HAND SIGNALS WITH VISUAL DATA

HAND SIGNALS WITH VISUAL DATA

BY
AYUSHYA AMITABH
CHRIS PANICAN
GERRY XU
OMAR ELNAGDY

ABSTRACT

Our project was to create a software to allow for gesture-based controls on the end user's computer with the aid of deep learning to help recognize the user's hand as well as the number of fingers being held up by the user.

THE NEED FOR HAND GESTURES

The use of hand gestures has been a common place in many futuristic movies – a thing of fantasy that only CGI could achieve. This, however, is no longer true – with the recent developments in AI we can now works towards realizing the dream of controlling our computer with effortless hand gestures without having to lay hands on the computer itself.

AGENDA

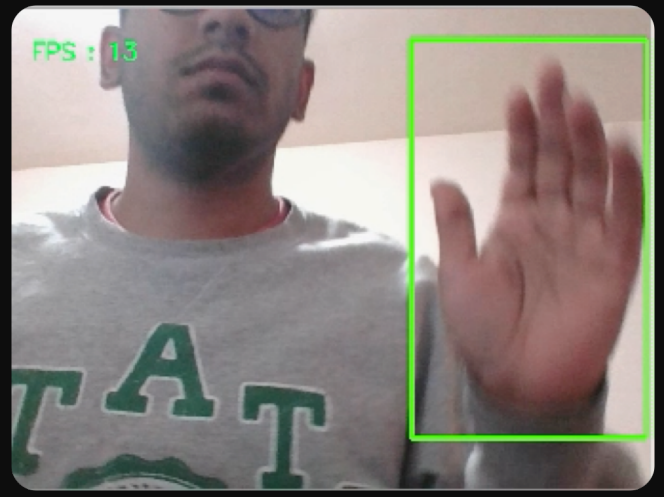
- CHALLENGES
- FEATURE SPECIFICATION
- TECHNOLOGIES USED
- ARCHITECTURE
- DEEP LEARNING
- DEMO

CHALLENGES

CHALLENGES

DYNAMIC BACKGROUND

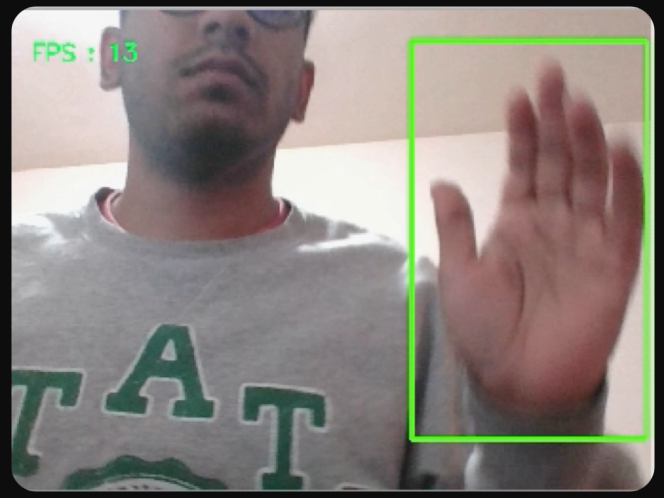
Our biggest struggle towards the end of the project was combining gesture detection with a dynamic background. We did get hand detection with a dynamic background using basic object detection models.



CHALLENGES

DYNAMIC BACKGROUND

The implementation of the dynamic background however, conflicted with our implementation of gesture detection. Background subtraction was hard without image depth information. Restructuring would require a complete retraining our gesture detection model.



CHALLENGES

DATASET GENERATION

Our next biggest challenge was dataset generation which eventually led to data overfitting. Because we were manually generating data from our webcams we were unable to properly cover all possible hand gestures and even covering transformations of a single gesture such as rotated, reflected, scaled and so on.

Detected as Palm

Not detected as Palm

CHALLENGES

OTHER CHALLENGES

Other challenges our group faced:

1. Motion detection – currently we resorted to simply comparing the current and last detected gesture to combine into a gesture
2. Optimization – because of limited time and resources we were unable to test on multiple platforms or even ensure low power usage.

FEATURE SPECIFICATION

FEATURE SPECIFICATION

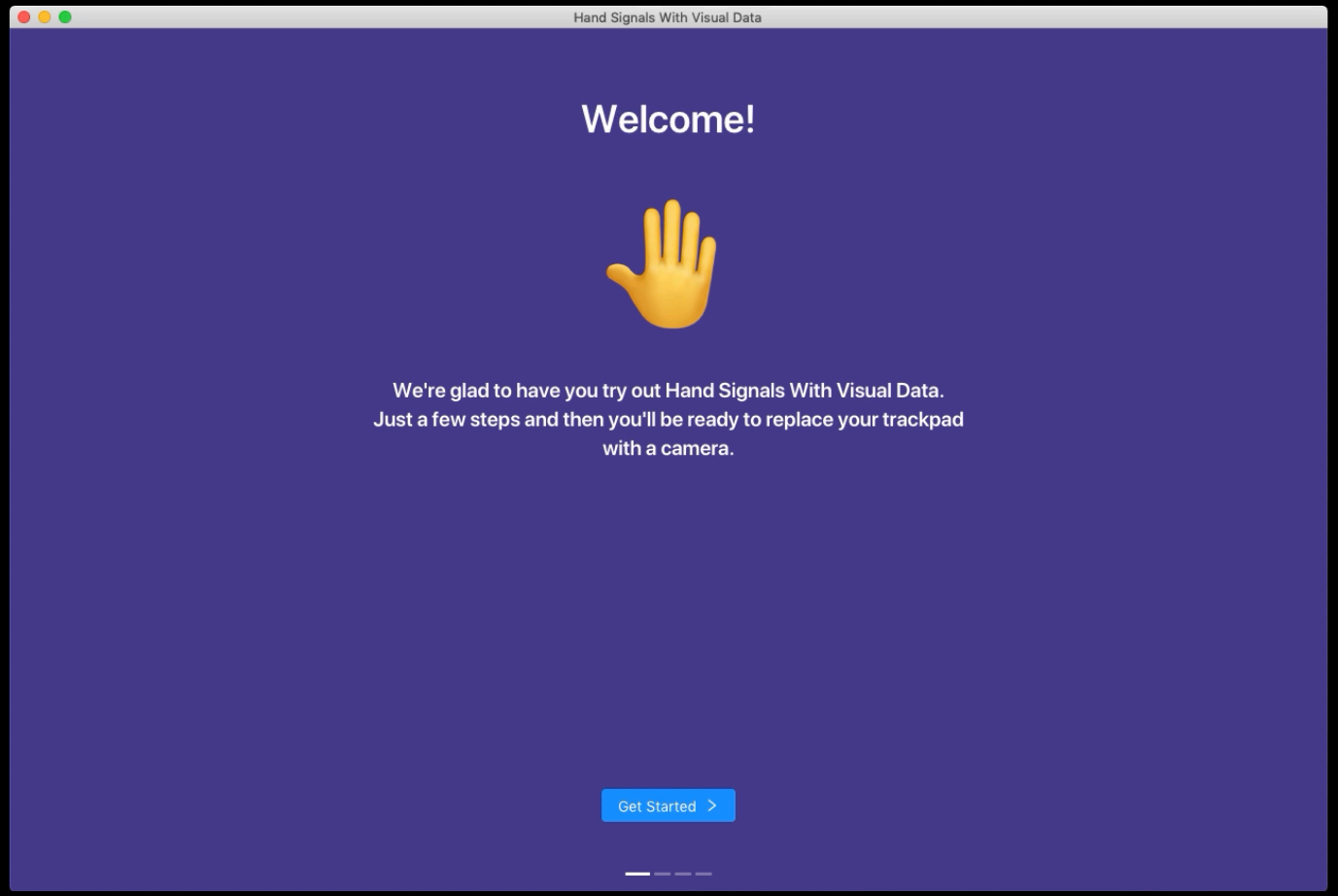
Our completed project had the following feature set:

1. Client Side App – with user onboarding, model downloading, starting gesture predictor, and some basic permission checking
2. Server Side – allows us as developers to upload new trained models, the user can then download this
3. Action Trigger – trigger keyboard and mouse events based on gestures
4. Gesture Recognition with static backgrounds (discussed later)

HAND SIGNALS WITH VISUAL DATA

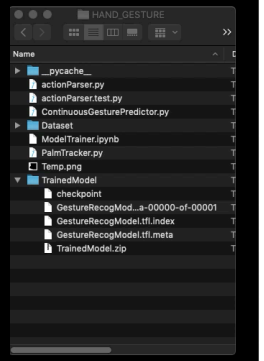
FEATURE SPECIFICATION

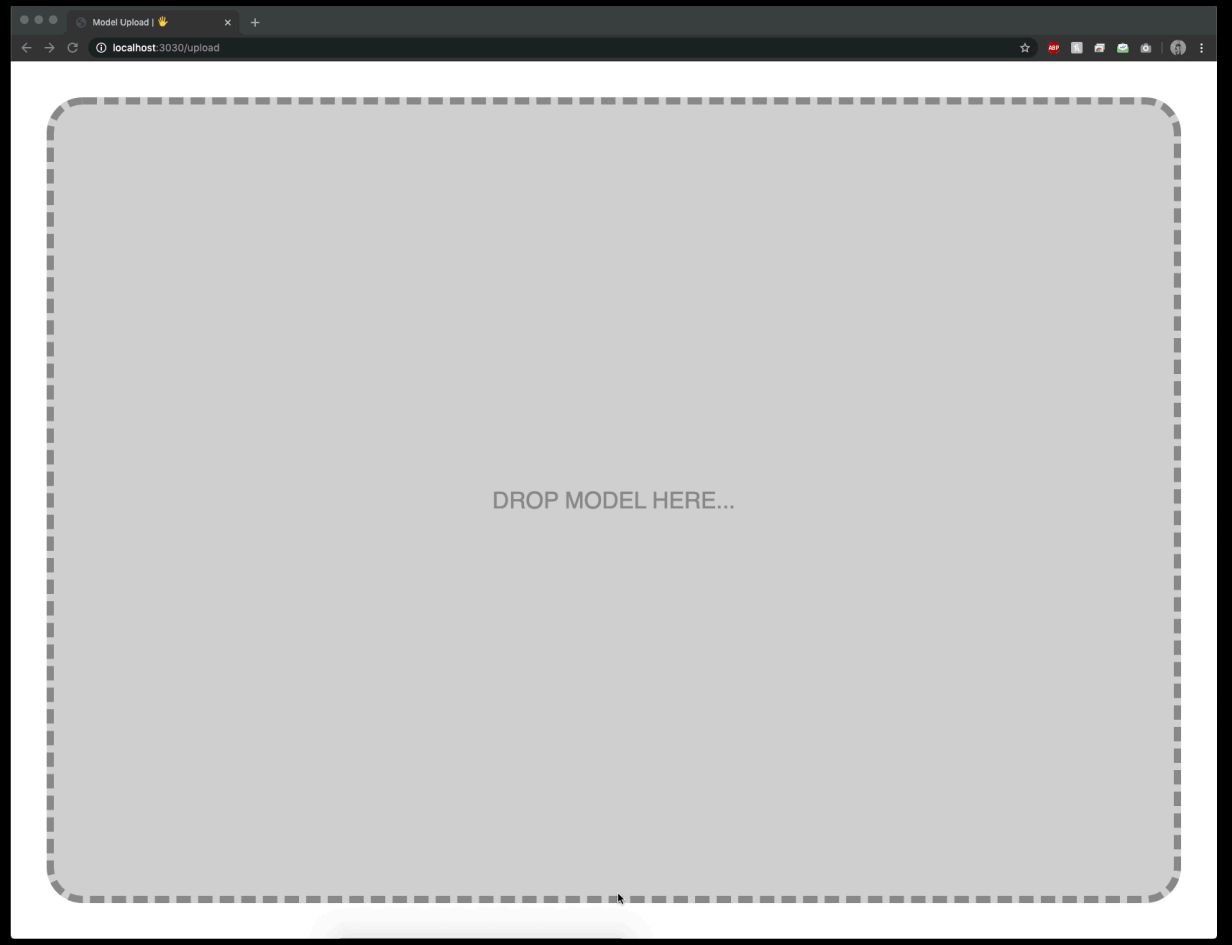
CLIENT SIDE APP



FEATURE SPECIFICATION

SERVER SIDE

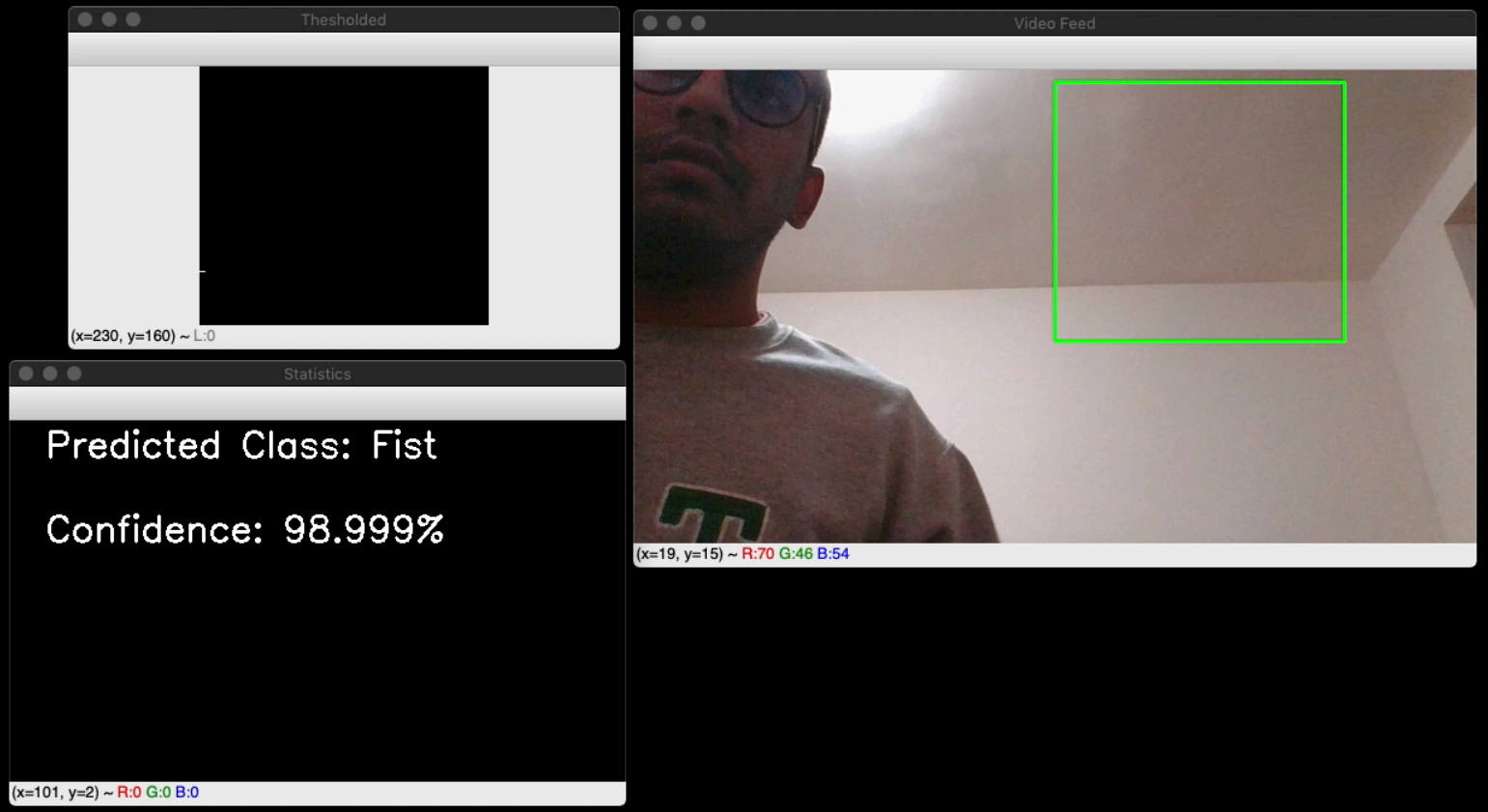




HAND SIGNALS WITH VISUAL DATA

FEATURE SPECIFICATION

ACTION TRIGGER



TECHNOLOGIES USED

TECHNOLOGIES USED

Let’s break down each feature set, to see what they’re each made of

1. Client Side App
2. Gesture Recognition (discussed in depth later)
3. Action Trigger

TECHNOLOGIES USED

CLIENT SIDE APP

The client side app is built using the Electron JS framework – this allows for a single code base for cross platform app. This way the app can be built once and used on any platform.

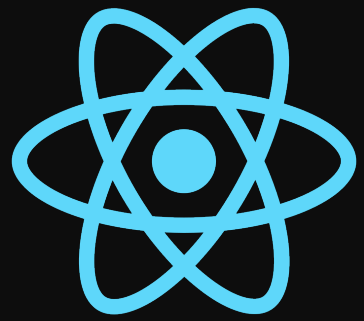
TECHNOLOGIES USED

CLIENT SIDE APP

Node JS

Handles system to app communication, and app to deep learning predictor

React JS



Renders UI

TECHNOLOGIES USED

CLIENT SIDE APP TO GESTURE PREDICTOR

→ Node JS

TECHNOLOGIES USED

CLIENT SIDE APP TO GESTURE PREDICTOR

Node JS → Python Scripts → TensorFlow API

Our script loads our stored model and then begins continuous predictions until it is interrupted or stopped

TECHNOLOGIES USED

CLIENT SIDE APP TO GESTURE PREDICTOR

→ TensorFlow API → Action Parser

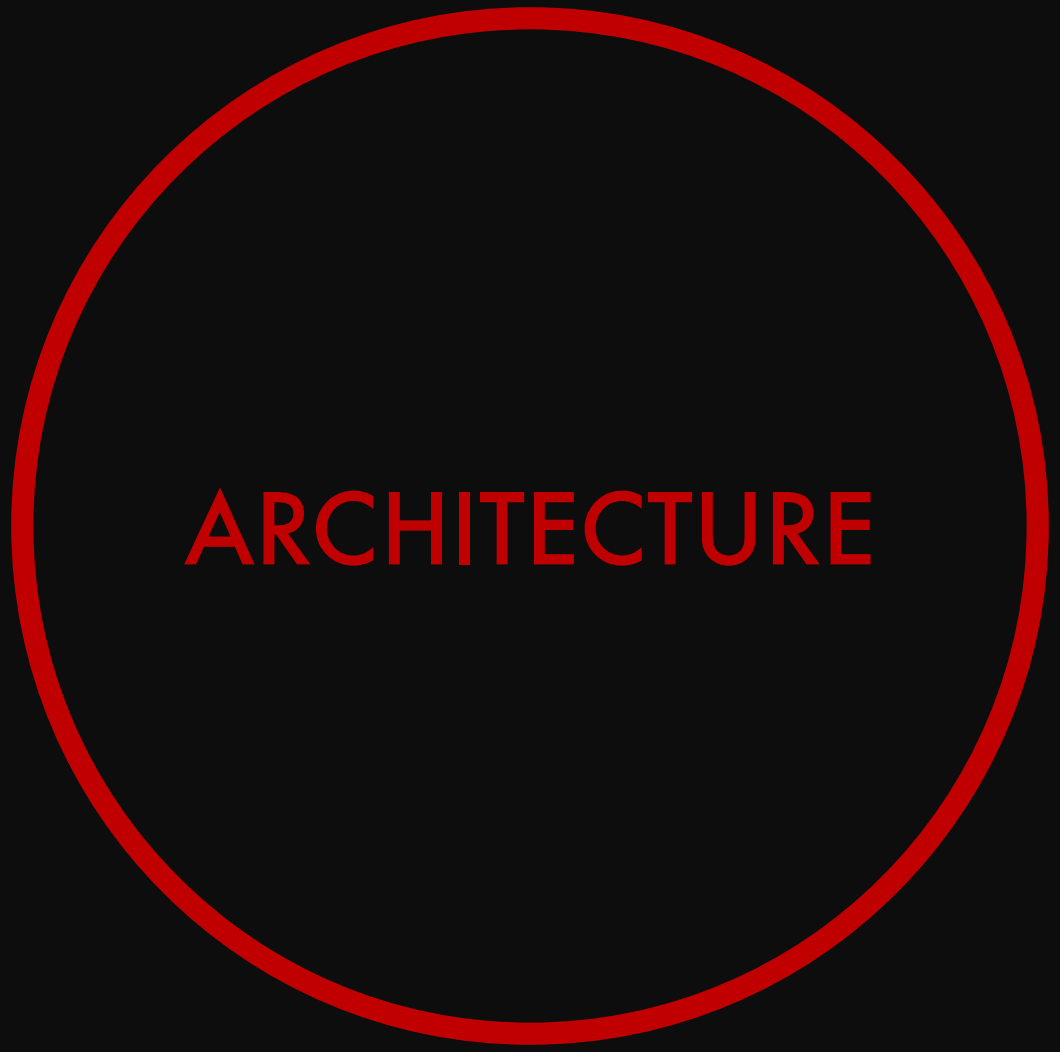
This is a custom class that abstracts a program called “cliclick”, using a custom Python dictionary that relates gestures to key event triggers

TECHNOLOGIES USED

ACTION PARSER

This is a custom class that abstracts a program called “cliclick”, using a custom Python dictionary that relates gestures to key event triggers

```
class actionParser:

    def __init__(self, commandDictionary, allowContinuous = False, waitTime = 0.5):
        """
        @param commandDictionary - Dictionary of command name and key combinations

        @param allowContinuous   - Allows for continuous input and doesn't check for last input. Defaults to False.

        @param waitTime          - Wait time value for when allowContinuous is False.

        ----------------------------------------------------------------------------------------------------

        Examples of commandDictionary

            "ActionName1" : "arrow-up",                             - single KEY

            "ActionName2" : [ "ctrl", "arrow-right" ],              - 1 MODIFIER + 1 KEY

            "ActionName3" : [ "cmd", "shift", "arrow-up", "del" ]   - x MODIFIER + x KEY

        """
```

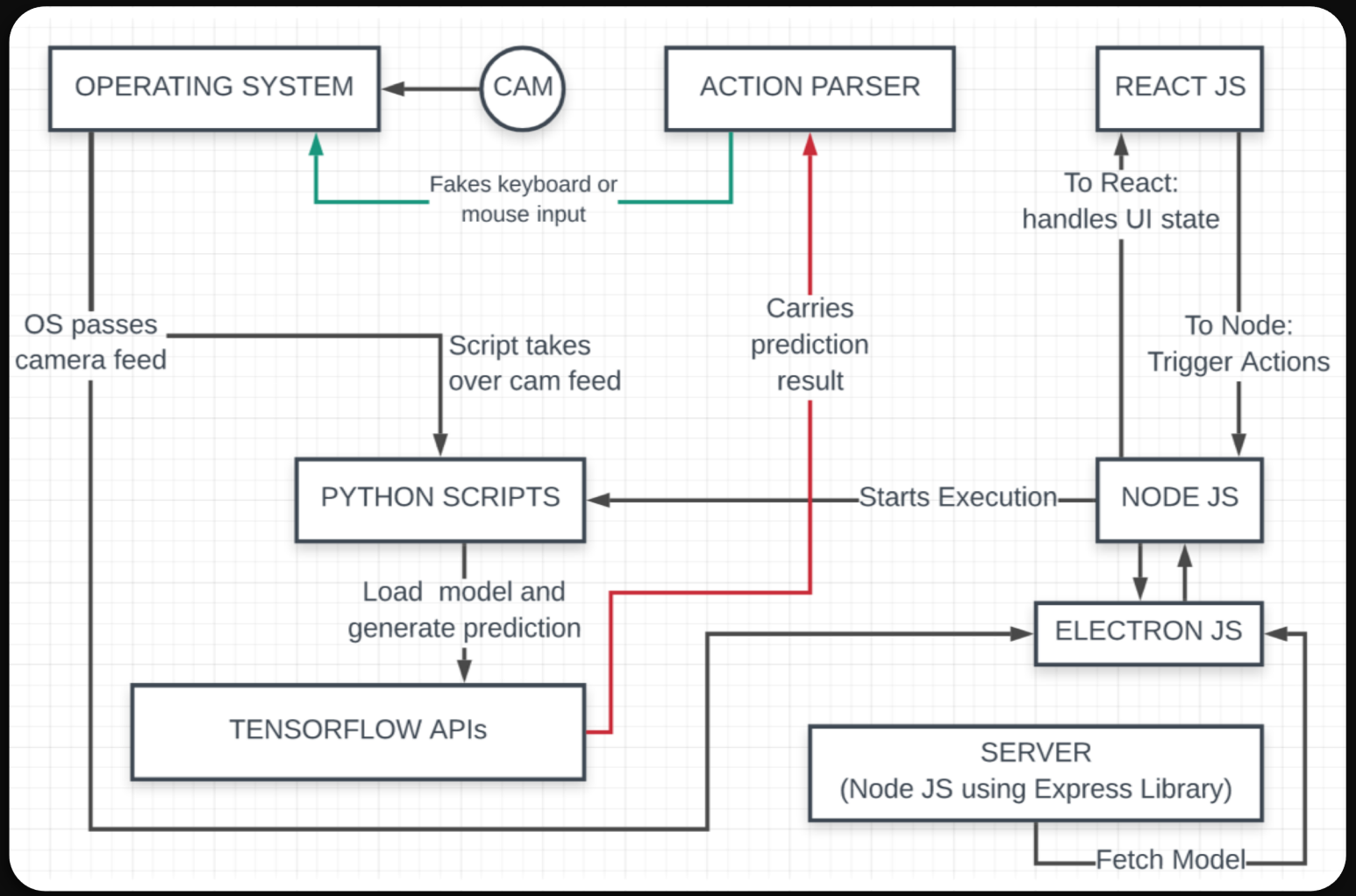


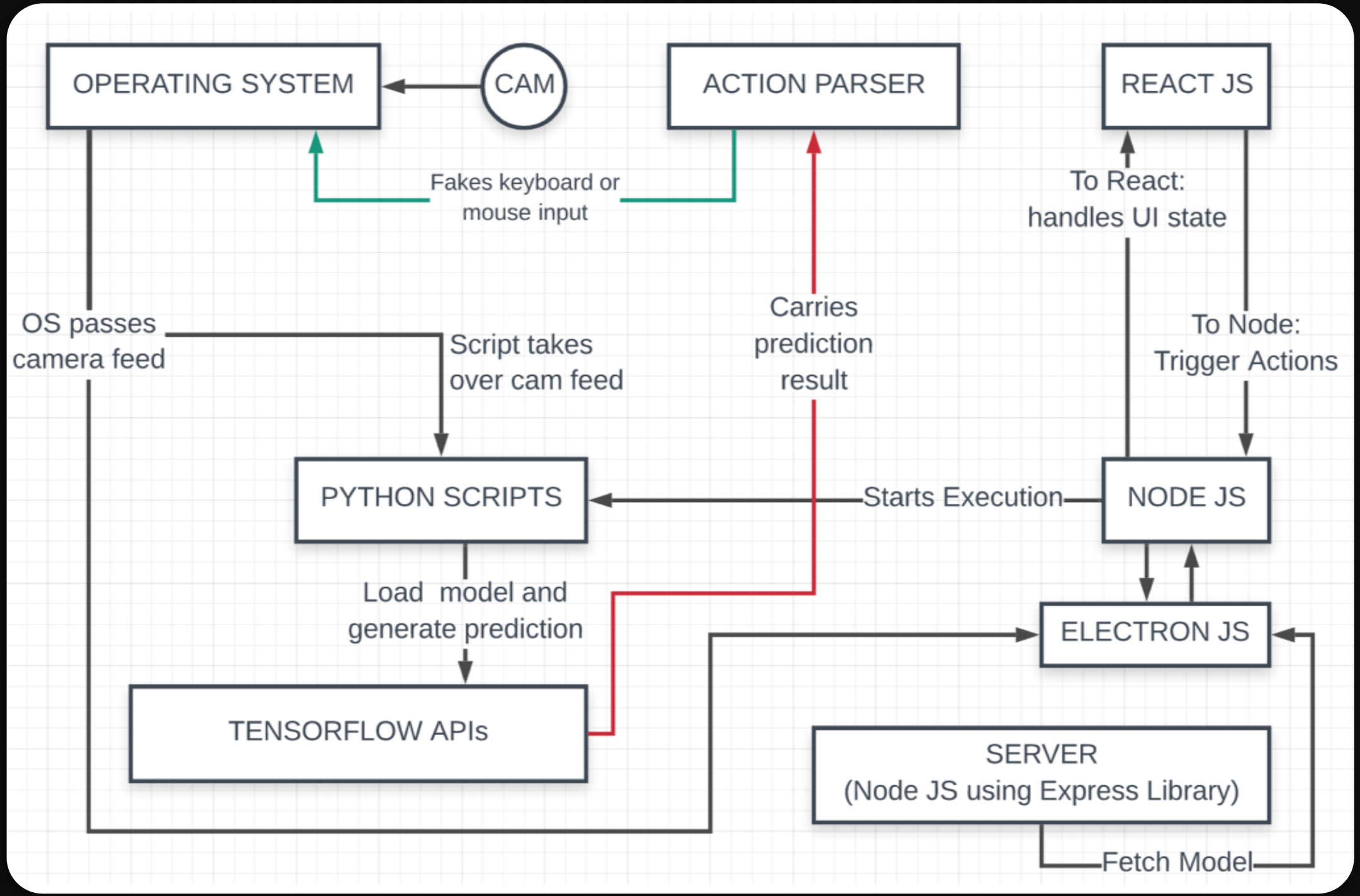
DEEP LEARNING DEMO

ARCHITECTURE

Now that we've see the individual parts,
let's see how it all comes together

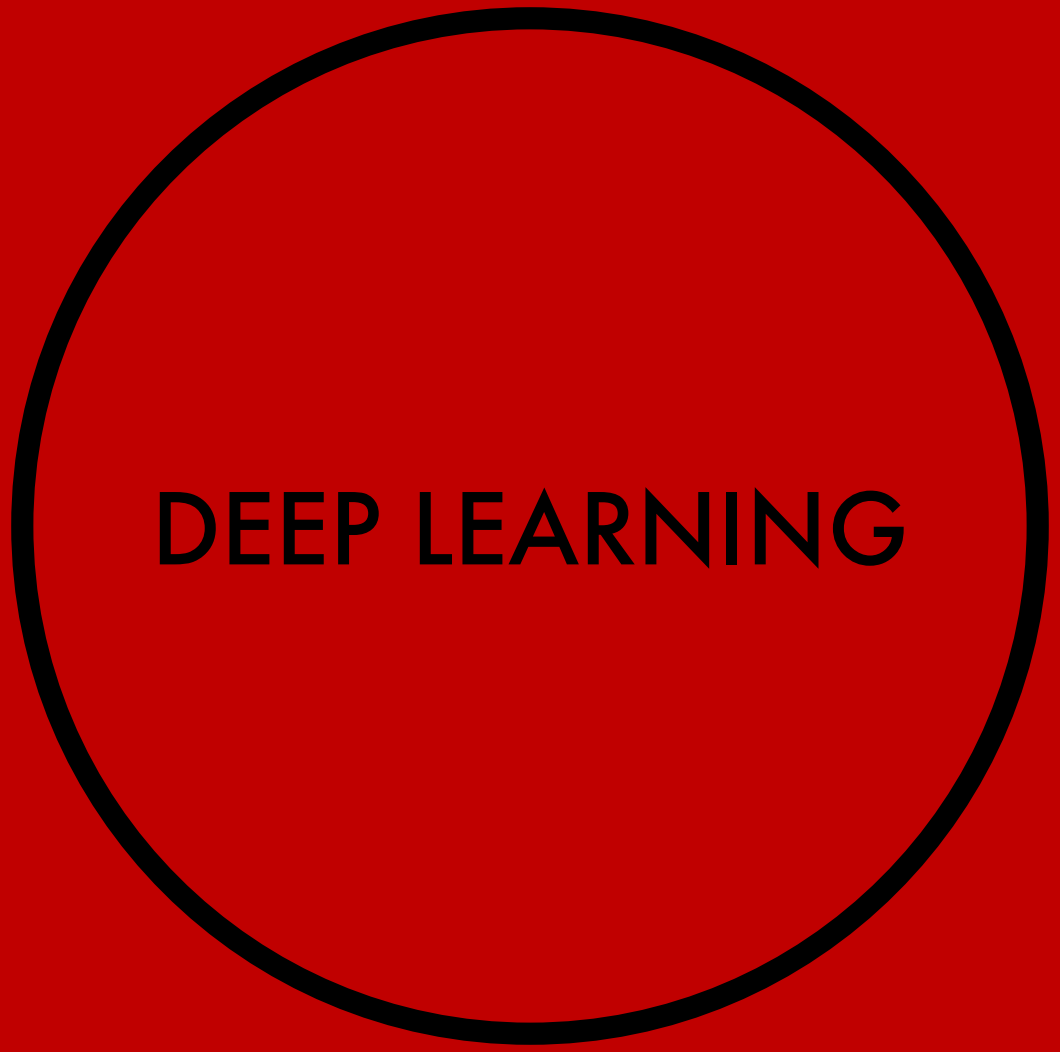
ARCHITECTURE




OPERATING SYSTEM
CAM
ACTION PARSER
REACT JS
Fakes keyboard or
mouse input
To React:
handles UI state
OS passes
camera feed
Script takes
over cam feed
Carries
prediction
result
To Node:
Trigger Actions
PYTHON SCRIPTS
Starts Execution
NODE JS
Load model and
generate prediction
ELECTRON JS
TENSORFLOW APIs
SERVER
(Node JS using Express Library)
Fetch Model

ARCHITECTURE

With all the other parts of the feature sets covered, let's now dive into the deep learning details of the project



DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

In our project, we will utilize deep learning by gathering visual data from a webcam and use that data to trigger commands in the user's machine. Our group picked CNN because of its ability to process images. Other neural networks, such as RNN, cannot excel in this task.

DEMO

HAND SIGNALS WITH VISUAL DATA

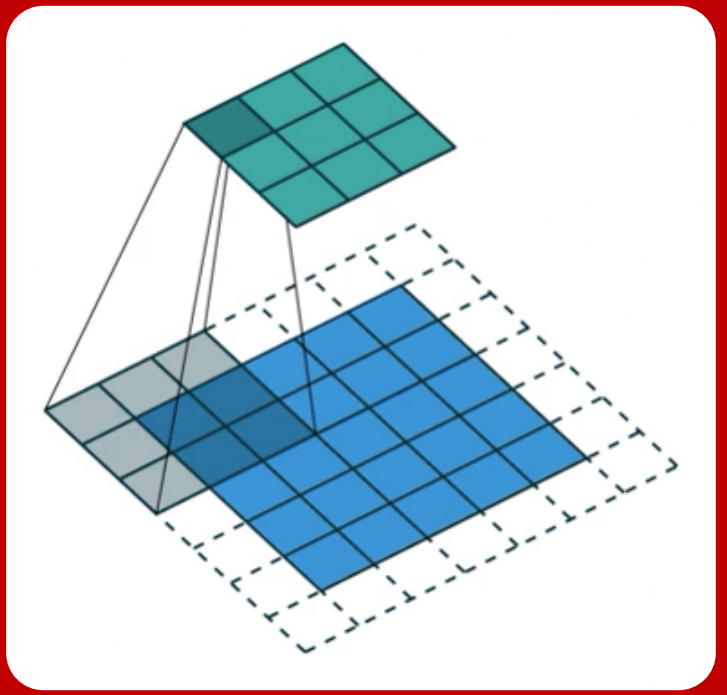
DEEP LEARNING

WHY CONVOLUTIONAL NEURAL NETS

CNNs can be thought of automatic feature extractors from the image.

A CNN effectively uses adjacent pixel information to effectively down-sample the image first by convolution and then uses a prediction layer at the end.

Prior to CNN, algorithm-based image classification required manual feature extraction.



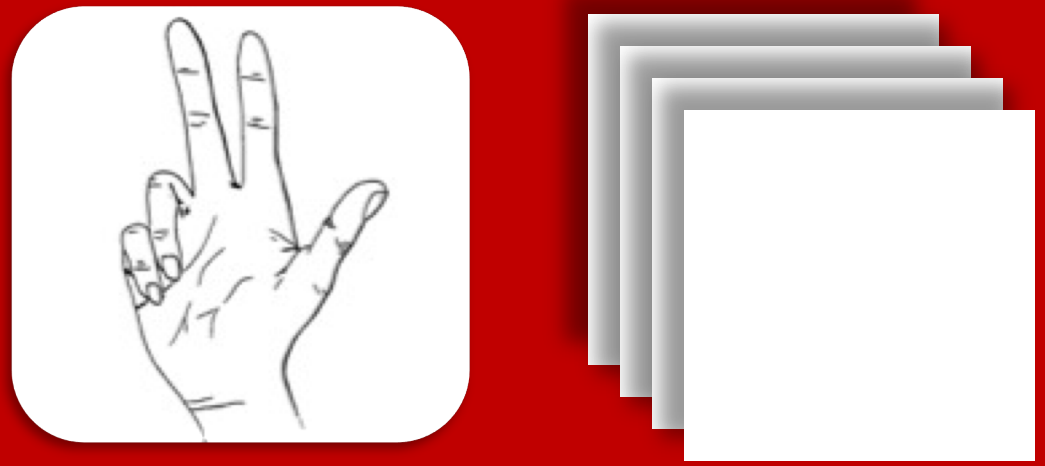
DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

BUILDING BLOCKS OF CONVOLUTIONAL NEURAL NETS

Convolution – receiving input signals from another layer and using filters/kernels to convolve across the input. This is where the network “learns” important filters as it runs.



INPUT IMAGE TO FEATURE MAPS

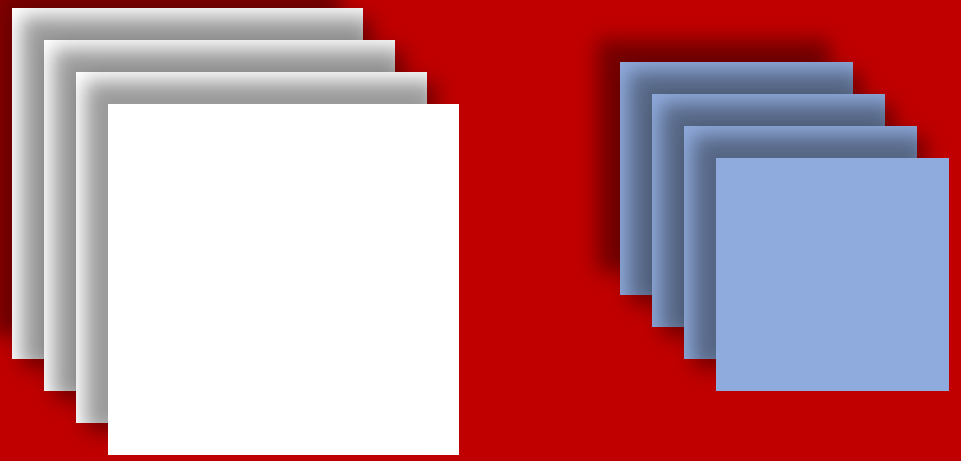
DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

BUILDING BLOCKS OF CONVOLUTIONAL NEURAL NETS

Subsampling – also known as pooling layer. It reduces the dimensions of data with a filter size and a stride and also controls overfitting. It is usually inserted after a convolution.



SUBSAMPLING FROM FEATURE MAPS

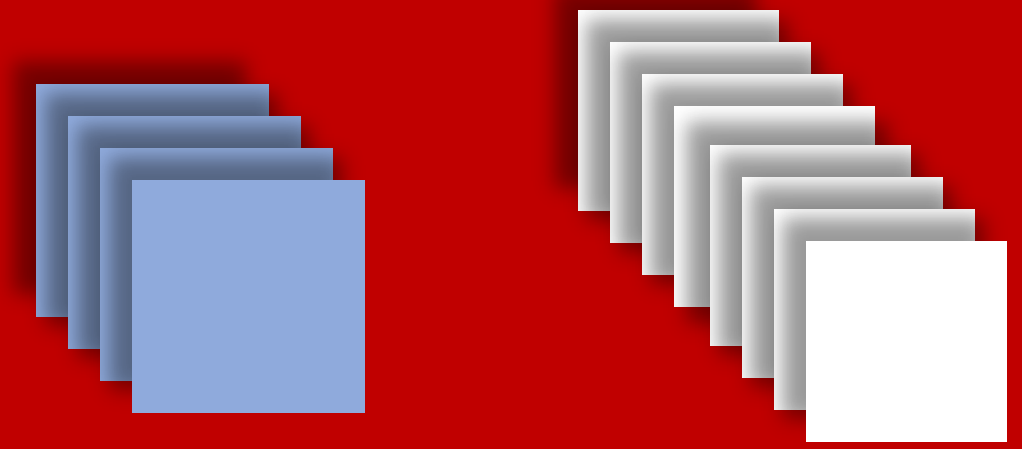
DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

BUILDING BLOCKS OF CONVOLUTIONAL NEURAL NETS

Subsampling – also known as pooling layer. It reduces the dimensions of data with a filter size and a stride and also controls overfitting. It is usually inserted after a convolution.



CONVOLVE POOLED LAYERS

DEMO

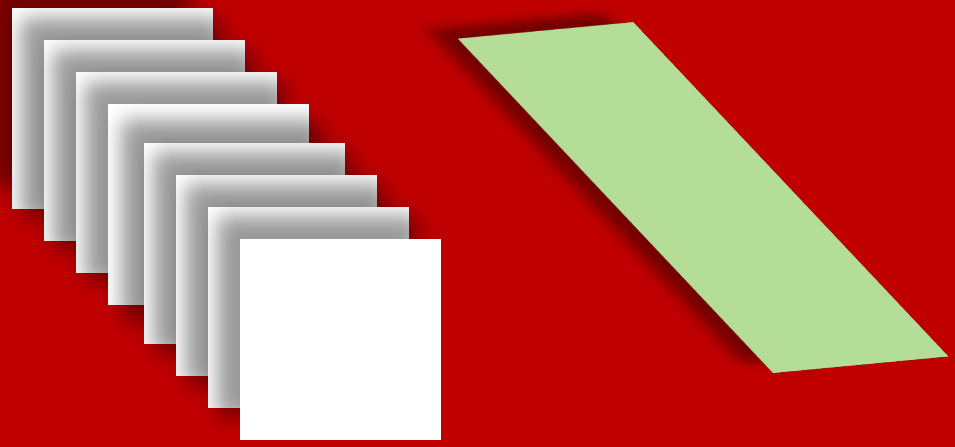
HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

BUILDING BLOCKS OF CONVOLUTIONAL NEURAL NETS

Activation – controls how information flows from one layer to another.

ReLU stands for rectified linear unit and is a type of activation function. Simply, allows activation when filter is outputs a positive value.



FLATTEN AND FULLY CONNECT
USING ReLU ACTIVATION

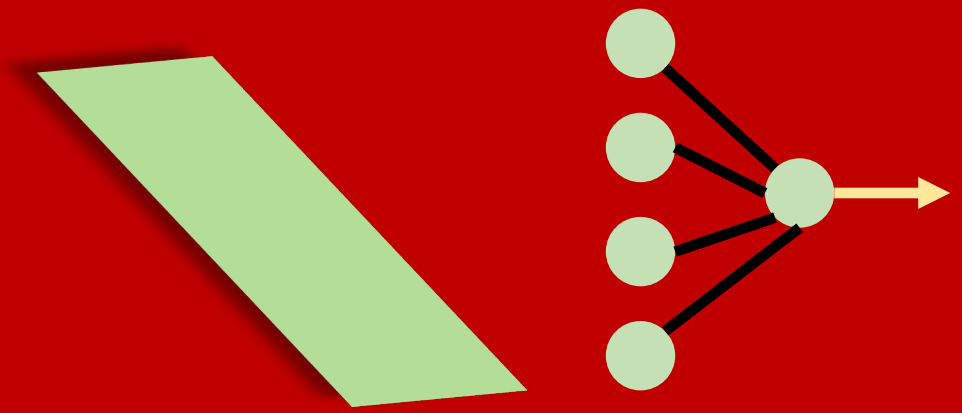
DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

BUILDING BLOCKS OF CONVOLUTIONAL NEURAL NETS

Fully connected – connects all neurons in a layer from its previous layer to its sub-layers.



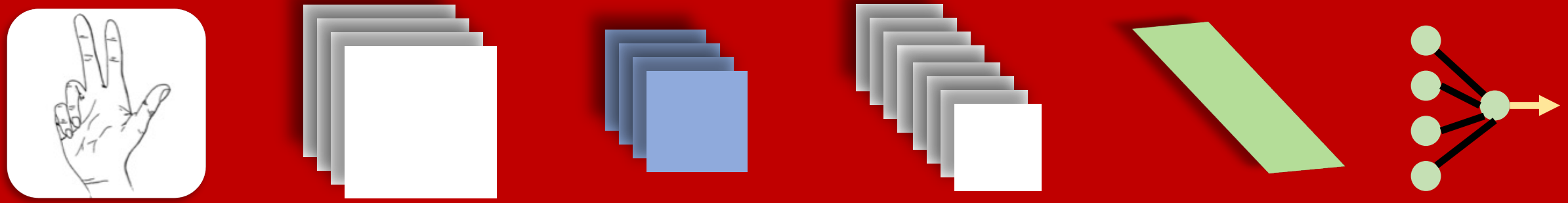
RESULT IS A FULLY CONNECTED NEURAL NET – READY TO MAKE PREDICTIONS

DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

CONVOLUTIONAL NEURAL NET PUT TOGETHER

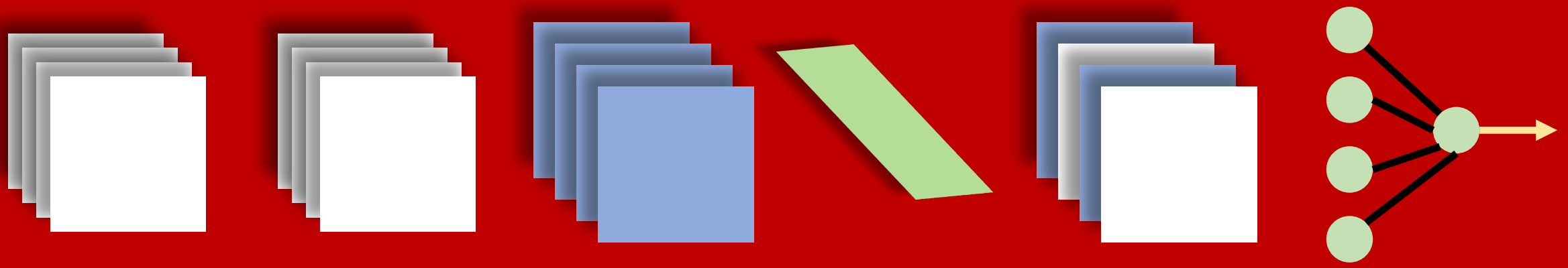


DEMO

HAND SIGNALS WITH VISUAL DATA

DEEP LEARNING

OUR CNN MODEL



CONVOLUTION + ACTIVATION (32,198,198)

CONVOLUTION + ACTIVATION (32,196,196)

POOLING & DROPOUT (32,98,98)

FLATTEN (307328)

DENSE, ACTIVATION & DROPOUT (128)

FULLY CONNECTED (5)

DEMO

DEMO

HAND SIGNALS WITH VISUAL DATA

Q&A

HAND SIGNALS WITH VISUAL DATA

HAND SIGNALS WITH VISUAL DATA