Detail Project Report
Hand Gesture Recognition Task.

Github Link Video Link

❖ Introduction:

- Goal was to control a Mountain Car from Open Gym, using Webcam, using hand gestures. This car can be controlled in three ways,
  - 0 : Accelerate to the Left
  - 1 : Don't accelerate (No Brake)
  - 2 : Accelerate to the Right

- So To solve this, I treated this problem as Image CLassification for the above three classes having three different actions.

- An approach was to collect a dataset for three different classes 0,1 & 2 and develop an algorithm to classify them. During this task, what challenges were faced and what optimal solutions to them were discussed.

❖ Short Summary:

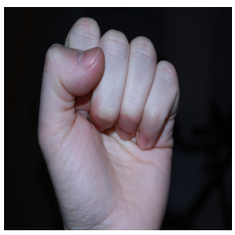
Device: CPU

Model Name	Accuracy on Real time test set	FPS
ResNet-18	92%	9
MobileNetV2	88%	12

Framework: PyTorch.
Metrics Used: Accuracy, Precision, Recall & Confusion Matrix

❖ Dataset:

a. Accessing the webcam via OpenCV, a dataset for 3 three different classes for three actions was created.
b. Following are the actions decided for controlling car:



(0)
(Accelerate Left)

(i)
(Don't Accelerate)

(ii)
(Accelerate Right)

A. Analogy to choose this action:

1. In the above action there is not much overlapping of features. All the features are unique.
2. We get high level features such as edges and boundaries information distinctly.
    a. In, (0) we get some information about nails along with curled fingers (can be noted in CNN layer filters of trained models).
    b. In (i), we get clear information about pam, which has more data points.
    c. In (iii) We get clear edges, also can be visualised in CNN layer filters of trained models.

B. Strategy To collect data:

1. Finalized dataset has been collected with different kinds of backgrounds (simple, cluttered and moderate) with two different hands of different people.
2. Reason to do this was, our goal is to generalise on gestures or shape of hand and not the background or about the particular hand of one person.

C. Risk:

1. It fluctuates in a very cluttered background.
2. During collecting data wrist part in some samples where been recorded, so this is quit risky to have inference on objects present at long distance from camera

D. Challenged During Data Collection:

1. I think dataset should have a variety of situations so as to deal with real time complex cases, so the challenge in this problem was to collect dataset with hard backgrounds. This can be solved using Green Screen strategy. Idea is to collect data in a green background, and then using OpenCV you can easily integrate with any scenes. Well for now, I collected a dataset with different backgrounds.

E. Data Stats: Considering data imbalance problems in future, equal samples for all classes were created.

- 1500 Images per classes
- 75% Training Set.
- 25% Validation Set.

F. Real time testing set - This set was separately collected excluded from the training and validation set. So to get metrics on real time sets.

Dataset was managed according to the folder hierarchy followed by classes names .

❖ Model Development:

Basically there are two ways to solve this problem,

1. Convert Images to Black & White and export edges information using Edge detectors such as Canny and then pass 2D image to CNN to extract features and then to Fully Connected
2. Directly pass RGB image for CNN model to detect features, etc.

My approach was to using (2), just to make our predictions more real time and faster.

To solve this:

Approach 1.

Framework: Keras.

Preprocessing:

Image was been resized into 62,62,1 so has to have required information.

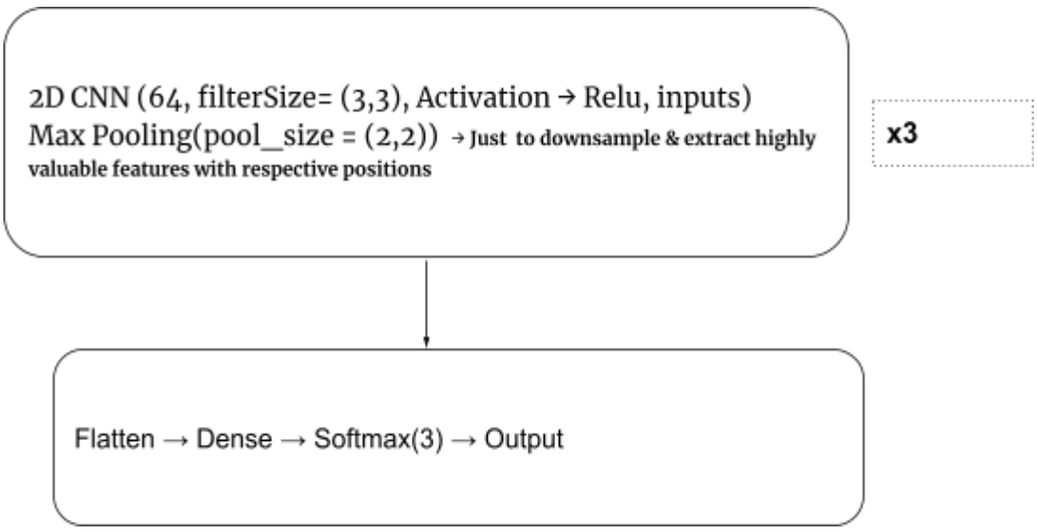
Rotation transform, brightness and contrast Augmentation techniques were used.

Model Development:

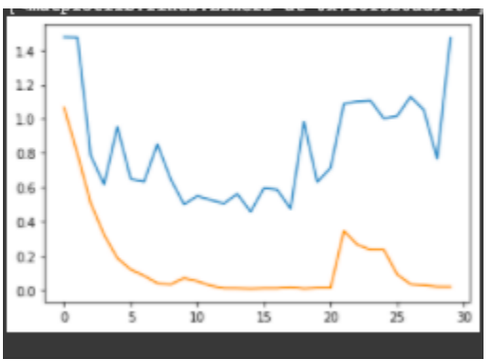
Basic Parameters:

- Dataset: 800 Total Images Per Class (75/25% Split for Training & Validation)
- Evaluation Metrics: Accuracy, Precision, Recall & Confusion Matrix

Initially just to setup a pipeline for training, testing and validation, trained a baseline model with Architecture as:



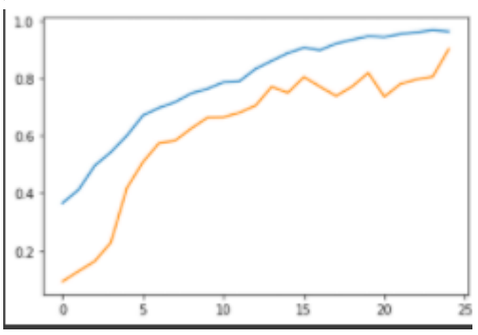
Training model above for a few epochs initially the loss was extremely bad. (During Training And validation phase loss and accuracy was monitored.)



Optimisation:

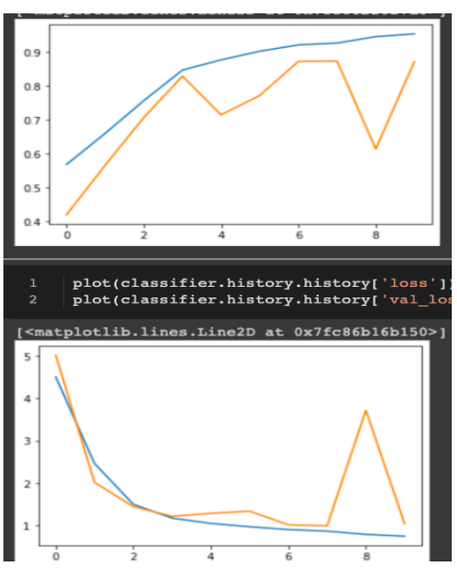
1. To optimise this model, after tuning few parameters changing kernel size to 5,5, (Reason:- As our object is much visible and our features are not so precise)
2. Increasing some CNN layers so as to learn more.
3. Adding Dropout & L2 Regularisation to control losses.

We landed up with Accuracy above 80%.



Here it was observed that training was taking to time so we introduced batch Normalisation to layers to deal with this and helped to converge Faster.

(And it helped) →



After Applying Batch Normalisation and from Above process it was observed that information from initial learned layers was missing, Till certain iterations loss goes well and afterwards, It comes back to the worst values. So here we need some Network to remember previously learned Information And hence I landed up using ResNet-18 pretrained Model.

Approach 2. (Successful)

ResNet -18:

Framework: *PyTorch*.

Preprocessing: Input_size = 224,224,3, random rotation, gaussian blur and other transformation was applied on run time to make training set complex

Dataset: 1500 Total Images Per Class (75/25% Split for Training & Validation)

- Evaluation Metrics: Accuracy, Precision, Recall & Confusion Matrix

***Reasons To Select Pretrained ResNet-18 Model*:**

1. Since we only have 1500 images per class in total, and limited GPU power it's very hard to train models from scratch to converge to global minima.
2. If we know some information about the real world we would get some information to start learning, hence using Transfer Learning on ImageNet weights is a good idea.

3. As we were facing reduction in accuracy after linear increment for some epoch, here we need some architecture to remember initial valuable information learned by initial layers.
4. After plotting CNN feature map, it was observed that the model was learning background well. An increase in feature size reduced this issue, but I thought if we get some reference point in terms of pre-trained weights, it would optimise more.
5. As we don't have complex features, 18 layers would be sufficient to learn.
6. Also I increased the layers of CNN, but then gradients started vanishing.
7. Although, model was showing good accuracy and loss numbers during training and testing but was failing on real world data.

Considering the above cases we landed up using ResNet-18 Pretrained layers.

Initially with random parameters, resnet18 was trained
After fine tuning model with following Hyper parameters:

1. Learning Rate: 0.001 As the object is clearly visible in frame we don't want very precise information. Hence to avoid overfitting we choose this value.
2. Batch_Size : 32 (To be in memory and generalize well!)

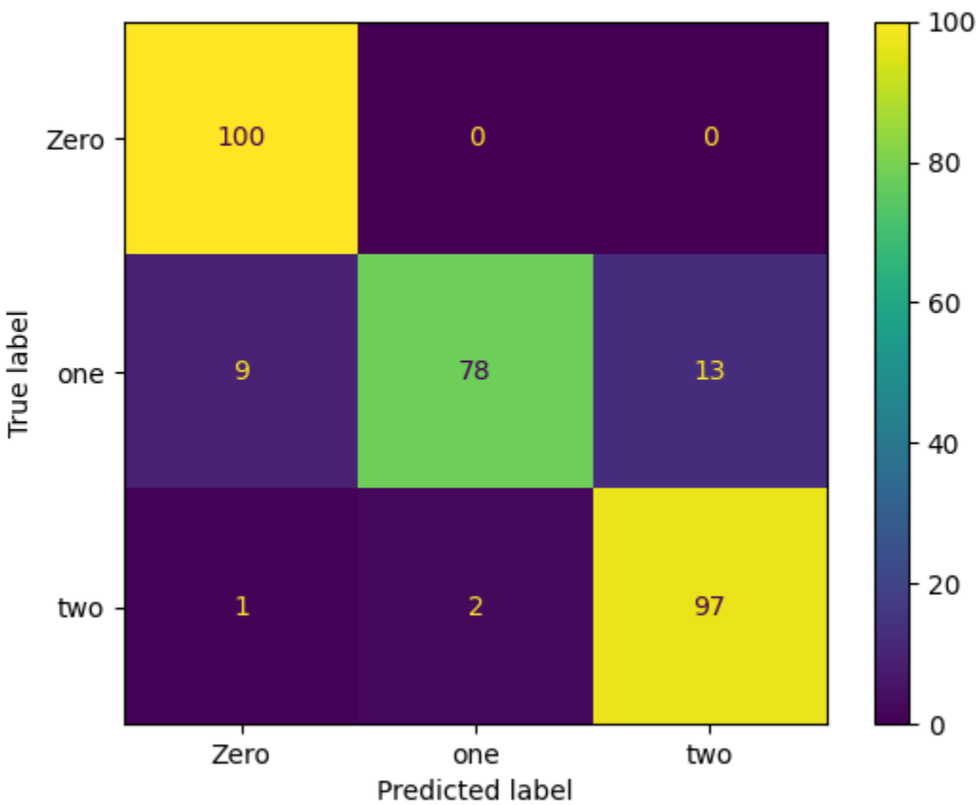
And other parameters.

Why ResNet worked:

1. After plotting a Filter of trained model, required shapes according to our features excluding background information was learned by model.
2. Skip connections helped the model to stay updated and not to lose track.
3. Each Skip connection is followed by [CNN+BatchNormalisation+Relu]
4. As in our chase, instead of bright pixels, we are interested in smooth and sharp pixels. So Average pooling at the end of the network really helps.

Finally after fine tuning and solving some overfitting issues we landed up Accuracy of 97% on Validation set.

Following are results on a Real time test set on 100 Images.

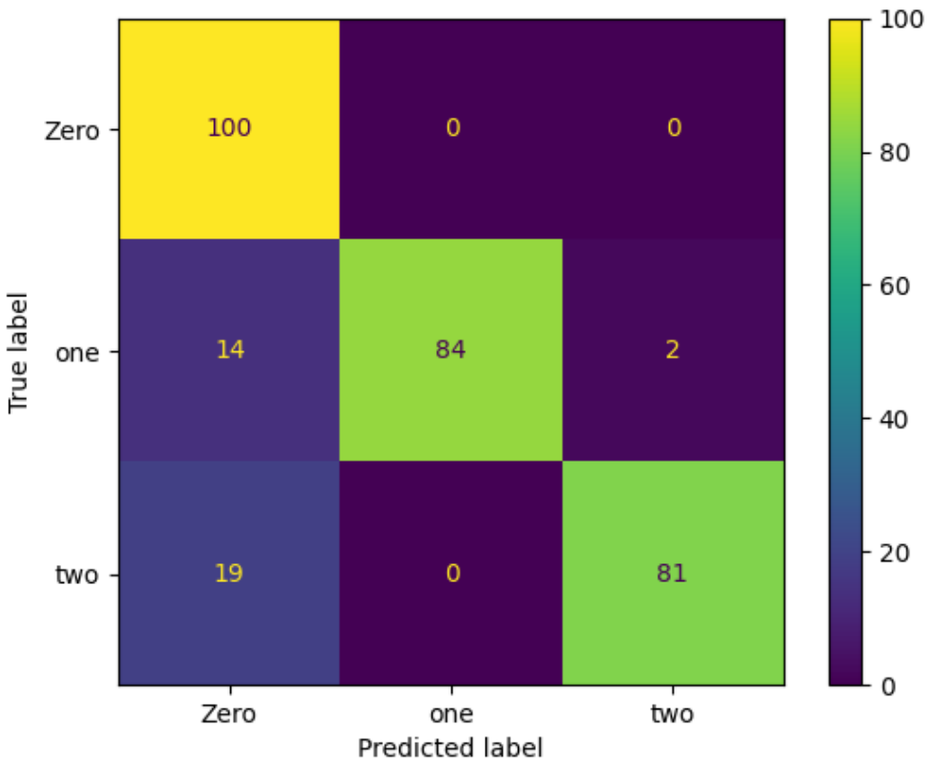


Overall it has Accuracy of 91.67% with precision of 92% (Ref GitHub for more metrics)
FPS: 9 FPS

Similarly, to increase more FPS, I also trained a second model using MobileNetV2.

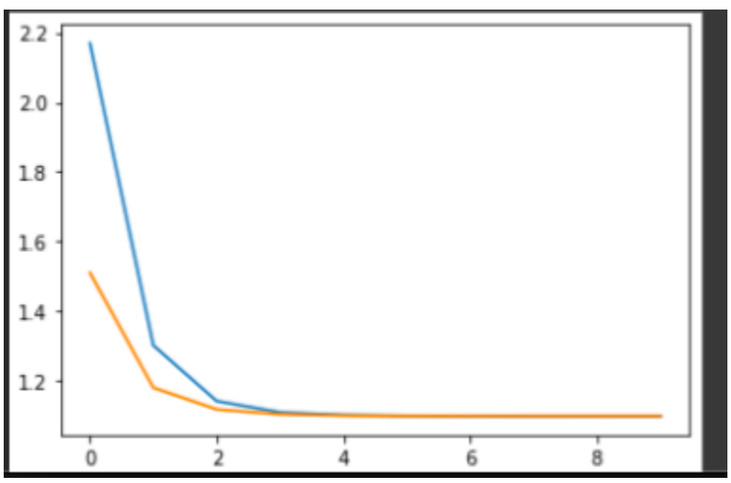
MobileNetV2:

MobileNet Architecture uses Depth Wise Convolutions which helps to increase time.
So we get good FPS, but probably low accuracy.

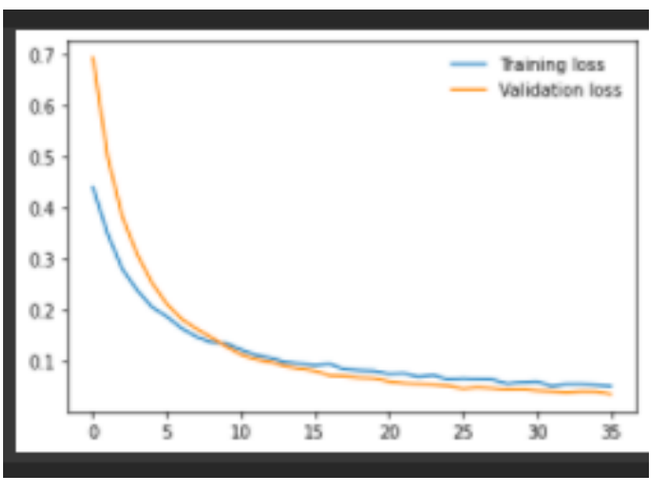


Overall it has Accuracy of 88.33% with precision of 91% (Ref GitHub for more metrics)
FPS: 12 FPS.

ResNet-18



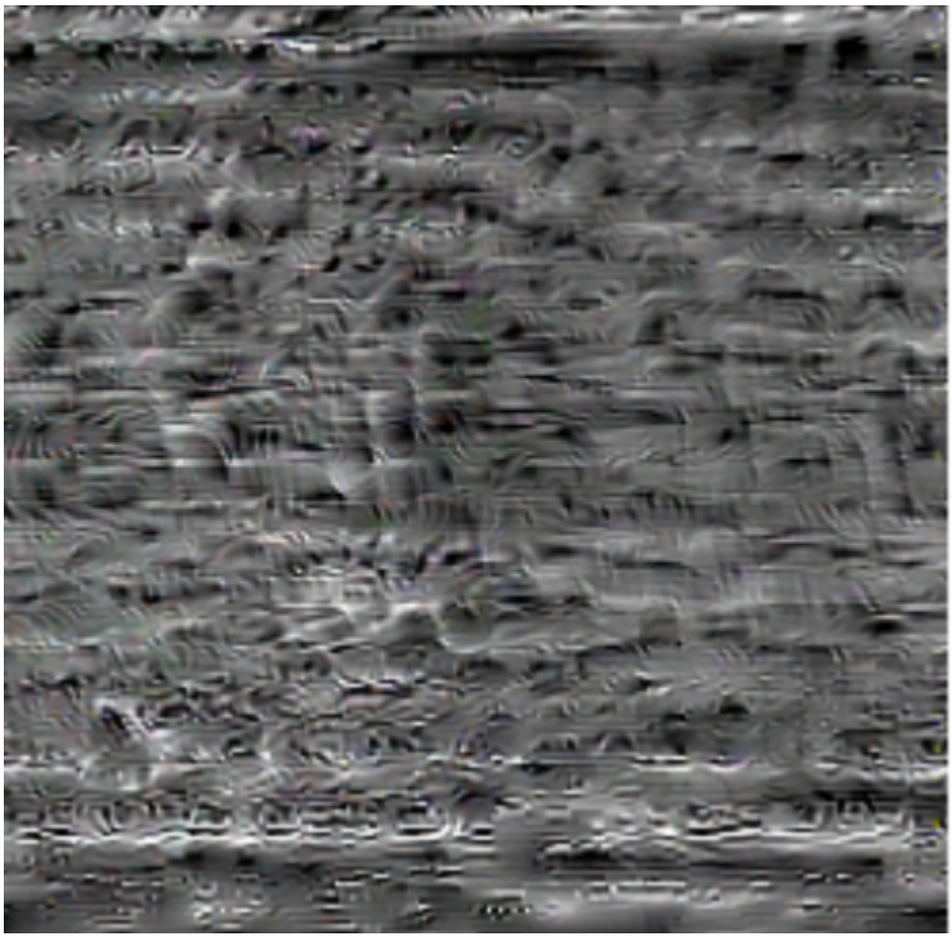
MobileNetV3



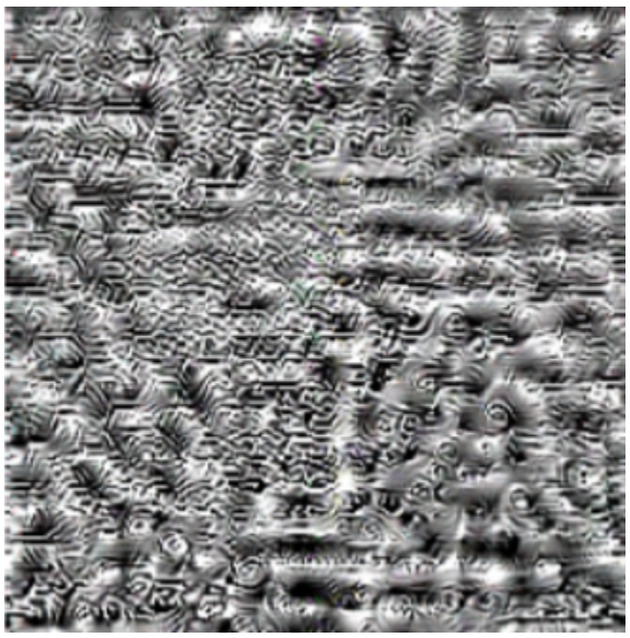
❖ Additional Task :

I just tried this library from official documentation, on random layers from the model.

Here is a visualisation for ResNet's One of the layers. If you can zoom in you may see some related features. (Three Curled Fingers with nails)

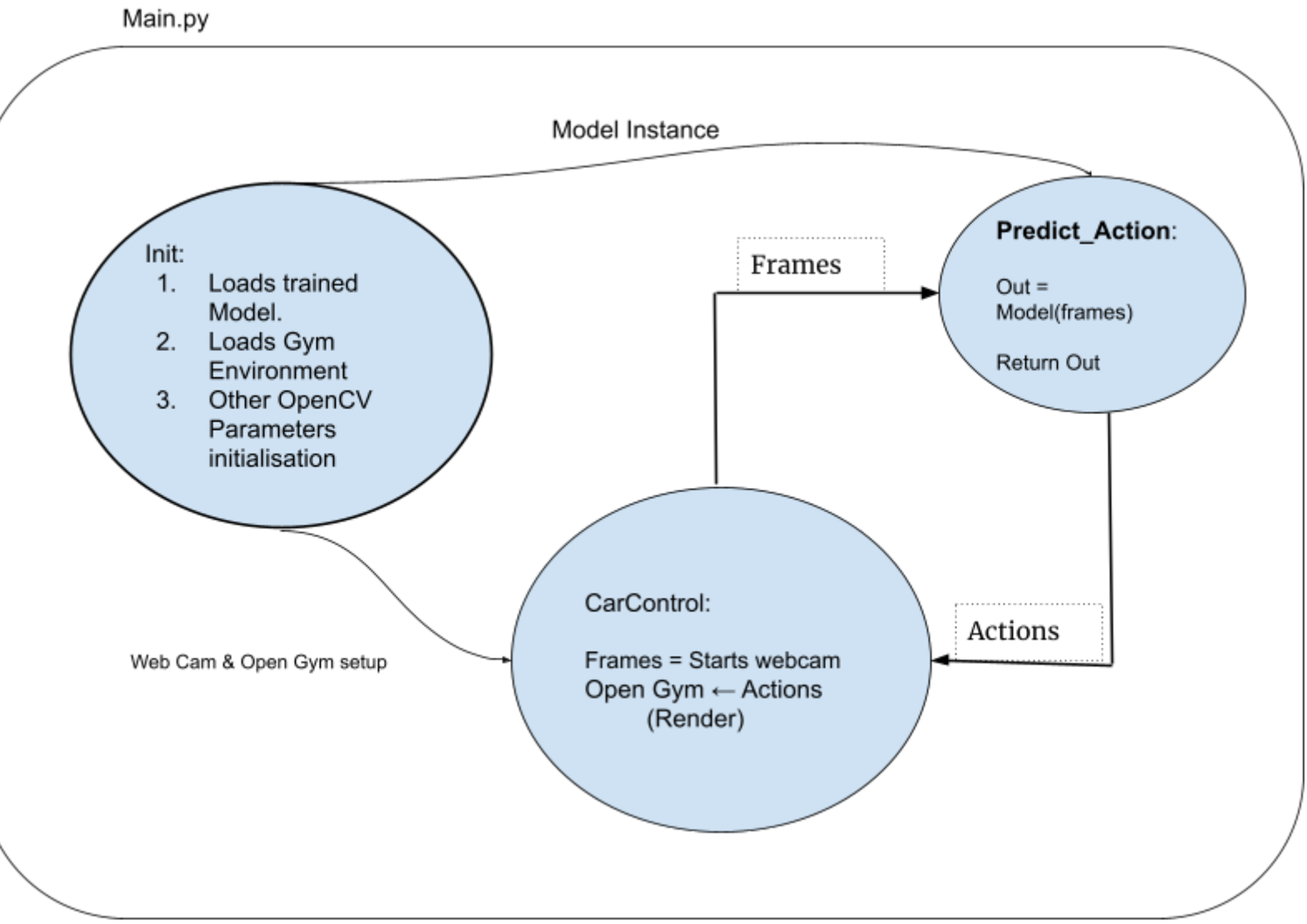


- Below you can see some shape of hand



Interpretation: Here we notice that the model has learned some required features according to the dataset provided. Initial layers were not very valuable, but as it goes on improving features, last layers give some valuable information.

❖ Code Work Flow:



- **Model_conversion** - Converts Model From Pytorch state dictionary to JIT which is production ready.
  - Also exports model to ONNX and keras (for tf-keras-viz)

- Dataset_capture.py - To capture Dataset
- Tranining.py - To Train model.
- Evaluation.py - To Evaluate model on given set

❖ Failed Approach:

I also tried visualising the model using tensorspace.js tool kit. Basically it gives more information about how neural nets look when predicting some inputs. For this we need to convert the Keras model (For this I converted Pytorch model to Keras). And then Keras model needs to be in their required format. I also converted into there required format. But it requires Node JS expertise to see visualisation. Unfortunately I was unable to do this!
Also to increase overall speed, I tried to do threading of frames. But due to time restrictions I was not able to achieve this!

❖ Suggestion And Further Improvements:

- ➢ Currently we have trained this model on Images, but in real time we have a sequence of images. Hence, we can solve this by using 3D CNN along with LSTM layers. We can do this by -
  - Extracting features from 3D CNN
  - Building multiple LSTM layers to process this frames followed by Softmax at the end of this layer.
  - By taking Average of all outputs of softmax we can predict the final action.
- ➢ Currently the model gets confused between 1 and 0 classes. So we can develop a 2 stage classifier. Although it would be much slower but can be optimised.
  If Current model (Parent model) gives probabilities biased for this two classes
  Then passing this frame to the next model which is trained on only a particular class.

- ➢ I forgot to do pruning for these models when converting them to .jit. Currently just converted to .jit. But Methods like pruning and quantisation helps to optimise FPS.

- ➢ Although these frames were slightly slow, (because of the model on cpu), using threading we can optimise some Frame rate.

- ➢ Also above -resnet-18 model has some more capacity to get better, due to GPU restrictions I was not able to train for longer. Else it can get better , increasing some epochs.

- ➢ GANs are also useful for generating data.

- ➢ While collecting data, having variety of background is very important to make it more real time, hence one strategy would be to record data in green background and separate object from background using algorithms like KNN
  And then render these objects with different backgrounds.

- Thank You!