spotify Pipeline

Summary & Conclusion

In this project we tried to predict gestures which can be used to control devices with different gestures aligned to different commands. In our case that device is Spotify Web Player.

We used pre-trained model VGG16 for building our model. Our model performs beyond expectations and peaked at an 95.78% accuracy during a deployment of 6 epochs, lasting roughly 1.5 hours. The first few deployments allowed us perspective into how to better the model for a more efficient and appropriate usage. Since we used a pre-trained model, we did not change the initial layers and only changed the last layer classification from 1000 to 7 to fit our model.

Once we were ready with our model, we loaded that model and used it for controlling the Spotify. And yes, we were able to control the Spotify with our gestures.

References

1. <https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/image-classification>
2. <https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/>
3. <https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/>
4. <https://www.ibm.com/cloud/learn/convolutional-neural-networks>
5. <https://neurohive.io/en/popular-networks/vgg16/>
6. <https://knowledge.dataiku.com/latest/kb/analytics-ml/image-classification-visual-way/pretrained-model-summary.html>
7. <https://www.kaggle.com/imsparsh/gesture-recognition?select=train.csv>
8. <https://www.kaggle.com/parikshitkumar/hand-gestures>
9. <https://aws.amazon.com/lambda/>
10. <https://learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/>
11. <https://google.github.io/mediapipe/solutions/hands.html>
12. https://en.wikipedia.org/wiki/Image_analysis

13. <https://www.datacamp.com/community/tutorials/object-detection-guide>
14. <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>
15. <https://www.sqlshack.com/getting-started-with-amazon-s3-and-python/>
16. <https://www.aptiv.com/en/insights/article/what-is-gesture-recognition>