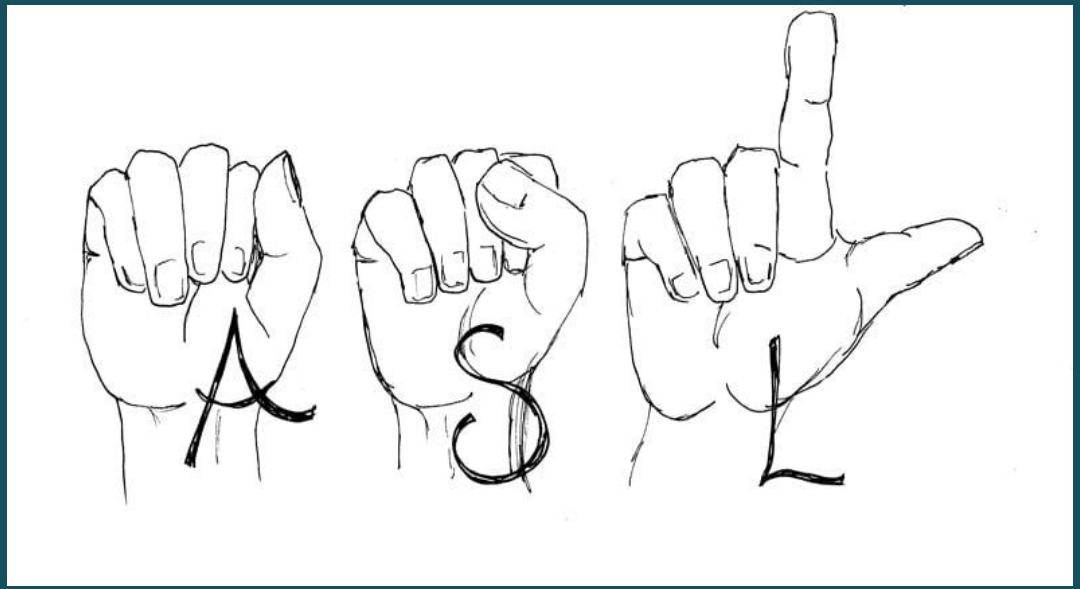
ASLNET



Neural Network for American Sign Language Translation

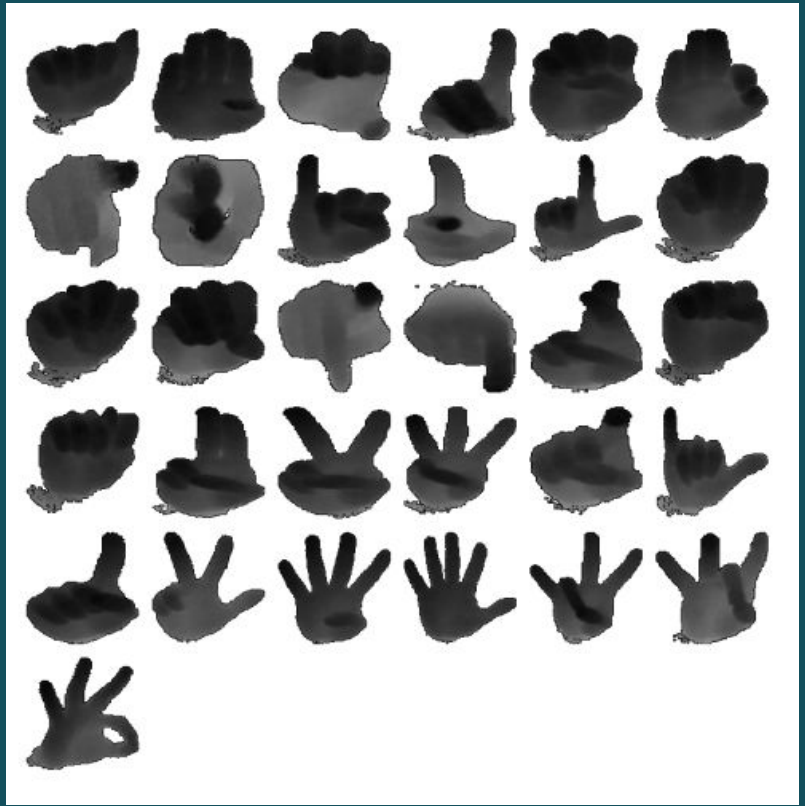
By Romeno Wenogk, Takumi Miyawaki, and Munachiso Nwadike

Dataset

- The ASL alphabet is the standardised method for American Sign Language
- The training data set contains 87,000 images which are 200x200 pixels.
- 29 classes, of which 26 are for the letters A-Z and 3 classes for *SPACE*, *DELETE* and *NOTHING*
- The test data set contains a mere 29 images, to encourage the use of real world test images.

Previous Works

- Work from UCSD on Fingerspelling Recognition
- Dataset of 31,000 depth maps
- Achieves >90% accuracy
- Differs from our model because it requires sophisticated tech such as Kinect or 3D Camera
- We aim to perform classification with merely a smartphone camera



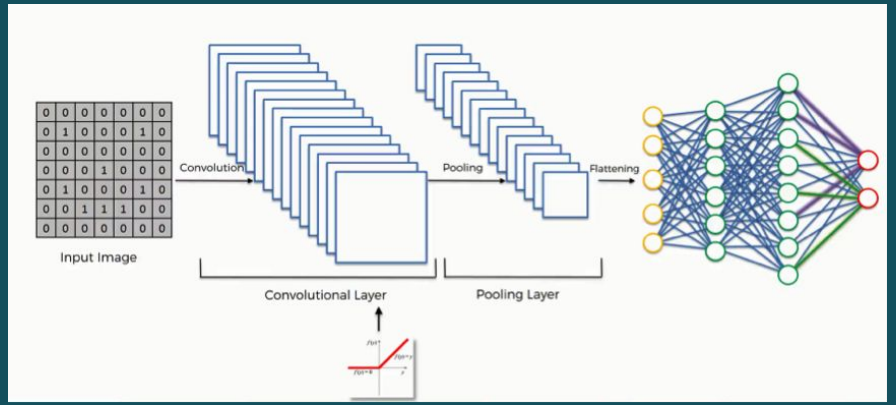
Convolutional Neural Network

Basic Architecture:

- Three convolutional layers with relu functions
- Three max-pooling layers
- Three fully connected layers

Result:

- Accuracy: 96.64%
- Execution time : 2.29ms




```
model = Sequential()
model.add(Conv2D(32, (5, 5), activation='relu',
                 input_shape=(120, 320, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(29, activation='softmax'))
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	del	nothing	space
A	937	12	0	7	8	5	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
B	4	882	0	2	8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	2	0	874	5	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	4	1	0	877	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	9	25	2	16	793	0	0	0	16	0	1	0	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0
F	2	0	0	2	2	953	0	0	1	0	0	0	0	0	3	0	2	0	0	0	0	0	1	0	0	0	0	0	0
G	0	0	0	0	0	0	890	8	1	7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	9	889	1	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	3	0	0	0	0	873	0	9	0	1	3	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	4	930	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	1	0	3	0	914	0	0	0	0	0	0	1	0	0	0	1	6	0	0	0	0	0	0
L	1	0	0	0	0	0	0	0	3	0	8	897	1	0	9	0	0	0	0	1	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	5	0	5	0	850	17	2	0	0	0	1	0	0	0	0	0	0	0	0	1	0
N	0	0	0	0	0	0	0	0	0	0	0	0	7	834	4	0	0	0	6	0	0	0	0	0	0	0	0	0	0
O	1	0	0	4	1	2	0	0	10	0	0	4	5	3	862	0	0	0	1	5	0	0	0	2	1	0	0	0	0
P	0	0	0	0	0	0	0	5	0	1	0	0	0	0	0	855	10	0	0	0	0	0	0	0	0	0	0	1	0	1
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	939	2	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	3	2	0	0	0	0	0	0	863	1	0	11	7	1	5	0	0	0	0	2
S	0	0	0	1	0	0	0	0	7	0	0	0	0	0	0	0	0	4	855	4	0	0	1	7	0	1	0	0	1
T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	1	855	0	3	1	0	2	9	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	11	3	1	701	165	12	24	1	2	3	0	1
V	0	0	0	0	0	0	0	2	1	0	1	0	0	0	0	0	0	1	0	0	2	805	39	6	11	6	0	0	4
W	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	21	811	10	2	1	0	0	0
X	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3	0	2	3	10	864	9	1	3	0	3
Y	0	0	0	0	0	0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	0	1	1	8	851	23	1	1	7
Z	0	0	0	0	0	0	0	0	0	0	0	0	2	0	3	0	0	0	1	0	0	0	1	4	0	836	3	0	3
del	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	914	0	3
nothing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	950	0
space	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	867

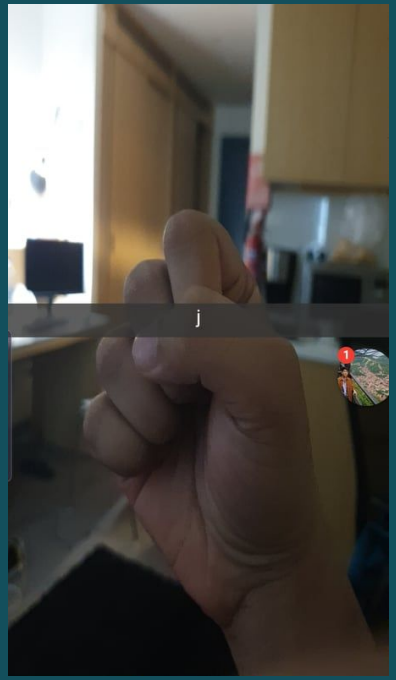
CNN with the application

Shortcomings:

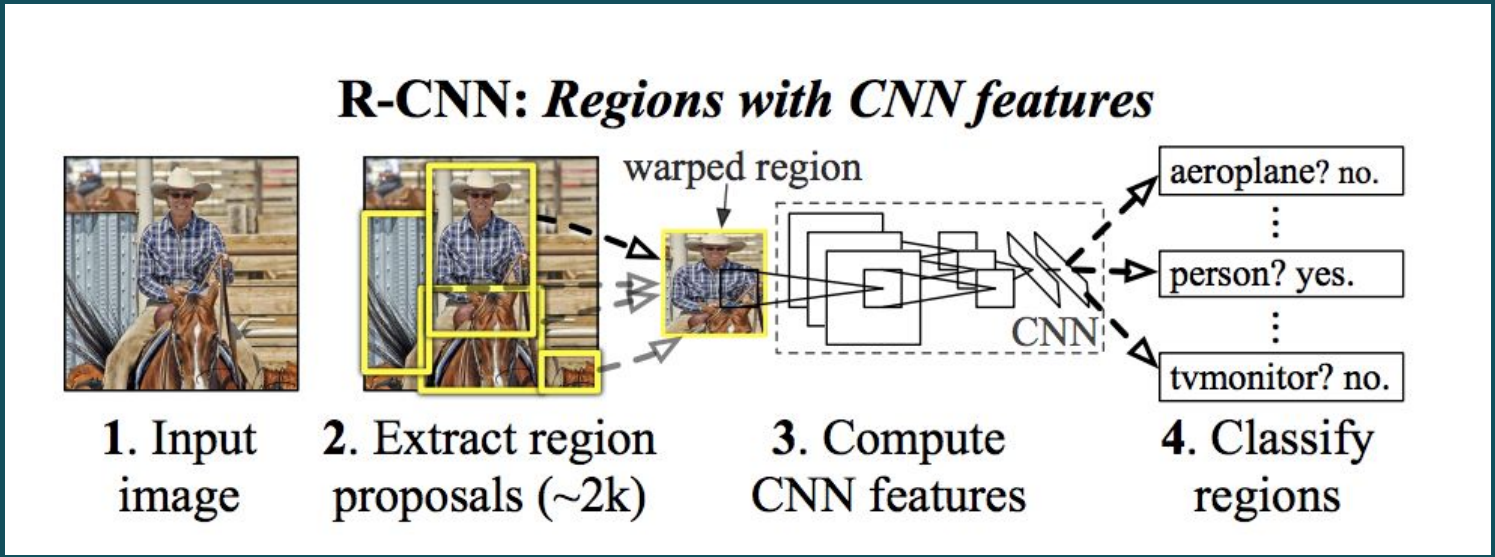
- Weak to size
- Dependent on dimension

Possible Alternatives:

- R-CNN
- Faster RCNN
- YOLO v3



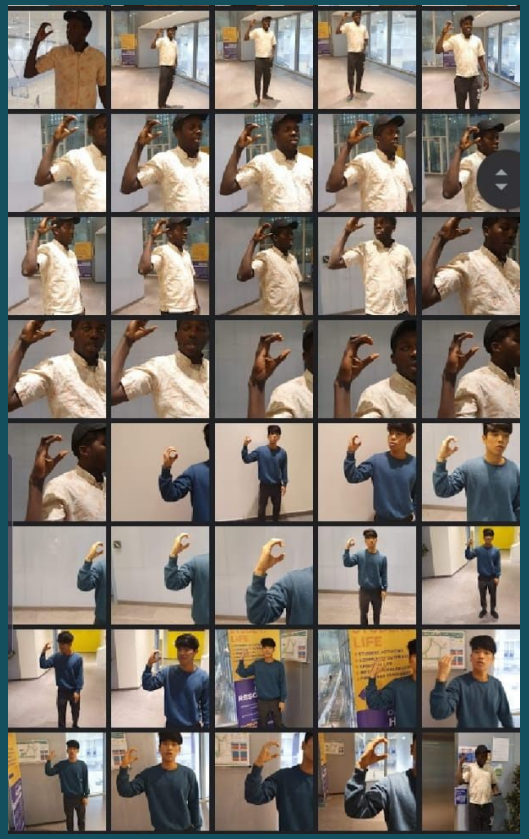
R-CNN (Region based Convolutional Neural Network)

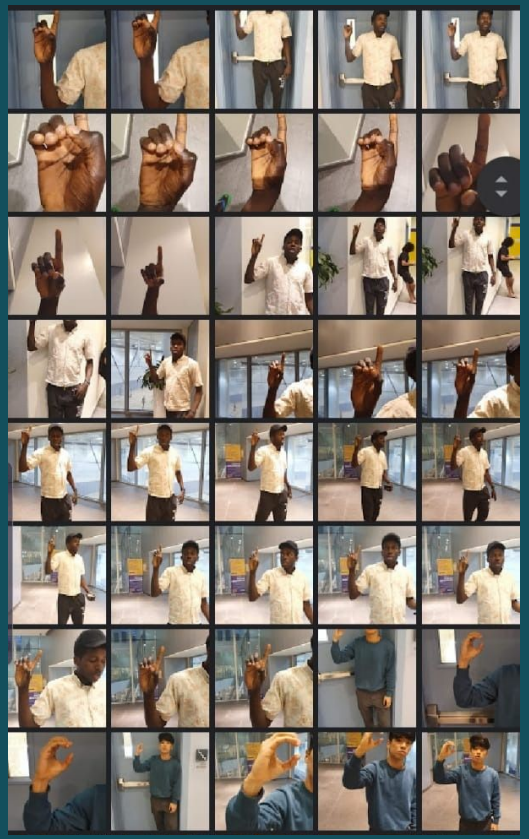


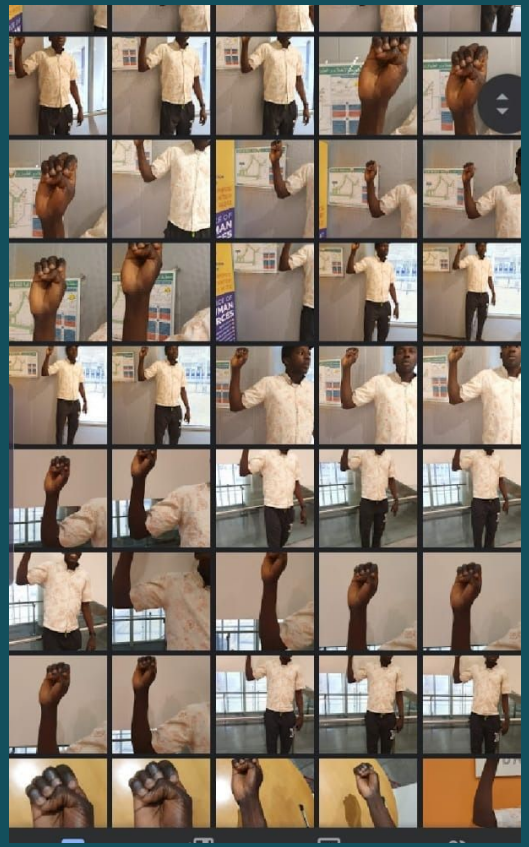
Deploys a region proposal network to detect and localise the position of the hand, then use a trained CNN for classification

Previous Works

- Faster R-CNN with multiscale detection for robust hand detection and classification in sign language, proposed by University of Science and Technology, Hefei, China.
- Simply detects hands - proprietary dataset



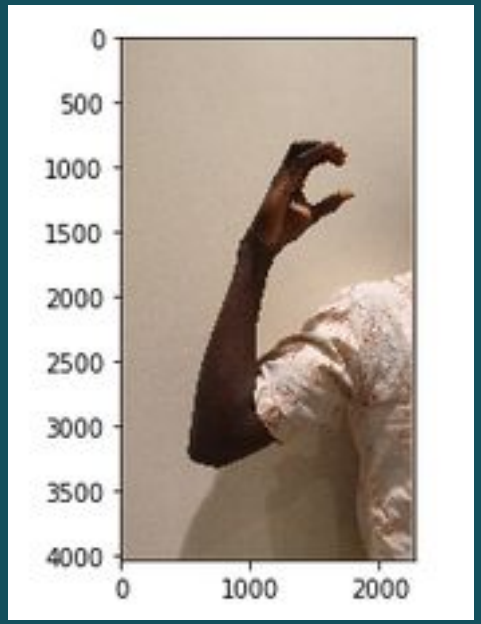


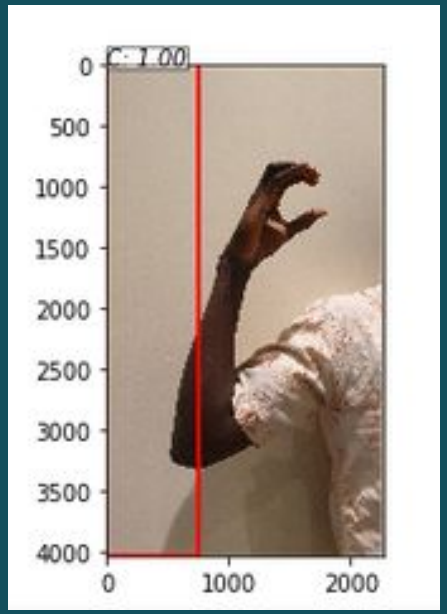


YOLO-> Challenge: Spot the Difference?

Initially, YOLO was learning the right classification, but the wrong location!

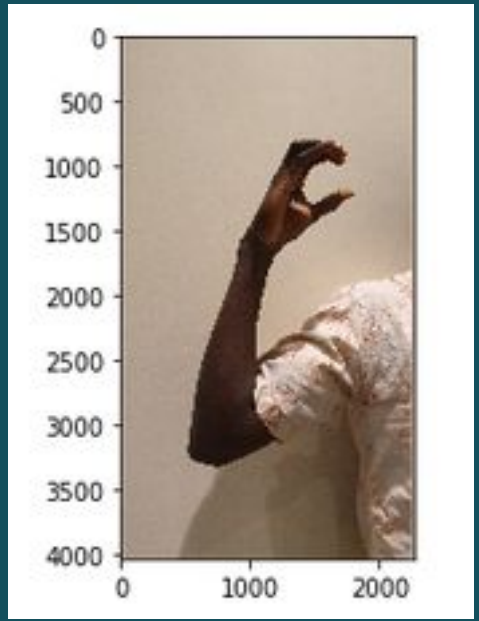
Challenge: Spot the Difference?

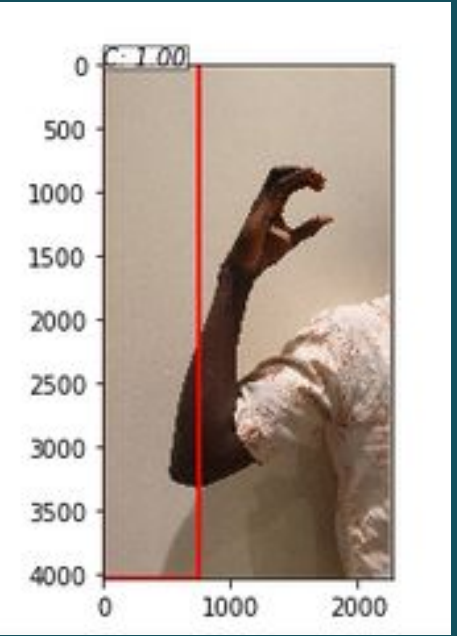




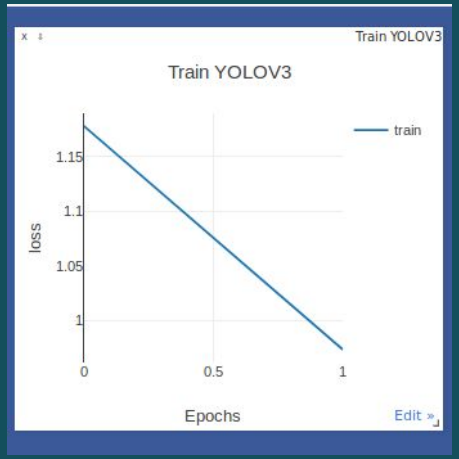
How is that Possible?

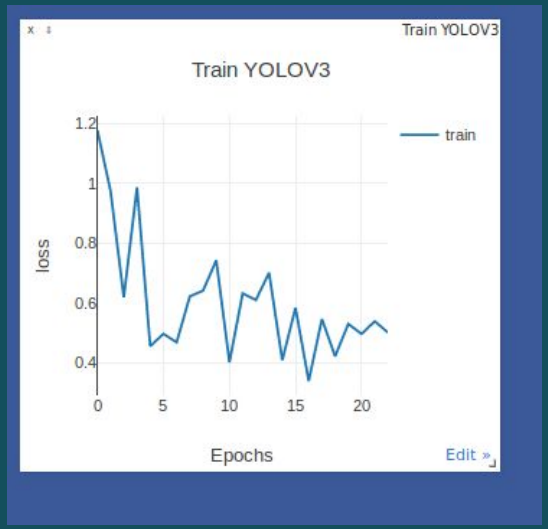
- Classification requires first having localisation, No?
- Not really!
- YOLO is ***convolutional***, so it considers every region in image anyway!
- YOLO Losses -> Seperate
- [location_loss (x, y) + class_loss]
- Can see “C” in the image, without knowing where!

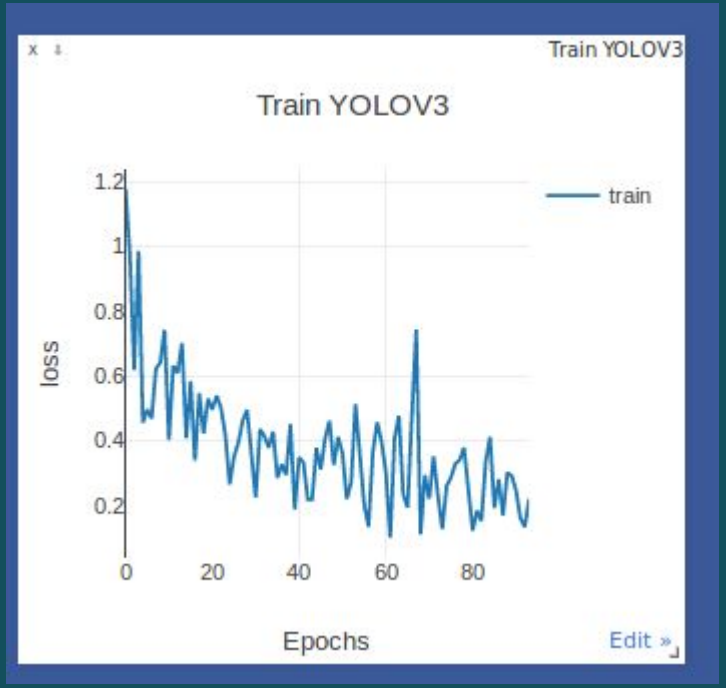




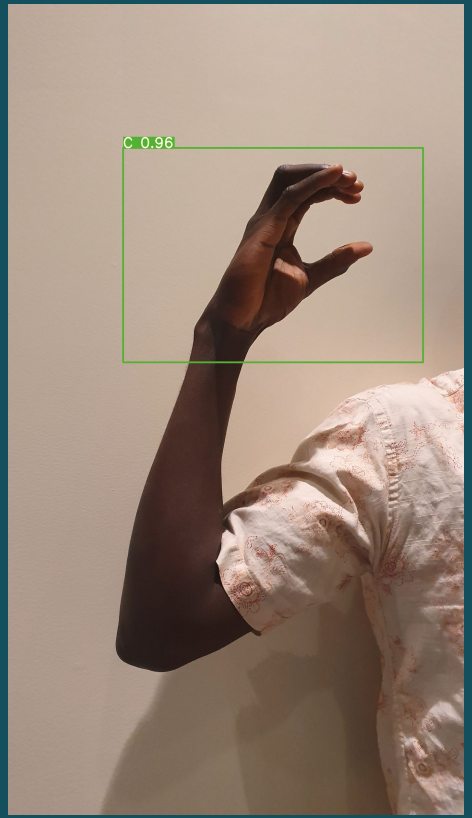
… And monitor the training for when the loss was no longer reducing...



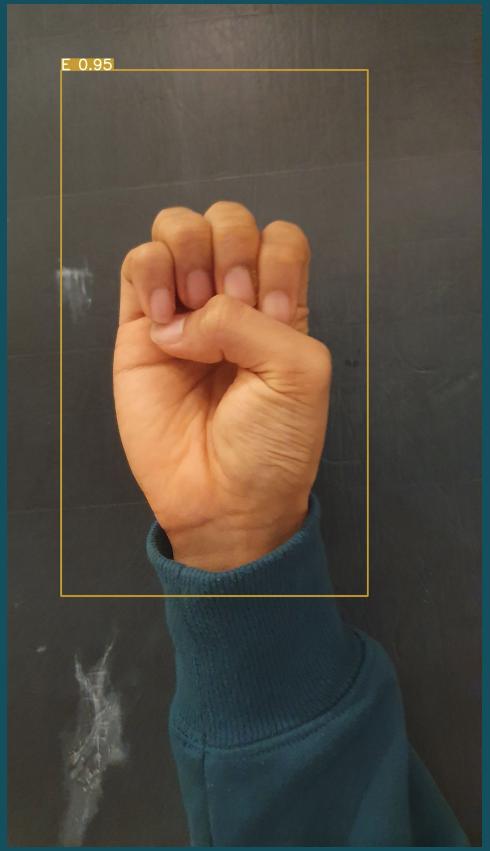


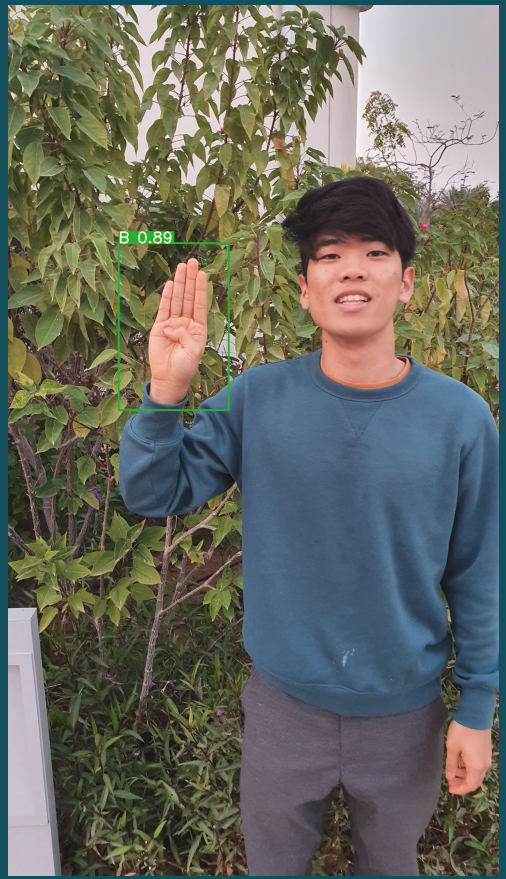


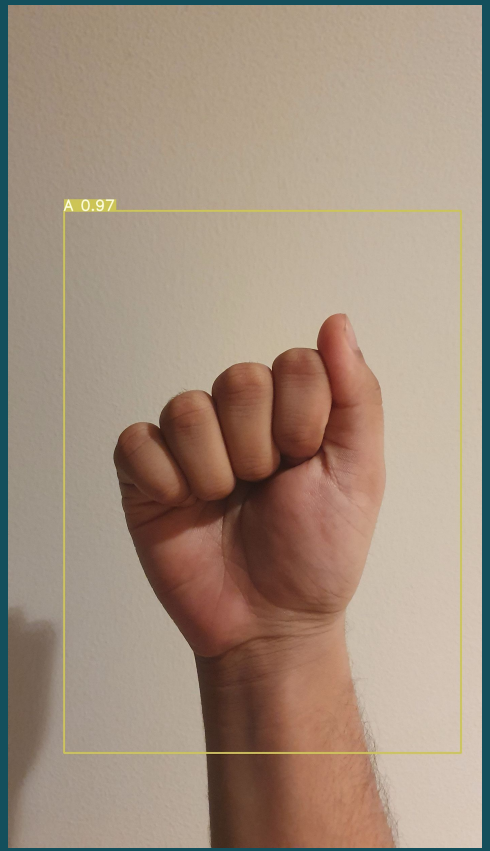
And VOILA!



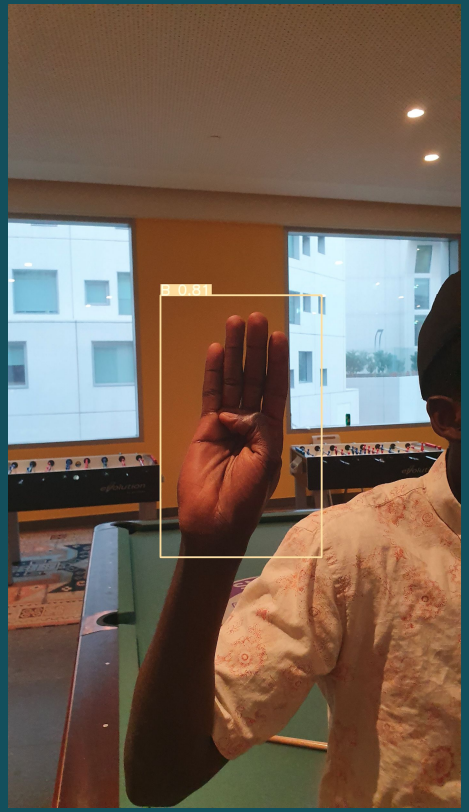
Notice that the Model Distinguishes A vs. E







Speed | Approx .14 ~.16s/image | Saving adds about .4 ~ .5s/image



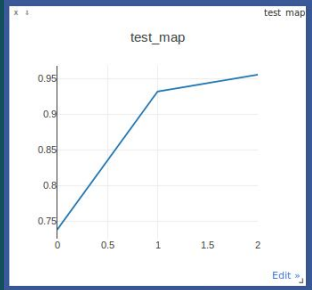
Performance

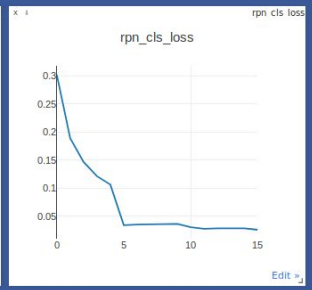
- YOLO 0.949 mAP on training data

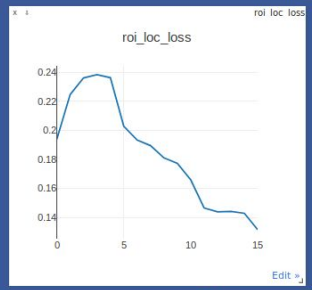
Class	Images	Targets	P	R	mAP	F1:
all	500	500	0.013	0.988	0.949	0.0257
A	500	101	0.0117	0.97	0.962	0.0232
B	500	99	0.0218	1	0.977	0.0427
C	500	100	0.00942	0.97	0.934	0.0187
D	500	100	0.00836	1	0.895	0.0166
E	500	100	0.0138	1	0.979	0.0272

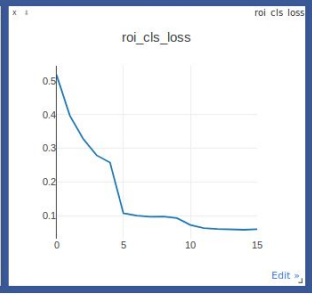
- Faster RCNN - mAP of 1 on training data (overfitting!) - so we stick with YOLO

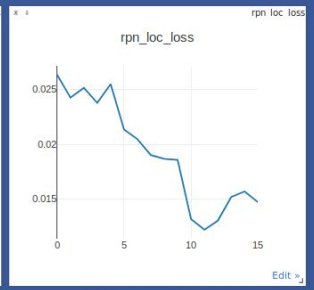
Faster RCNN


test_map
0.95
0.9
0.85
0.8
0.75
0
0.5
1
1.5
2
Edit »

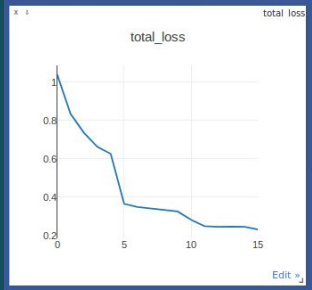

rpn_cls_loss
0.3
0.25
0.2
0.15
0.1
0.05
0
5
10
15
Edit »

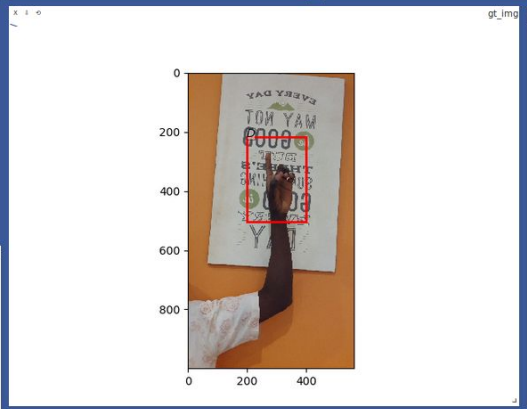

roi_loc_loss
0.24
0.22
0.2
0.18
0.16
0.14
0
5
10
15
Edit »

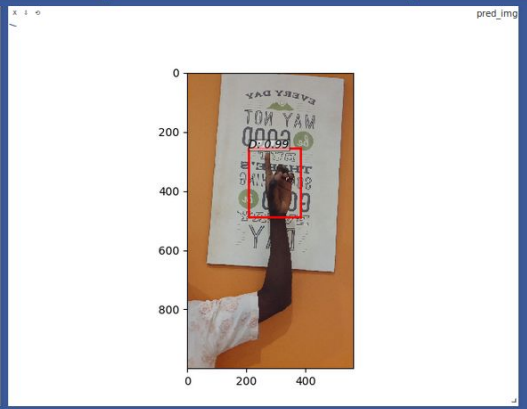

roi_cls_loss
0.5
0.4
0.3
0.2
0.1
0
5
10
15
Edit »


rpn_loc_loss
0.025
0.02
0.015
0
5
10
15
Edit »

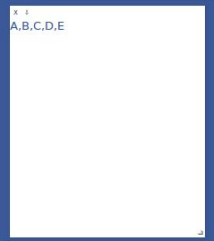

[1204_140735] lr:0.001,
map:0.7376773078523317,loss:
{'rpn_loc_loss':
0.0255062616884824 75,
'rpn_cls_loss':
0.10640982094872763,
'roi_loc_loss':
0.2362140293269112,
'roi_cls_loss':
0.2579055302739141,
'total_loss':
0.6260356424301858}
[1204_140958] lr:0.001,
map:0.9316878317229487,loss:


total_loss
1
0.8
0.6
0.4
0.2
0
5
10
15
Edit »

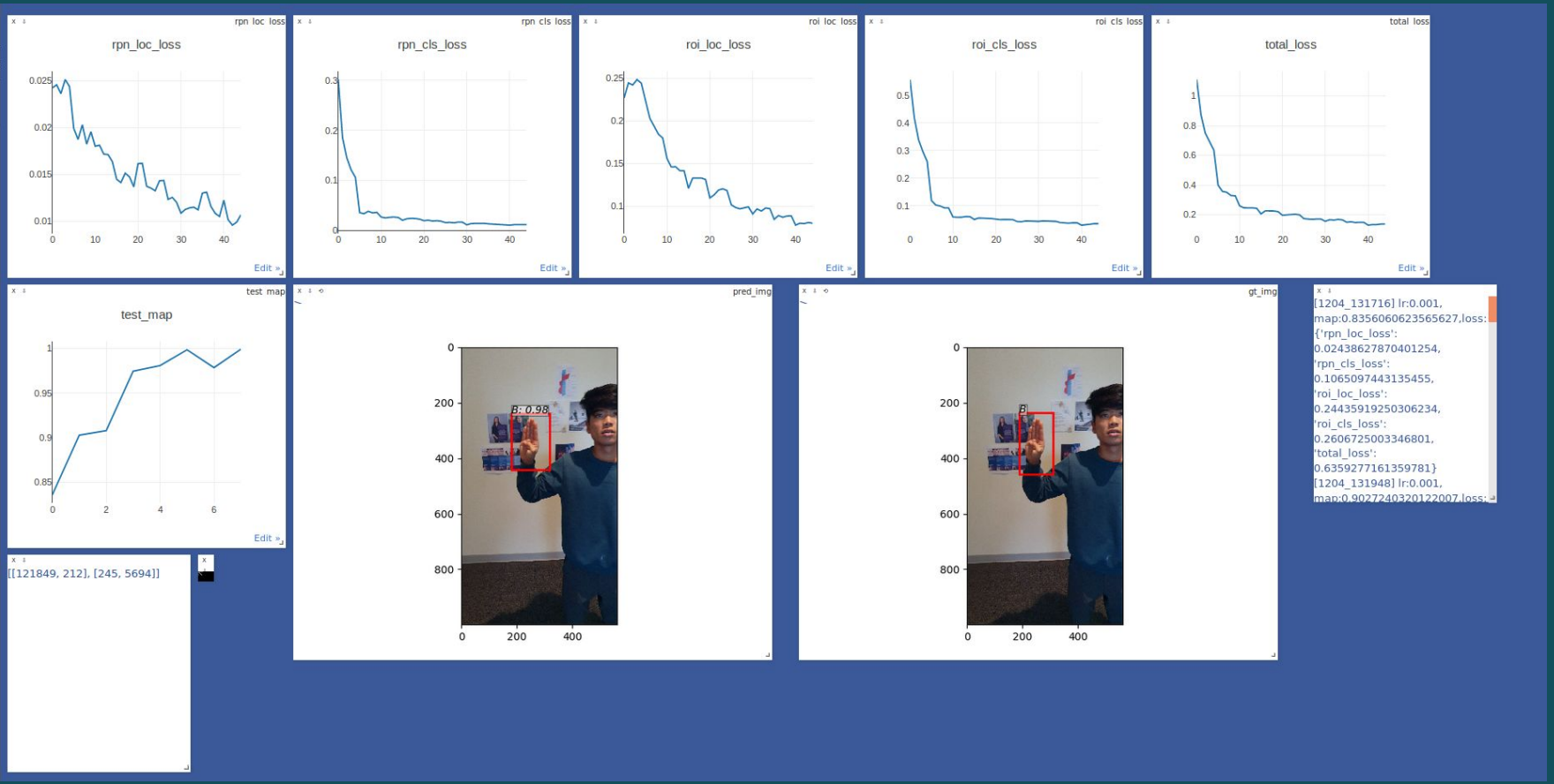

gt_img
0
200
400
600
800
0
200
400


pred_img
0
200
400
600
800
0
200
400
0.99

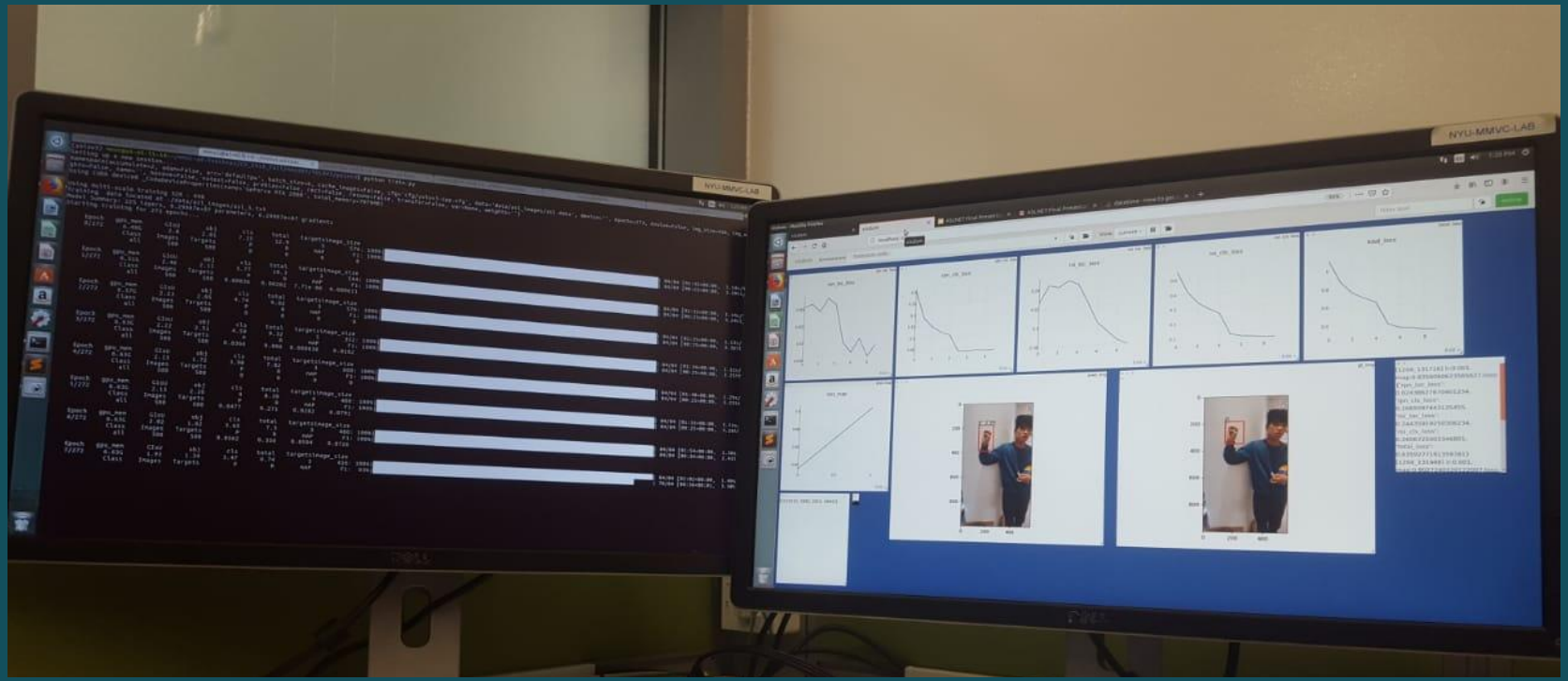

[[121544, 502], [650, 5304]]


A,B,C,D,E

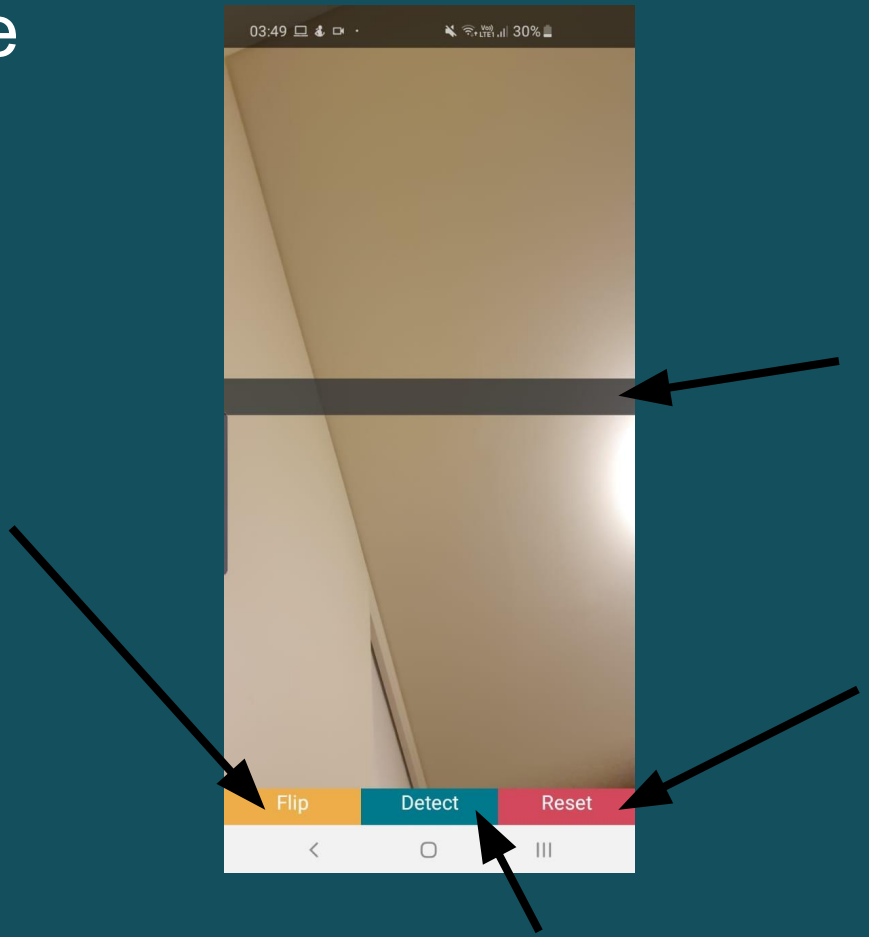
+25 epochs later …



Final Product Demo | YOLO vs. Faster RCNN

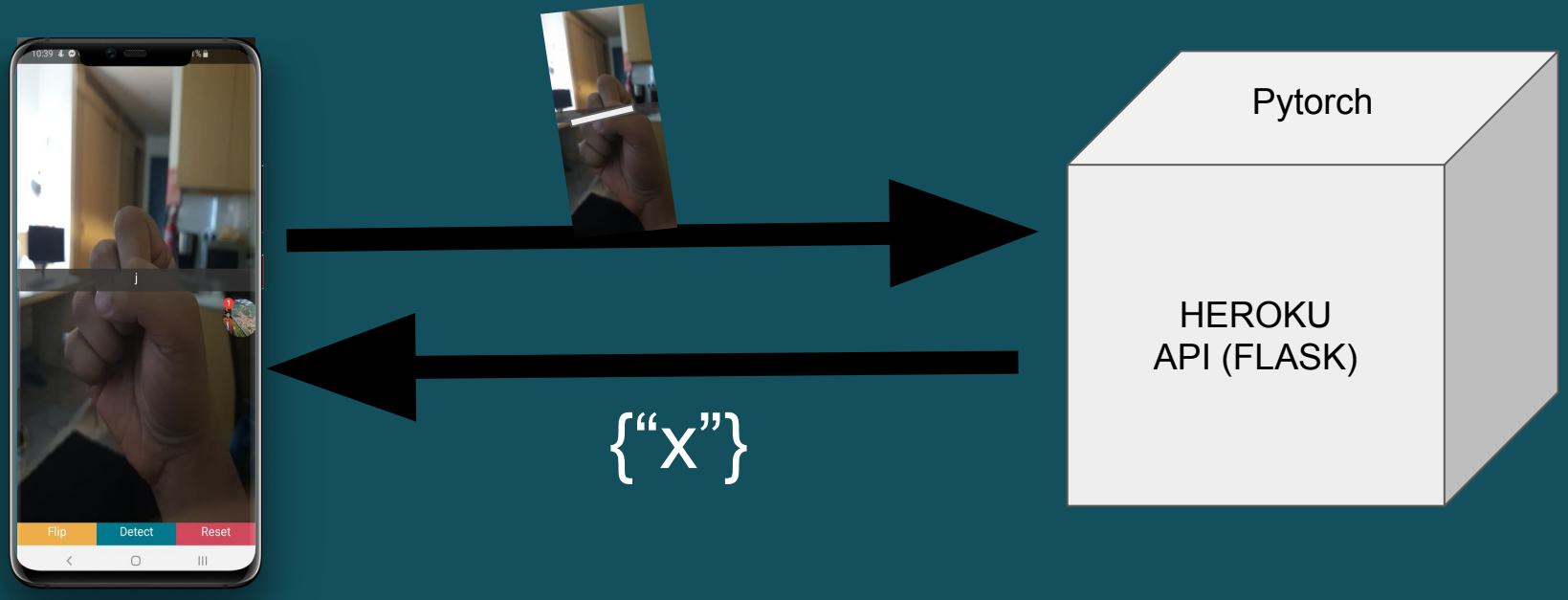


App interface



App infrastructure version 1

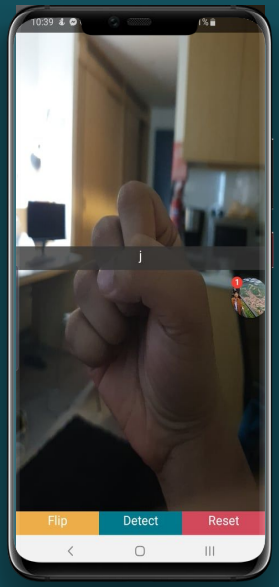
- API, Built on Flask
  - Hosted on heroku
  - Pytorch Yolo pretrained model ported into Onnx
  - Onnx model ported into Tensorflow.js
  - Allows API calls with images and ASL letter detected
- React Native for mobile application
  - Uses mobile phone camera
  - Runs pre-image processing on the phone itself (rescaling, grayscaling)
  - Uploads image to API and gets response and displays it


Pytorch
HEROKU
API (FLASK)
{“x”}
Flip
Detect
Reset

Shortcomings

- Too slow without a GPU
- Saved model file was 450mb+

App Infrastructure 2



{“x”}

Pytorch

~~HEROKU~~
API (FLASK)

Campus server ()

Improvements

YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers

Rachel Huang*
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, United States
rachuang22@gmail.com

Jonathan Pedoeem*
Electrical Engineering
The Cooper Union
New York, United States
pedoeem@cooper.edu

Cuixian Chen
Mathematics and Statistics
UNC Wilmington
North Carolina, United States
chenc@uncw.edu

***Abstract*—This paper focuses on YOLO-LITE, a real-time object detection model developed to run on portable devices such as a laptop or cellphone lacking a Graphics Processing Unit (GPU). The model was first trained on the PASCAL VOC dataset then on the COCO dataset, achieving a mAP of 33.81% and 12.26% respectively. YOLO-LITE runs at about 21 FPS on a non-GPU computer and 10 FPS after implemented onto a website with only 7 layers and 482 million FLOPS. This speed is 3.8×**

https://arxiv.org/pdf/1811.05588.pdf