Faculty of Computer Science & Engineering

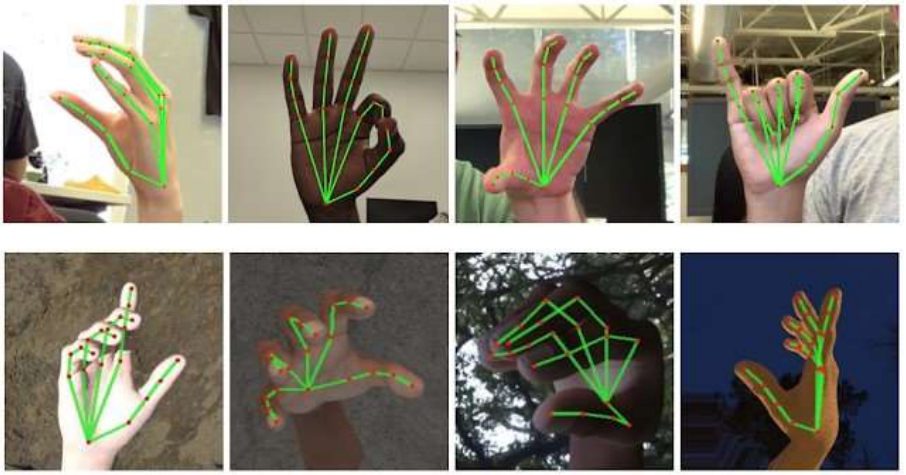
Project for the subject:

Digital Image Processing

Topic:

Gesture recognition

Media player



Mentor:
Ivica Dimitriovski

Made by:
Filip Shijakov 196048
Mihaela Pavleska 196016

Abstract

Computers have been on the forefront of engineering and practical usage of mathematical theories. The one computer that the science community hasn't managed to crack is our own - the brain. Parallel with that development, humans have been trying to make their lives easier through automatization and wireless communication. Our project aims to bridge the gap between ease of use and human-like thought processes with the usage of deep learning and neural networks. In this paper we will be evaluating the algorithms that attempt to simulate the brain, and how we can use it for everyday practices.

We used these algorithms to create a simple media player using gesture recognition. Gestures are the most natural way for people to express information in a non-verbal way. Our approach has some advantages:

- *No additional equipment needed.*
- *More human-friendly controls.*
- *All you need is a camera.*

Introduction

Artificial intelligence (AI) is the future. It is already part of our everyday lives. With the rise of the artificial intelligence and the more and more parts of our lives it gets involved with, we get introduced to concepts popular as Machine Learning, Deep Learning, Neural Networks, etc. But what are they really?

With the great improvement of computers and internet technologies, we come across those terms a lot. Social Media, Games, Photo Applications and Security services as simple as unlocking your phone use these things. You probably have fingerprint lock on your phone or face recognition that you use every day, but do you know how it works?

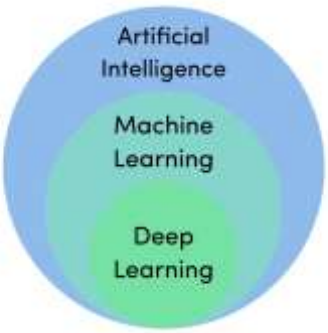
You will learn about them, what they are, what makes them unique and significant, and what the distinctions between them are.

Our program is a Media Player that uses gesture recognition for controls, allowing you to enjoy your favorite movie from the comfort of your own bed and give instructions with your hands (PAUSE, PLAY, MUTE, etc.).

MediaPipe, a Google open-source framework for media processing, was used to create the software. Machine learning (ML) is used to extract 21 landmarks of a hand from a single shot. We'll look at how this app is developed, what algorithms it uses, how they work, and how we can use all of this knowledge to create complex tools that make our lives easier and more fun.

Machine Learning

Machine learning is a branch of artificial intelligence that uses data analysis for analytical model building. It's based on the idea that systems can examine the data, identify patterns, learn from it and make decisions with minimal human intervention.
Researchers interested in AI wanted to see if computers could learn from data without being programmed to act a certain way. ML is a product of that idea and pattern recognition.



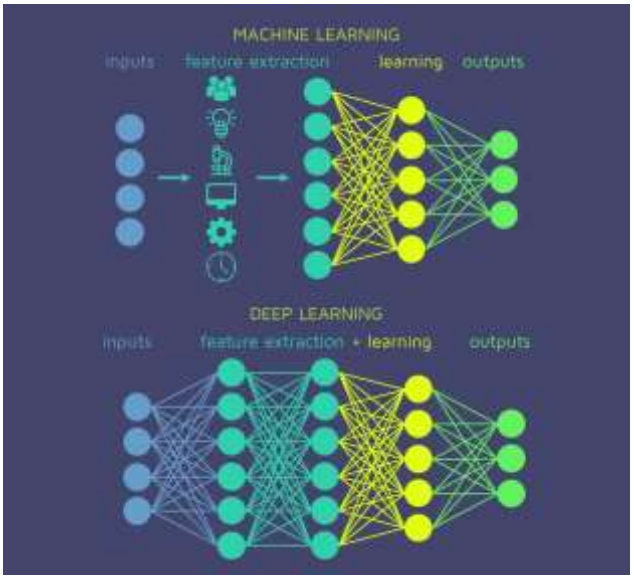
ML algorithms have been around for a long time, but the ability to automatically perform complex mathematical calculations fast to big data is a recent development. Today the essence of ML is widely known and we have a lot of ML applications for everyday life, online recommendations (Netflix, Amazon, Spotify, ..), fraud detection, email filtering, speech recognition, self-driving Google cars, finding new energy sources, health care devices and sensors that can use data to assess a patient's health in real time, etc. By using algorithms to build models that uncover connections, organizations can make better decisions without human intervention.

There are four machine learning approaches:

- **Supervised learning** algorithms are trained using labeled examples, such as an input where the desired output is known. Supervised learning uses patterns found in the examined input to predict likely future events on unlabeled data.
- **Semi-supervised learning** algorithms are used for the same applications as supervised learning, but they train both from labeled and unlabeled data, because unlabeled data takes less effort to acquire and is cheaper. Semi-supervised learning is useful when the cost associated with labeling is too high to allow for a fully labeled training process.
- **Unsupervised learning** algorithms don't know the expected outcome. The goal is to explore the data and figure out what is being shown.
- **Reinforcement learning** is a branch of machine learning that studies how intelligent agents should operate in each environment to maximize the concept of cumulative reward. Reinforcement learning, along with supervised and unsupervised learning, is one of the three main machine learning paradigms.

Deep Learning

Deep learning is a machine learning technique that teaches computers to learn by example. It learns directly from images, text or sounds. Models are trained by using labeled data and complex neural network architectures. Similar to machine learning, deep learning can be supervised, semi-supervised or unsupervised. Deep learning uses statistics and predictive modeling. Machine learning algorithms are typically linear and deep learning algorithms are stacked in a hierarchy.



The first step in a machine learning workflow is to manually extract useful characteristics from photos. The characteristics are then used to build a classification model for the elements in the image. But using the deep learning approach, relevant characteristics are automatically obtained from photos. Furthermore, deep learning performs "end-to-end learning," in which a network is given raw data and a goal to complete, such as classification, and it automatically learns how to do so.

Another distinguishing factor is that deep learning algorithms scale as data grows, whereas shallow learning techniques converge. Machine learning algorithms that peak at a given level of performance as more instances and training data are added to the network are referred to as shallow learning.

Each algorithm in the hierarchy applies a nonlinear transformation to its input and uses what it learns to create a statistical model as output. Iterations continue until the output has reached an acceptable level of accuracy. The number of processing layers through which data must pass is what inspired the label **deep**.

In machine learning, the programmer needs to supervise the program and pair the best algorithms with the right tools and processes and be specific with what types of things it should be looking for to decide if the content is of adequate quality. This is called feature extraction and the success rate depends on the programmer's ability to properly define the weights of the features.

For the output of deep learning to be precise and always accurate it needs a lot of training on enormous data sets until the weights of the neural inputs get the right answer every time. At this point the neural network has taught itself what a stop sign is, or your face in the case of Facebook, or a cat, which is what Andrew Ng did in 2012 at Google.

Ng's breakthrough was to take these neural networks, and essentially make them huge, increase the layers and the neurons, and then run massive amounts of data through the system to train it. In Ng's case it was images from 10 million YouTube videos. Ng put the "deep" in deep learning, which describes all the layers in these neural networks. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

Today, image recognition by machines trained via deep learning in some scenarios is better than humans, and that ranges from cats to identifying indicators for cancer in blood and tumors in MRI scans.

While deep learning was first theorized in the 1980s, the need for large amounts of labeled data and substantial computing power are the two main reasons it has only recently become popular and useful. High performance graphics processing units have a parallel architecture that enables reducing training time for deep learning.

Most deep learning methods use neural network architectures. Neural networks come in different forms, such as recurrent neural networks, convolutional neural networks, artificial neural networks, and feed forwarding neural networks. They all have benefits for specific use cases, and they function in a similar way, by giving data to the model and letting it figure out itself whether it has made the right decision or progress about a given data element.

Google's AlphaGo, a computer program that plays the board game Go, learned the game and became the best player by playing a lot of games by itself and learning from the mistakes.

Convolutional Neural Networks (CNNs or ConvNet)

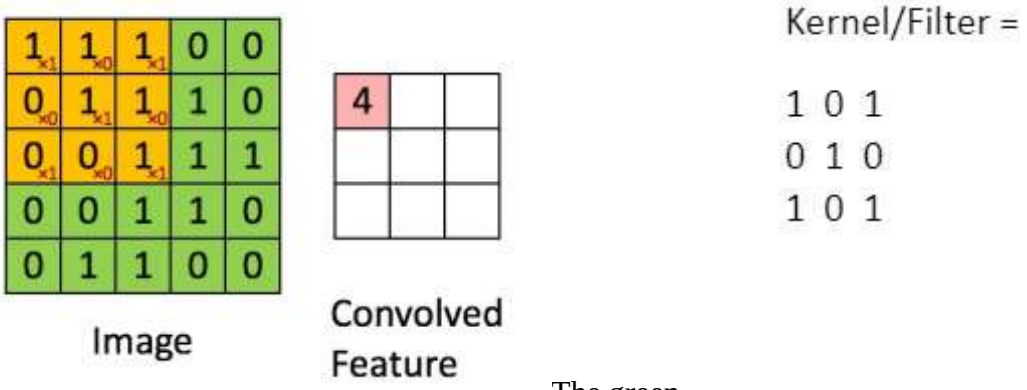
One of the most popular types of deep neural networks is known as convolutional neural networks. It is an algorithm that takes image input, assigns weights and biases to different aspects of the image and it’s able to differentiate one from the other. They have applications in image and video recognition, image classification, image segmentation, medical image analysis, etc. That’s how the algorithm that we used for our gesture recognition works and will take a deeper look into it now.

The architecture of a CNN is analogous to that of the connectivity pattern of neurons in the human brain. Each neuron receives input from only a restricted area of the previous layer called the neuron's receptive field.

A matrix of pixel values makes up a picture. For simple binary pictures, flattening the image (3x3 image to a 9x1 vector) and sending it to a Multi-Level Perceptron for classification would satisfy the requirements, producing an average accuracy score while accomplishing class prediction. When it comes to complicated pictures with pixel dependencies, however, this solution would have little to no accuracy.

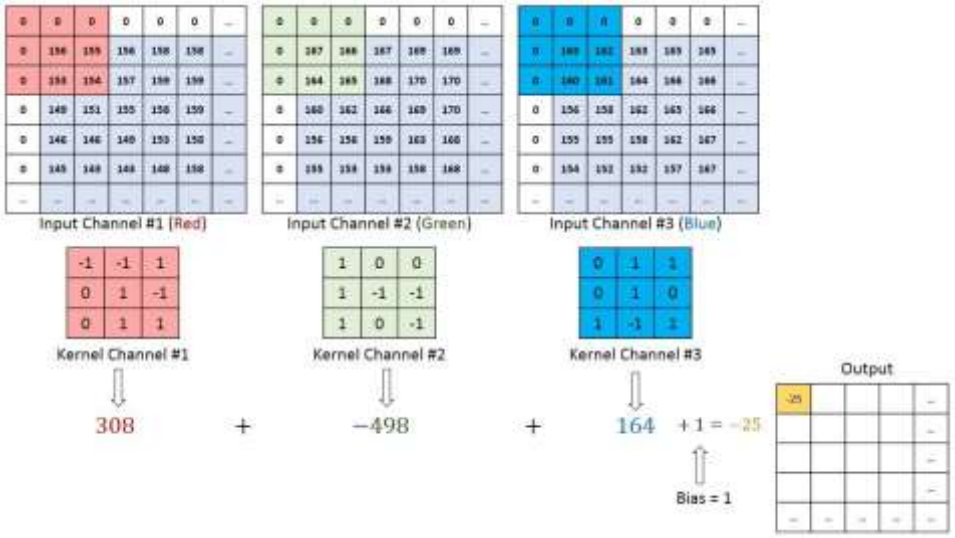
Working with a larger picture requires a lot of processing power. CNN's job is to condense the pictures into a format that is easier to process while preserving elements that are important for a successful prediction. This is critical for creating an architecture that is capable of learning features while also being scalable to large datasets.

Convolution Layer — The Kernel



The green section resembles a 5x5x1 input image. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter represented in yellow.

Because Stride Length = 1 (Non-Strided), the Kernel shifts 9 times, each time executing a matrix multiplication operation between the kernel's values and the region of the picture that the kernel is hovering over. With a given Stride Value, the filter travels to the right until it parses the whole width. Moving on, it uses the same Stride Value to hop down to the beginning (left) of the picture and repeat the procedure until the full image has been traversed.



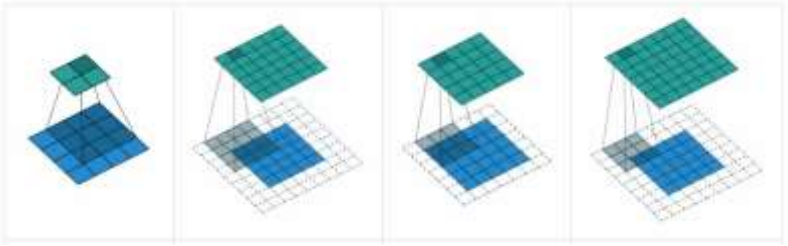
The Kernel has the same depth as the input picture in the case of images with many channels (e.g. RGB). Matrix multiplication between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) is conducted, and the results are averaged with the bias to give a compressed one-depth channel Convoluted Feature Output.

The Convolution Operation's goal is to extract high-level data from the picture, such as edges. No need to limit ConvNets to just one Convolutional Layer. The first ConvLayer is responsible for collecting Low-Level information such as edges, color, gradient direction, and so on. With added layers, the architecture adapts to High-Level characteristics as well, resulting in a network that understands all of the photos in the dataset in the same way that people do.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in the case of the former, or Same Padding in the case of the latter.

When using Valid Padding, the input image is not padded. This means that the filter window always stays inside the input image. This type of padding is called valid because for this padding only the valid and original elements of the input image are considered. When using Valid Padding, there can be a loss of information. Generally, elements on the right and the bottom of the image tend to be ignored. How many elements are ignored depends on the size of the kernel and the stride.

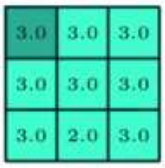
When using Same Padding, the input is half padded. The padding type is called SAME

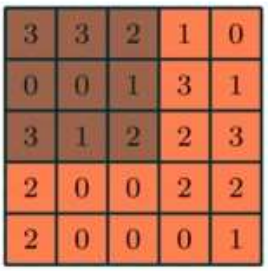


because the output size is the same as the input size (when stride=1). Using ‘SAME’ ensures that the filter is applied to all the elements of the input.

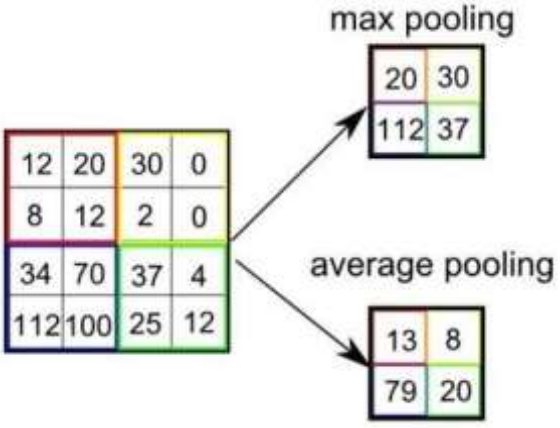
Pooling layer

The Pooling layer, like the Convolutional Layer, is responsible for shrinking the Convolved Features spatial size. Through dimensionality reduction, the computational power required to process the data is reduced. It's also beneficial for extracting rotational and positional invariant dominating features, which helps keep the model's training process running smoothly.





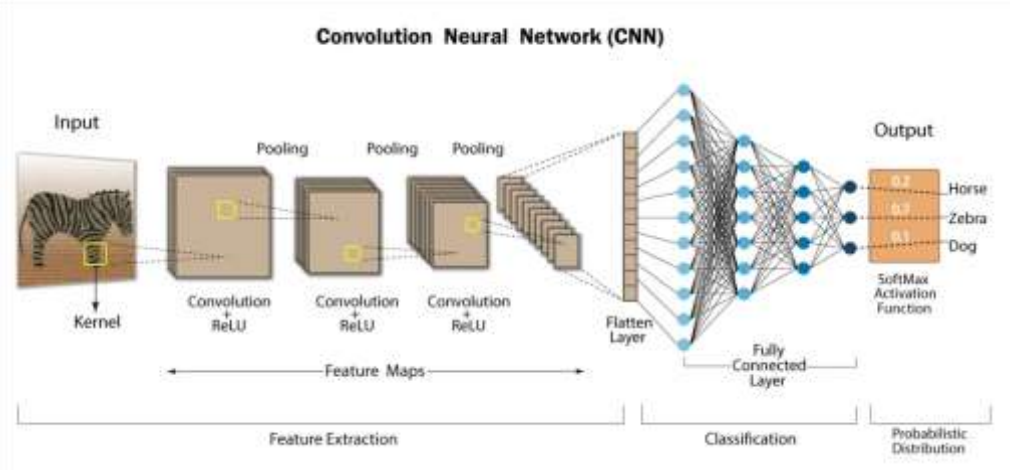
Pooling is divided into two types: maximum pooling and average pooling. The largest value from the part of the picture covered by the Kernel is returned by Max Pooling. Average Pooling, on the other hand, returns the average of all the values from the Kernel's section of the picture.



The i-th layer of a Convolutional Neural Network is made up of the Convolutional Layer and the Pooling Layer. Depending on the picture complexity, the number of such layers may be expanded even higher to capture even more low-level features, but at the cost of more computational power.

Fully connected layer

The final classification is done using fully connected layers after multiple convolutional and max pooling layers. In normal (non-convolutional) artificial neural networks, neurons in a fully connected layer have connections to all activations in the former layer. As a result, their activations may be calculated using an affine transformation with matrix multiplication and a bias offset (vector addition of a learned or fixed bias term).



Single shot detector (SSD)

The difference between image classification and object detection is that in image classification, the return value is a prediction of the object that is on the image, while in object detection not only is the object predicted, but its location is bounded with anchors.

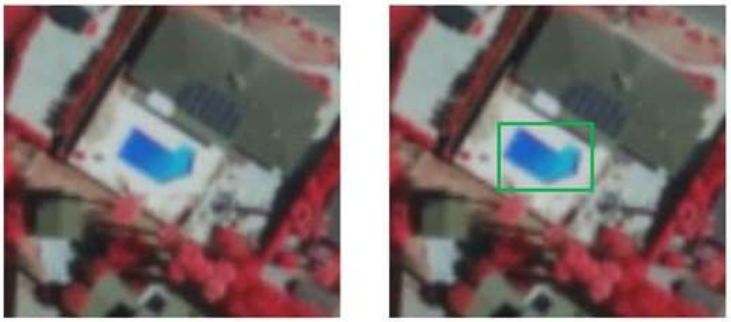
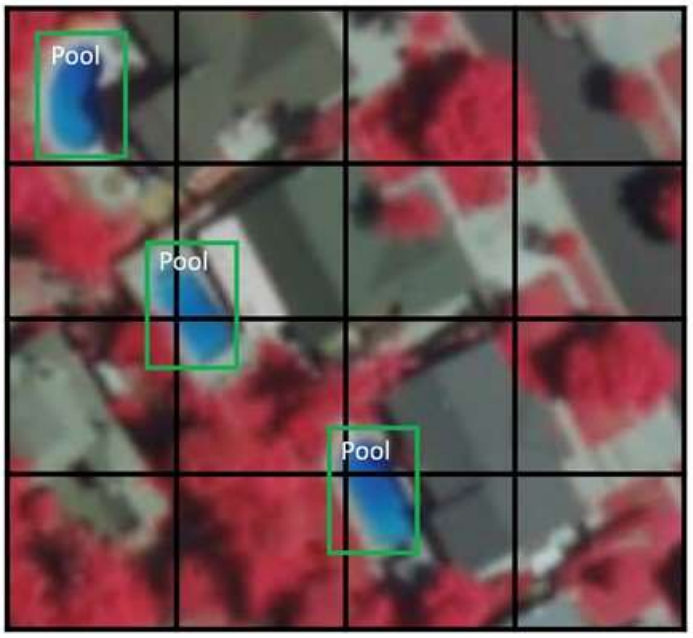


Image classification Object detection

Single-shot detector is an object detection algorithm that finds the object with the help of a convolutional network in one pass through the network. The SSD consists of two components: the backbone model and the SSD head. The backbone model is a pre-trained image classification network that is used as a feature extractor. The SSD head is composed of one or more convolutional layers added to the backbone model and the output is the bounding box and classes of the objects in the spatial location of the final layer's activation.

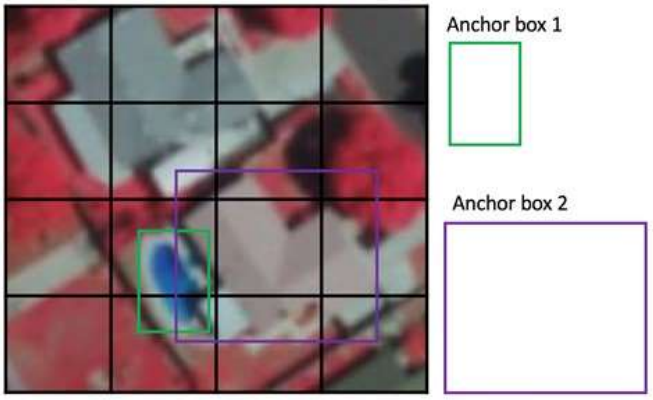
SSD divides the image like a grid and has each grid cell be responsible for detecting



objects in that cell. If no object is present, it is considered as a background class and the location is ignored.

Anchor box

Anchor boxes are assigned to each grid cell in the SSD. These anchor boxes are predefined and each one is responsible for size and shape within a grid cell. For example, the swimming pool in the image below corresponds to the taller anchor box while the building corresponds to the wider box.



In the training phase SSD tries to match the appropriate anchor box with the bounding boxes of each ground truth object. The anchor box with the highest degree of overlap with the object is responsible for predicting the class and location of that object. This is used for training the network and detecting objects once the network has been trained.

Zoom level

Objects in each grid cell can be of different size, for some objects to be detected anchor boxes of larger size could be required, and for others of smaller size. For this reason, the zoom parameter is used to specify how much the anchor box could scale up or down with respect to each grid cell.

Receptive field

The region in the input space that a particular CNN's feature is looking at is called a receptive field. Features at different depths of the convolutional layers have different size receptive fields, the deeper the network goes, the larger the receptive field of each feature is. Features in the same feature map have the same receptive field and look for the same pattern but at separate locations.

Receptive fields enable SSD to detect objects at different scales and output a tighter bounding box. Several convolutional layers are applied to the backbone feature map and each layer outputs a detected object as a result. As earlier layers represent smaller receptive fields, they are useful for detecting smaller size objects, while deeper layers are responsible for detecting larger objects.

Hand detection

For this project we are using MediaPipe, a hand detection algorithm developed by Google.

MediaPipe Hands uses a machine learning pipeline consisting of multiple models working together. Firstly, a palm detection model that returns a bounding box around the hand, a hand landmark model that works on the cropped image defined by the palm detector and returns 21 hand landmarks.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in the pipeline crops can be generated by the previously defined landmarks, the palm detector is only invoked when the landmark model can't identify the hand presence.

Palm detection model

To detect the initial hand location, a single-shot detector is used. Detecting hands is a complex task, the model has to work on a variety of hand sizes and be able to detect occluded and self occluding hands. The lack of sharp contrast features (like mouth and eyes in a face) in hands makes it difficult to detect them from their visual features alone

The problem above is addressed using a different strategy. First, a palm detector is trained to detect the palm, since estimating a bounding box over a rigid object is significantly simpler than detecting hands with articulated fingers. Moreover, palms can be modeled using square bounding boxes ignoring other aspect ratios, therefore reducing the number of anchors by a large factor.

Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects.

Lastly, the focal loss is minimized to support many anchors resulting from the high scale variance.

With the above techniques, an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Hand landmark model

After the palm detector, a subsequent hand landmark model identifies 21 hand coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

Dataset

To obtain ground truth data, the following datasets have been created:

In-the-wild dataset: This dataset contains 6K images of large variety, e.g., geographical diversity, various lighting conditions and hand appearance. The limitation of this dataset is that it does not contain complex articulation of hands.

In-house collected gesture dataset: This dataset contains 10K images that cover various angles of all physically possible hand gestures. The limitation of this dataset is that it's collected from only 30 people with limited variation in background. The in-the-wild and in-house dataset are great complements to each other to improve robustness.

Synthetic dataset: This dataset contains 100K images from a rendered synthetic hand module over various backgrounds. A commercial 3D hand model is used, rigged with 24 bones and includes 36 blendshapes, which control fingers and palm thickness. The model also provides 5 textures with different skin tones. Video sequences of transformation between hand poses have been created and 100K images have been sampled from them.

Solution Architecture

Our project is a media player that is controlled by the gestures of the hand. The media player takes 7 different instructions: 'PLAY', 'PAUSE', 'FAST FORWARD', 'REWIND', 'NEXT VIDEO', 'PREVIOUS VIDEO' and 'VOLUME <num>'. These instructions are represented by 5 different hand gestures.

A gesture is represented as a form of a string with 6 binary values. The first value is for the orientation of the hand, 0 for vertical and 1 for horizontal orientation. The next five binary values are for the thumb, index, middle, ring and pinky finger respectfully, 0 when the finger is straightened out and 1 when it is tucked in. With our representation 26 unique hand gestures are possible, we use 6 hand gestures in our program. The string with binary values Is transformed into an instruction name and it is given to the media player.

The media player is implemented using PyQt5. PyQt is a GUI toolkit for the software Qt, implemented as a Python plug-in. We use one QThread where we capture video input frame by

frame using OpenCV2, perform all the calculations in order to find the position of the hands and emmett back to the main window both the frame and the corresponding action. For the video player we use QWidget which is the base class of all user interface objects.

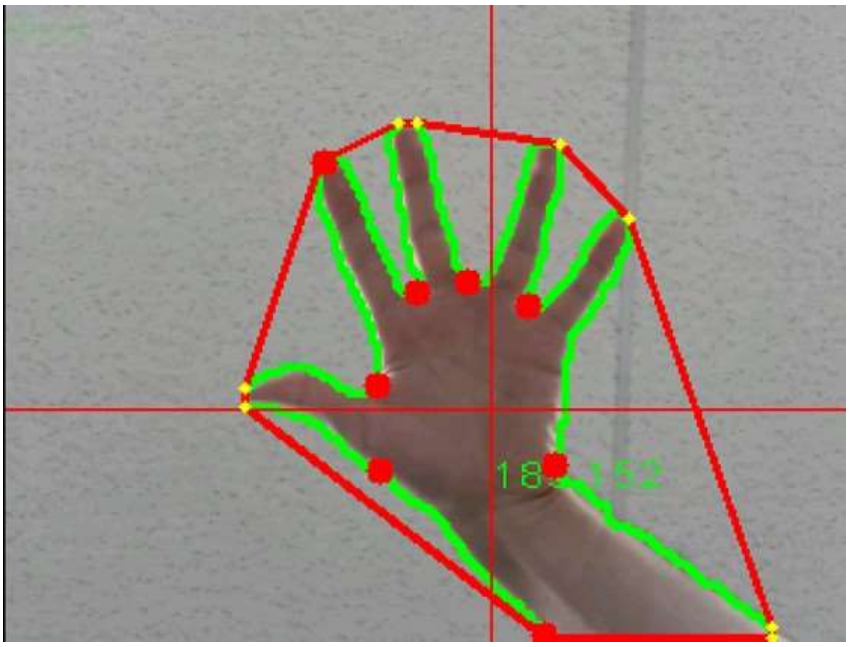
First Model

Our first approach to this problem was to use the contours of the hand to count the number of fingers.

The idea is to calculate the number of spaces between two straight fingers. All the functions mentioned in this paragraph are available in the cv2 python library.
Starting off the algorithm, the threshold of the image is found, to distinguish the hand from the background, then from the threshold the contours of the hand are found. Having all the points that form the contours of the hand, a convex hull is drawn around the points. A convex hull is the minimal convex set containing all the points that form the contours (the hand and its convex hull is shown on the image below). Using the hull and the convex point sets, the convexity defects are found. The algorithm for finding the convexity defects is as follows:

- for each pair (H[i], H[i+1]) of adjacent hull points
- for each contour point (C[n]) that lies between [H[i], H[i+1])
- find the point with the maximum distance from the line joining (H[i], H[i+1])
- store the index of (H[i], H[i+1], max distance point) and the max distance in an array

After running the above algorithm, the result is shown on the image below.



By having a lower bound for the distance between a contour point and a hull point, the points that are not between the fingers (have small distance from the hull) can be filtered out. Counting the remaining points after the filter and adding one to that number gives the number of fingers that are straightened out.

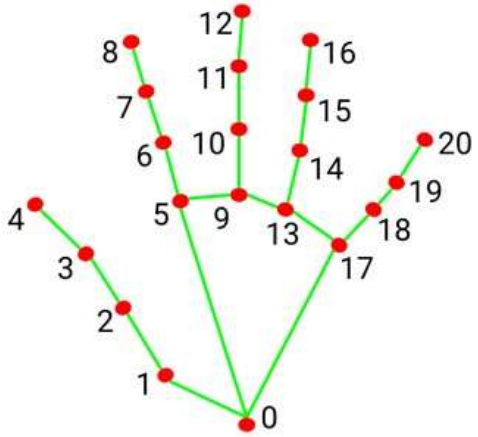
There are a few problems with this solution.

Depending on the rotation of the palm, the distance between the rightmost point on the image above and the hull can vary. If the hand is rotated too much in the direction of the thumb, the distance between the hull and that contour point may be large enough to pass the filter, adding +1 to the number of counted fingers.

The second problem with this solution is that the fingers must always be apart, there must always be space between them for the algorithm to work.
The background is another drawback, if the background is complicated and dynamic, the threshold algorithm will not work well.

Final model

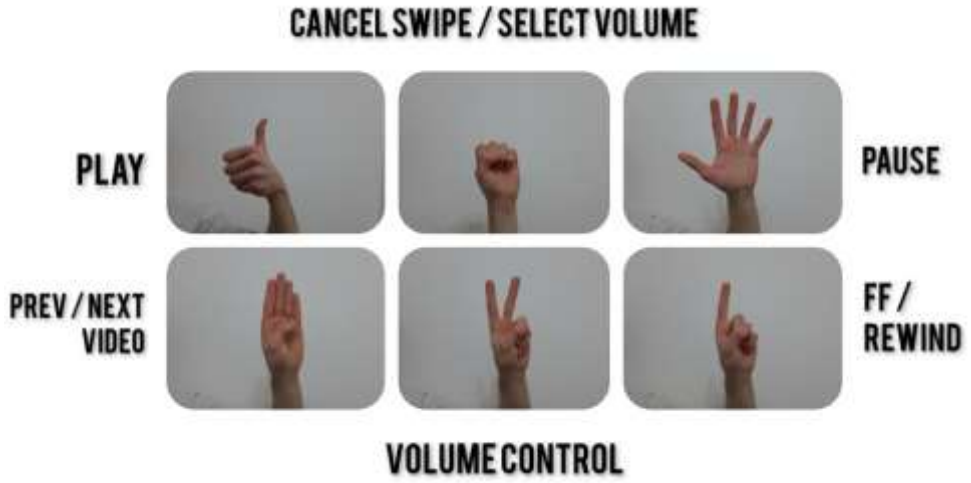
For our final solution we used the python library ‘mediapipe’. Mediapipe is a computer vision library that contains solutions for face recognition, hand detection, body posture recognition and many more. For our project we use the classes and functions that are related to hand detection . With a media pipe from the frame captured by a camera, we get 21 different hand landmarks, each representing a specific point in the hand. The landmarks and its names can be seen in the image below.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

To check if a finger (all fingers except the thumb) is tucked in, the angle between each finger’s _MCP, _PIP, _DIP coordinates is calculated, _PIP being the vertex point, if the angle is acute enough, then the finger is considered to be tucked in, otherwise the finger is considered straight. For the thumb the distance between the tip of the thumb (4th coordinate) and the 9th and 13th coordinates is calculated separately. If one of the distances is smaller than a certain value, the thumb is considered tucked in, otherwise the thumb is considered straight. To get the orientation of the palm, we check whether the y coordinate of the 17th landmark is below the y coordinate of the 1st landmark, if this is the case, the palm is in a horizontal position, otherwise it is in a vertical position.

The possible gestures and the instruction/s each represents are given below.



'PLAY' and 'PAUSE' are static gestures, when the program recognizes them, it sends the media player instructions to play or pause the current video. When the program recognizes any of the other gestures (except the fist), it goes into a 'DOING' state, since these gestures have multiple stages. To exit this state, one must remove the hand from the camera view or do a fist gesture.

The 'FF/REWIND' and 'PREV/NEXT VIDEO' gestures represent 2 instructions. By doing any of the two, the program goes into its 'DOING' state, next a swipe is required. For the swipe to work, every n frames the x coordinate (from the 2d view of the frame) of the tip of the index finger (8th coordinate) is compared to the previously taken x coordinate of the same finger, if the distance between them is large enough, then a left or right swipe is executed, depending on the direction of the motion of the hand, on the contrary if the length is not large enough, the previous value for the index finger takes the current value of the same finger. The hand can take any form to execute the swipe, if the user decides to cancel the swipe, he can remove the hand from the screen or do a fist. For a 'left swipe' the returned command is 'PREV VIDEO'/'REWIND' and for a 'right swipe' the returned command is 'NEXT VIDEO'/'FAST FORWARD' (depending on the previously done gesture).

For the volume control gesture, the instant it is recognized and the program goes into its 'DOING' state, the x coordinate of the index finger is taken. Every next frame the current value of the x coordinate of the index finger is subtracted from the previously taken x value of the same finger, this value is stored in the command 'VOLUME {num}' and returned. When the user is satisfied with the volume, he should do a fist or remove the hand from the screen to set the volume.

References

1. https://google.github.io/mediapipe/solutions/hands.html
2. https://arxiv.org/pdf/2006.10214.pdf
3. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
4. https://developers.arcgis.com/python/guide/how-ssd-works/
5. https://developers.arcgis.com/python/guide/how-retinanet-works/
6. https://www.youtube.com/watch?v=py5byOOHZM8
7. https://en.wikipedia.org/wiki/Convolutional_neural_network
8. https://www.sas.com/en_us/insights/analytics/machine-learning.html#:~:text=Machine%20learning%20is%20a%20method,decisions%20with%20minimal%20human%20intervention.
9. https://en.wikipedia.org/wiki/Machine_learning
10. https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network#:~:text=Deep%20learning%20is%20a%20type,gain%20certain%20types%20of%20knowledge.&text=While%20traditional%20machine%20learning%20algorithms,of%20increasing%20complexity%20and%20abstraction.
11. https://developer.nvidia.com/deep-learning
12. https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/