



Detecting Manufacturing Nonconformities through Live Video Deep Learning Classification

General Assembly Data Science
Immersive Capstone

Rashidi
SG-DSI-27

Contents

- Background
- Project Overview
- Model Development
- Model Analysis
- Implementation of Regions of Interest (ROIs)
- Production Model Implementation
- Conclusions



Background

Background

Problem Statement:

To implement a Live Video Deep Learning Classification Model to effectively **detect product defects** and **procedural non-conformities** in the Manufacturing Production Line.



Product Defects



Procedural Non-conformities

Target Audience:

Quality Control stakeholders in the Manufacturing sector

Background

Preamble

As a Manufacturing Quality Data Scientist, I, together with a task force, was tasked to **investigate a Customer return merchandise authorization (RMA) event** relating to a defect in one of our newly released products.

Upon investigation, we found that there was a **random periodical procedure non-conformance which was undetected** and led to the Customer receiving the poor-quality product.

return merchandise authorization (RMA)

≈

return of broken products under warranty

random periodical procedure non-conformance

=

random human errors

Background

Task

In the interim while the Manufacturing Process Engineer devise a more permanent solution, I was tasked to **produce a Computer Vision Deep Learning solution that is effective in screening out non-conforming products and deviations** specific to the investigated root cause to prevent further Quality escapes.

Project Overview

Project Overview

Overall Manufacturing Production Flow

General Manufacturing Production Flow

Material
Preparation



Product
Assembly



Quality
Testing and
Inspection



Packaging/
Shipping

Deep Learning Classification

Poor Product
Assembly



Undetected in
Quality Testing
and Inspection

Deviated Manufacturing Production Flow

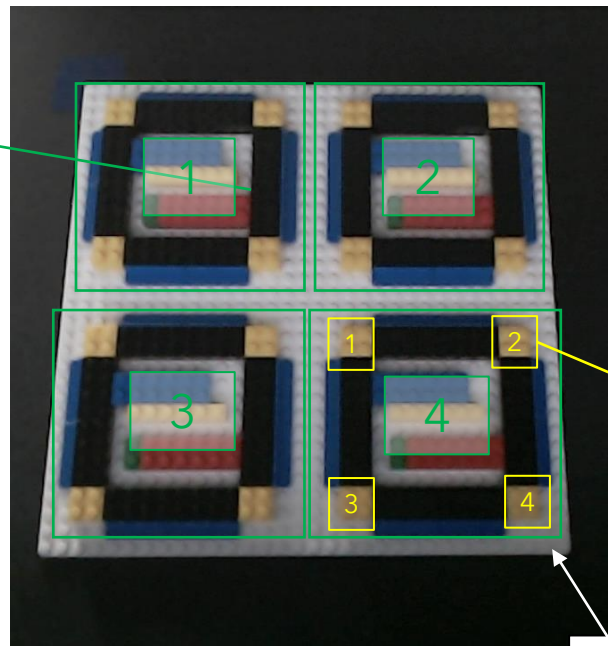
Project Overview

Product Assembly Procedure

Note: Lego® blocks were used to mimic the product and its assembly

- Operators are tasked to install Yellow corner blocks onto all corners of each Printed Circuit Board (PCB)
- Deviation occurred during assembly of Yellow corner blocks
 - Red block wrongly installed

4 *
Printed
Circuit
Boards

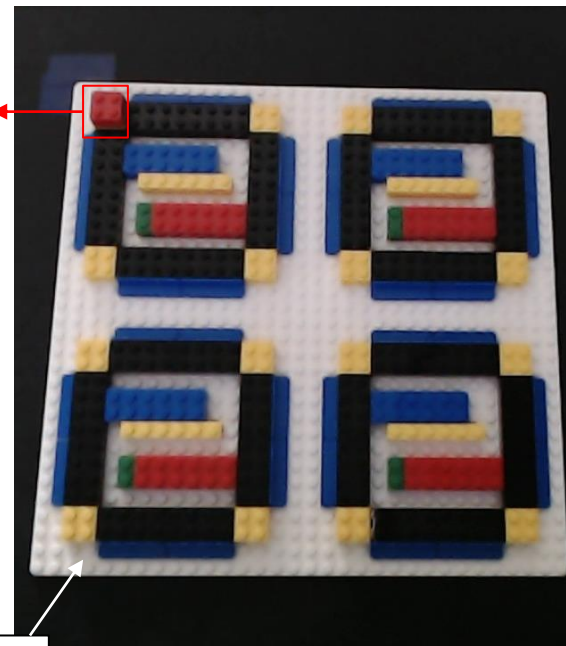


Passing
Product

Red
block

4 * 4 *
Yellow
Blocks

White
Baseplate



Failing
Product

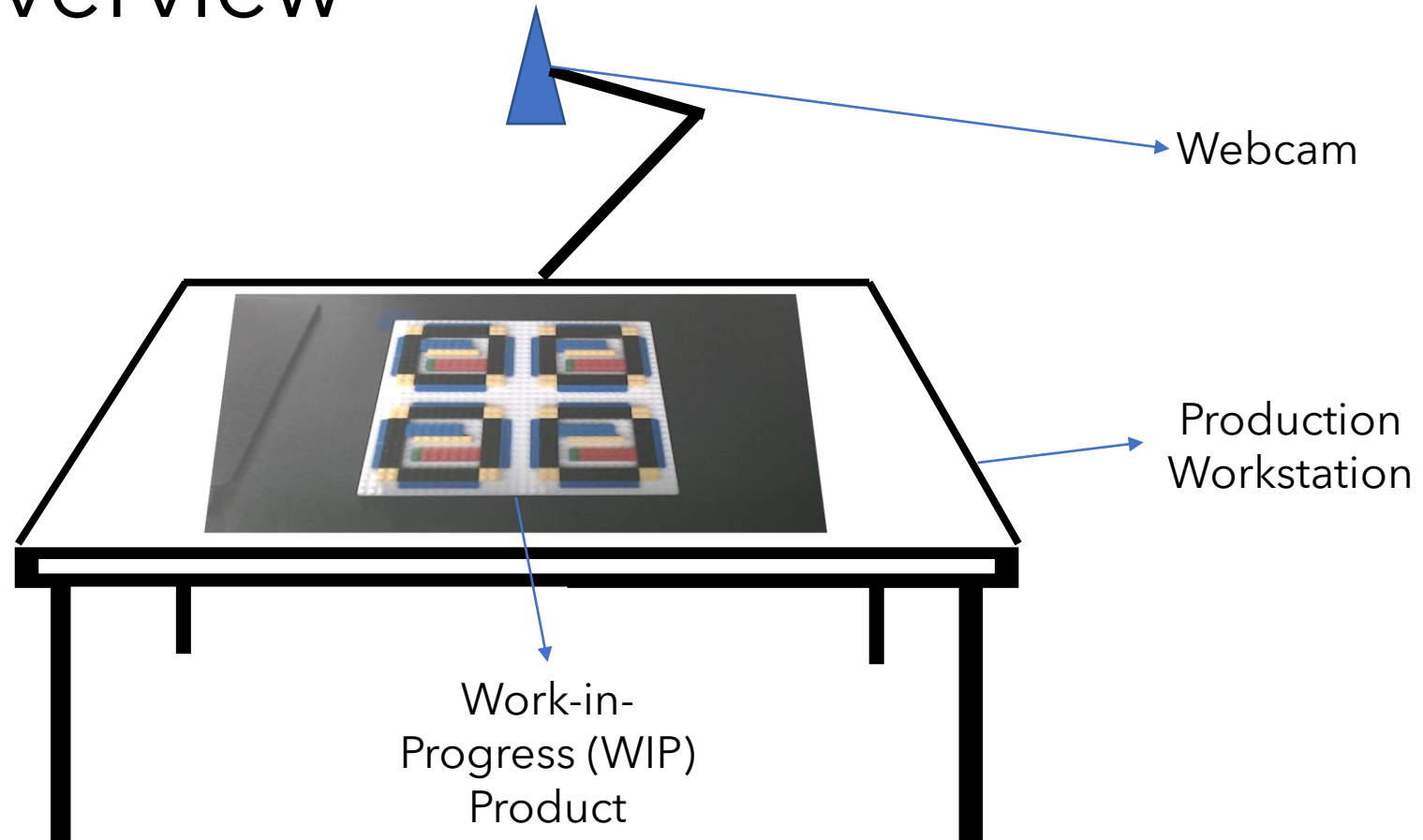
Real-life Example



Source: @lulupcb

Project Overview

Setup Required



Project Overview

Comparing Proposals for Model Implementation

Static Image Classification

Requires **user-input to trigger manual image capture** for each product assembly occurrence.

Pros

- Less computational power usage overall.
- Resistant to noise.

Cons

- **Lowers Production Output**
- Additional procedure added to current assembly workflow.
 - Requires Retraining of the Production Operators.

Live Video Classification

Requires **user-input to start the Live Video Classification program once** at the start of the workday/start of a batch.

Pros

- **Minimal disruption to Production Output**
- Can be triggered automatically by existing Manufacturing Execution Systems (MES) APIs.

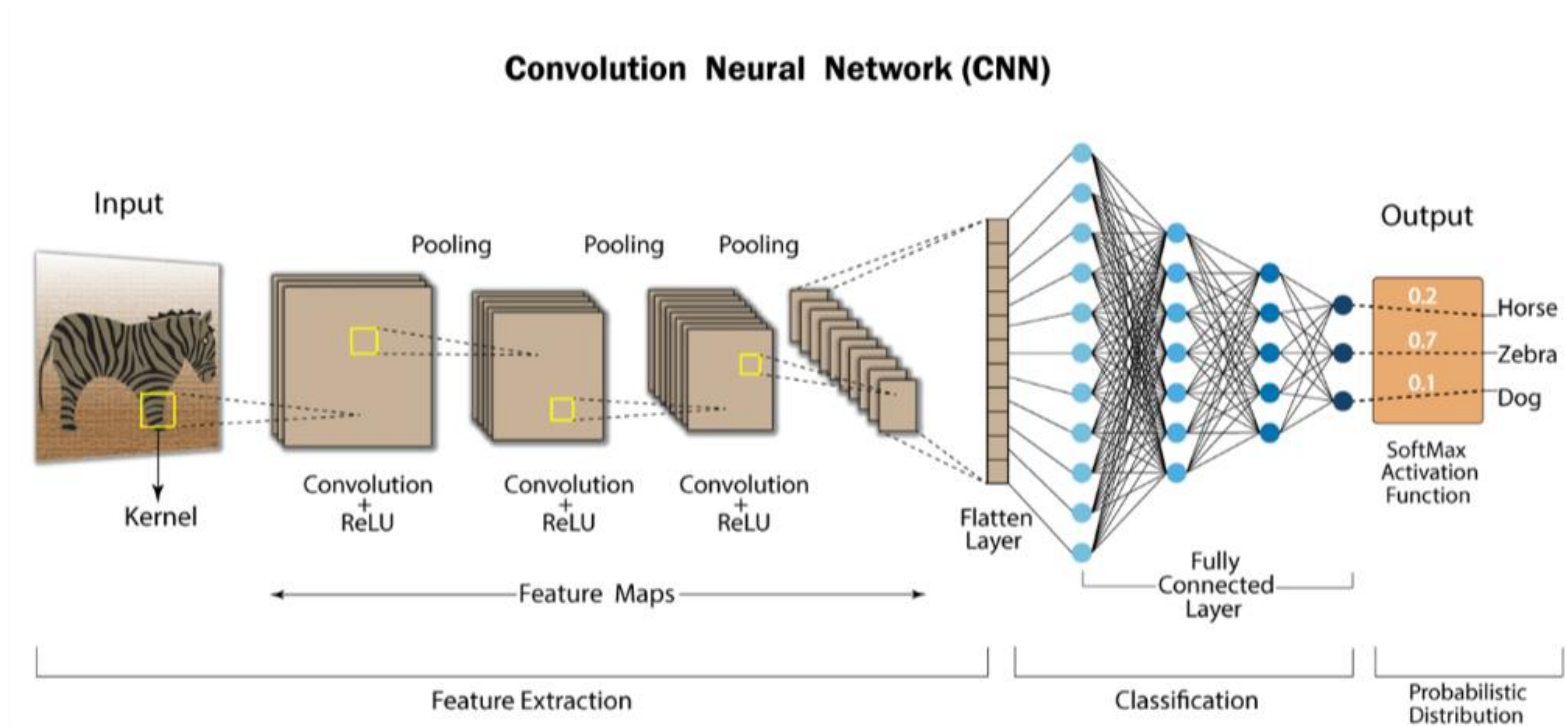
Cons

- Requires more computational power since the video feed is evaluated constantly.
- Not as robust to noise.

Model Development

Model Overview

Convolutional Neural Networks



Source: <https://developersbreach.com/convolution-neural-network-deep-learning>

Model Selection

Custom CNN vs. Transfer Learning

Custom CNN

Model: "sequential"			
Layer (type)	Output Shape	Param #	
conv2d (Conv2D)	(None, 240, 320, 16)	448	
max_pooling2d (MaxPooling2D)	(None, 120, 160, 16)	0	
conv2d_1 (Conv2D)	(None, 118, 158, 64)	9280	
max_pooling2d_1 (MaxPooling2D)	(None, 59, 79, 64)	0	
flatten (Flatten)	(None, 298304)	0	
dense (Dense)	(None, 64)	19091520	
dropout (Dropout)	(None, 64)	0	
dense_1 (Dense)	(None, 32)	2080	
dropout_1 (Dropout)	(None, 32)	0	
dense_2 (Dense)	(None, 1)	33	
Total params: 19,103,361			
Trainable params: 19,103,361			
Non-trainable params: 0			

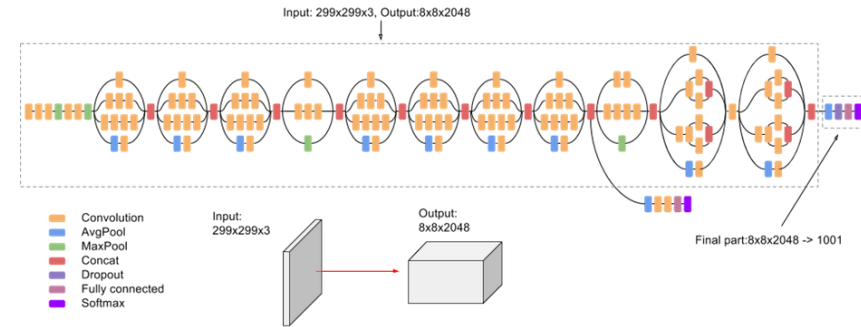
Feature
Extraction

Classification
(Fully Connected
Feed Forward
Neural Network
w/ Dropout)

```
1 cnn_model_2.evaluate(X_val_sc, y_val)

119/119 [=====] - 15s 122ms/step - loss: 0.0400 - accuracy: 0.9845
[0.04004882648587227, 0.9844614267349243]
```

Google's InceptionV3



InceptionV3 Trained Model (Transfer Learning)

Model: "sequential"		
Layer (type)	Output Shape	Param #
keras_layer (KerasLayer)	(None, 2048)	21802784
dense (Dense)	(None, 1)	2049
Total params: 21,804,833		
Trainable params: 2,049		
Non-trainable params: 21,802,784		

48 layers

```
1 tf_model.evaluate(X_val_sc, y_val)

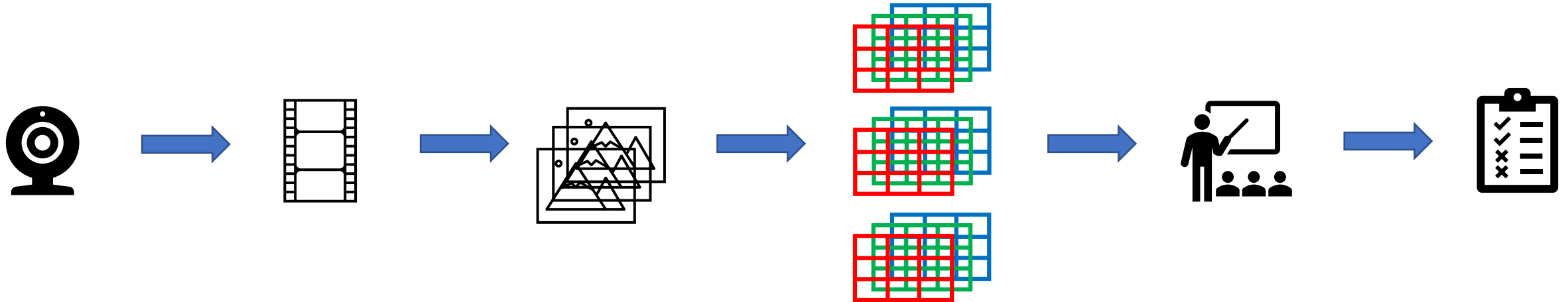
119/119 [=====] - 124s 786ms/step - loss: 0.0707 - accuracy: 0.9747
[0.07070834934711456, 0.9747169017791748]
```

Reasons:

- Marginally better accuracy
 - 0.9845 vs. 0.9747
- **Significantly faster prediction times**
 - **For ~3.7k images, 15s vs. 124s**

Model Development

Model Development Workflow

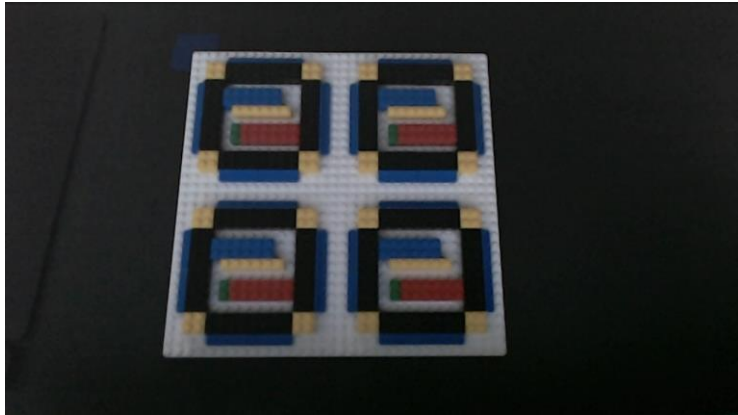


Model Training

Base Dataset Analysis

1: Pass

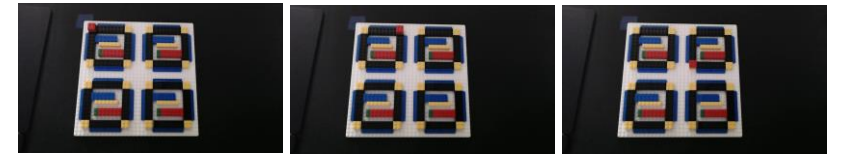
- 7,000 image (1280 x 720 pixels)



0: Fail

- ~500 images each from different kinds of failure (Total: 7000) (1280 x 720 pixels)
 - i.e. red blocks differing in location and quantity

1x Red



Some Reds



All Red



Model Training

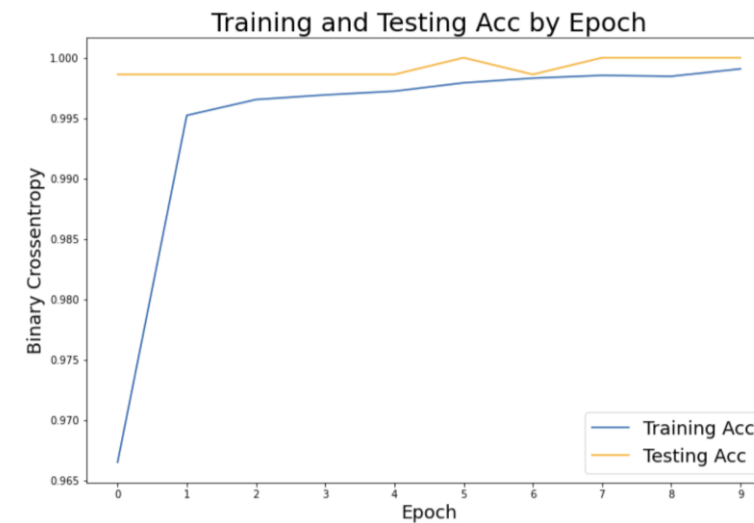
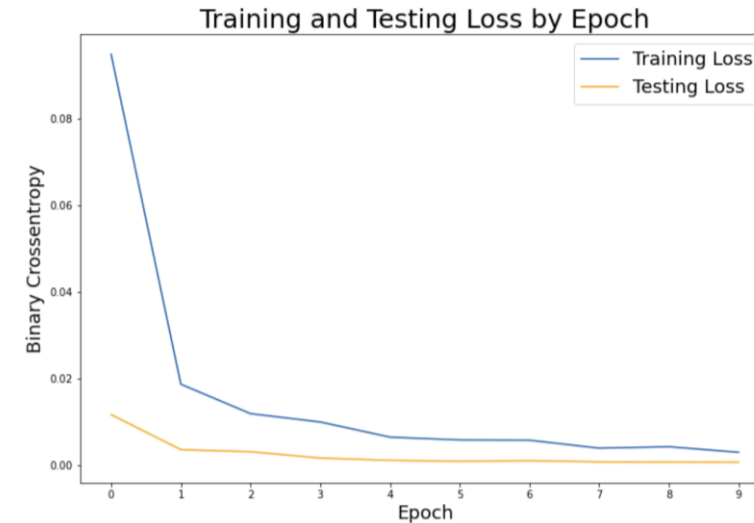
Custom CNN w/ Base Dataset

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 240, 320, 16)	448
max_pooling2d (MaxPooling2D)	(None, 120, 160, 16)	0
conv2d_1 (Conv2D)	(None, 118, 158, 64)	9280
max_pooling2d_1 (MaxPooling2D)	(None, 59, 79, 64)	0
flatten (Flatten)	(None, 298304)	0
dense (Dense)	(None, 64)	19091520
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

```
=====
```

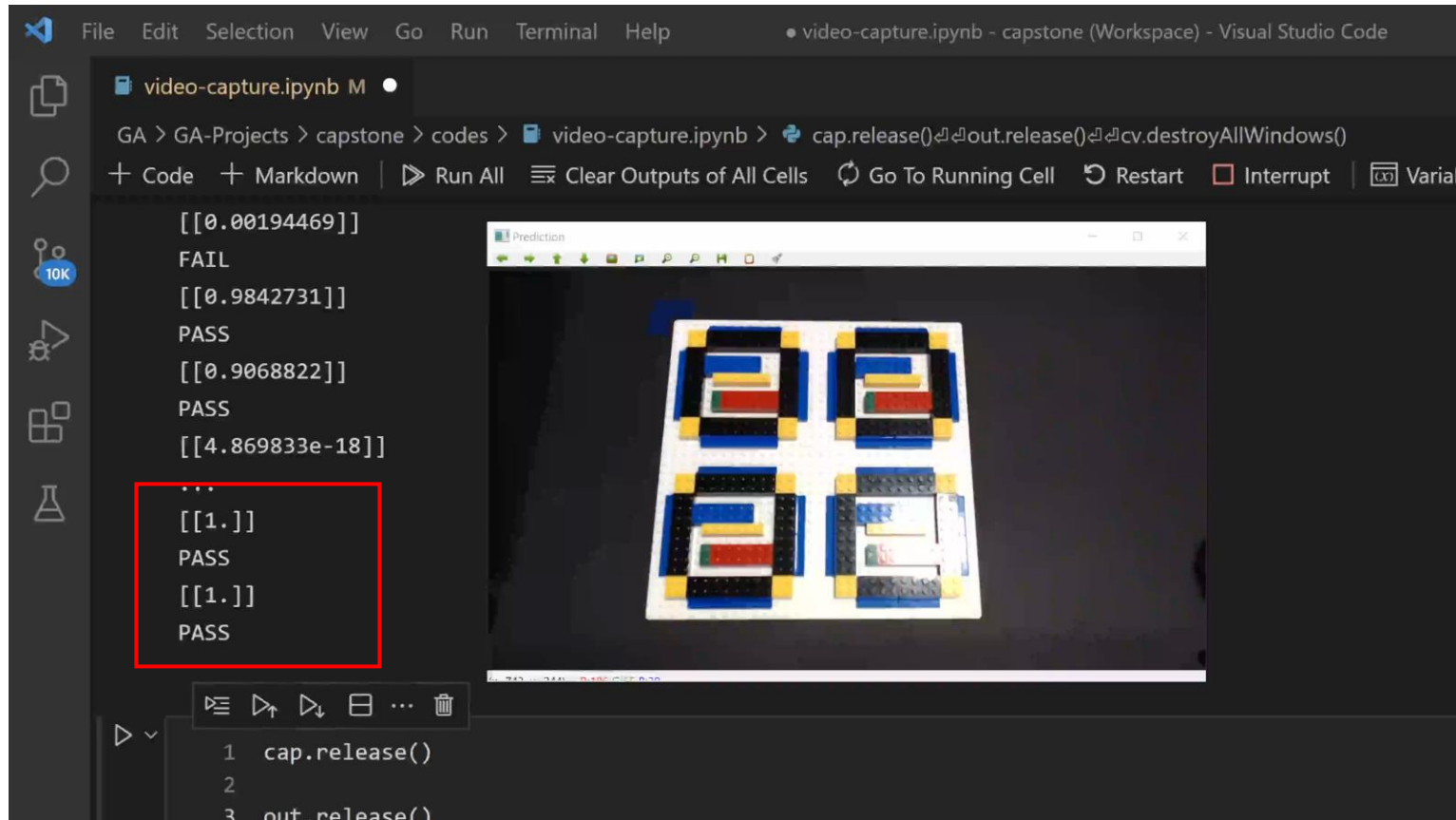
Total params: 19,103,361
Trainable params: 19,103,361
Non-trainable params: 0



After 10 epochs, we achieved 99.8% test accuracy!

Model Testing

Custom CNN w/ Base Dataset



The screenshot shows a Jupyter Notebook interface with the following content:

- File Explorer: GA > GA-Projects > capstone > codes > video-capture.ipynb
- Code Editor:
 - Cell 1: `cap.release()`, `out.release()`, `cv.destroyAllWindows()`
 - Cell 2: `[[0.00194469]]`, FAIL
 - Cell 3: `[[0.9842731]]`, PASS
 - Cell 4: `[[0.9068822]]`, PASS
 - Cell 5: `[[4.869833e-18]]`
 - Cell 6: `[[1.]]`, PASS (highlighted with a red box)
 - Cell 7: `[[1.]]`, PASS
- Output Area: A video frame showing four blue and yellow robotic arms arranged in a 2x2 grid, labeled "Prediction".

Actual = 'Pass'

Video streamed from webcam
@ 720p, 5 fps

Issues:

1. Misclassification due to orientation
2. Misclassification due to hands being in the frame
3. Misclassification due to glare on certain parts of the product

Likely due to low variability in the 'Pass' data

Model Retraining

Variability in 'Pass' Data

2000 images of varying hand poses, product orientation, and lighting conditions for 'Pass'



Model Retraining

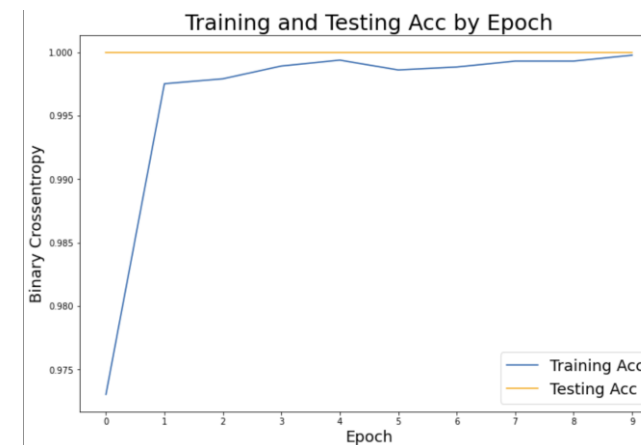
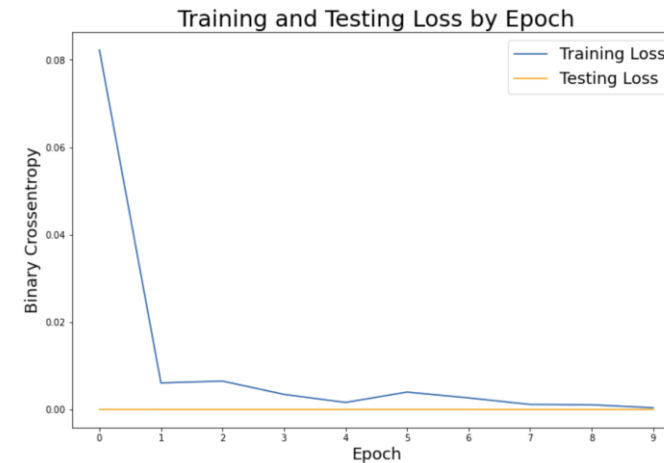
Custom CNN w/ Variability in 'Pass' Data

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 240, 320, 16)	448
max_pooling2d (MaxPooling2D)	(None, 120, 160, 16)	0
conv2d_1 (Conv2D)	(None, 118, 158, 64)	9280
max_pooling2d_1 (MaxPooling2D)	(None, 59, 79, 64)	0
flatten (Flatten)	(None, 298304)	0
dense (Dense)	(None, 64)	19091520
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

```
=====
```

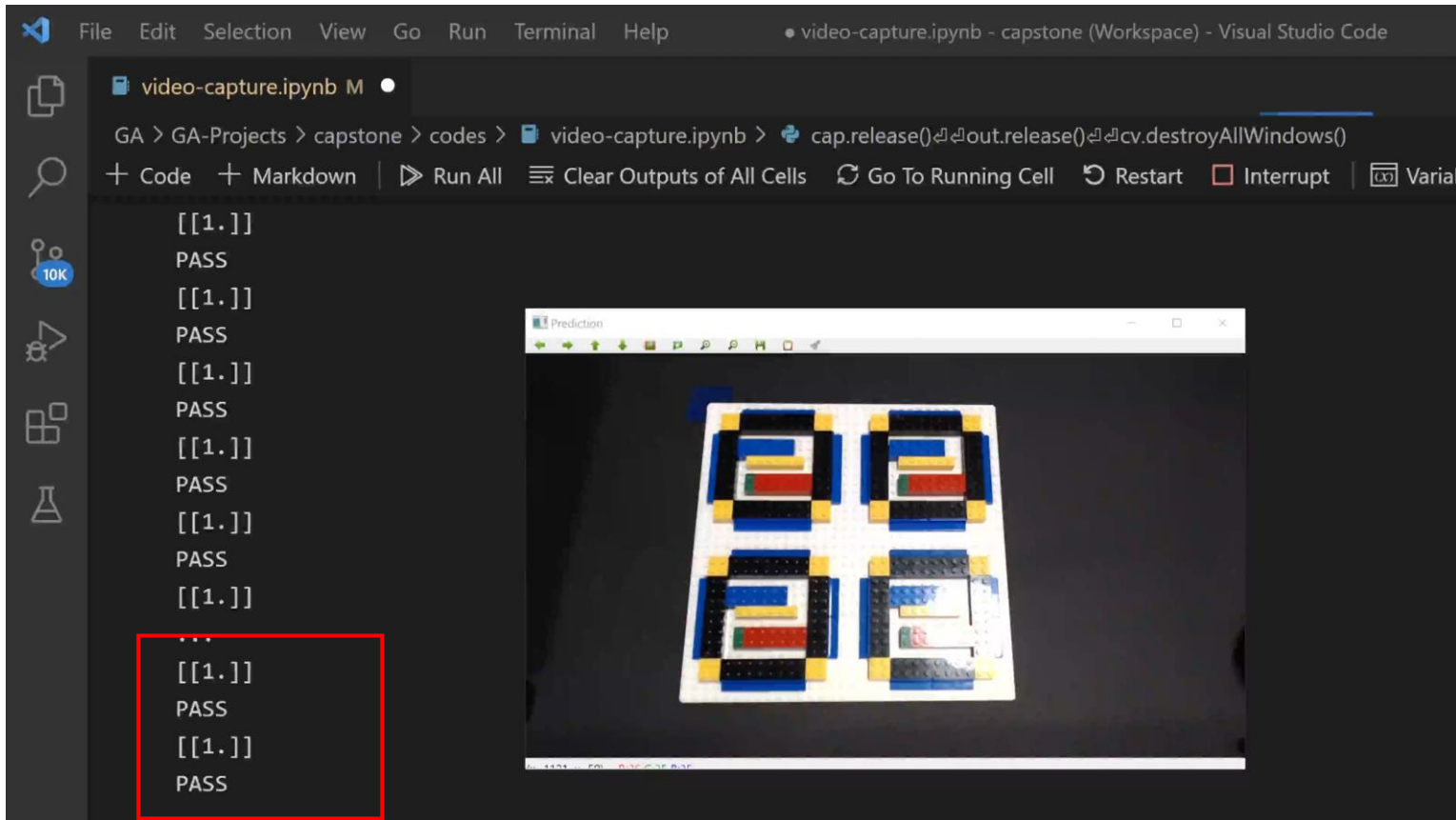
Total params: 19,103,361
Trainable params: 19,103,361
Non-trainable params: 0



After 10 epochs, we achieved 99.9% test accuracy!

Model Testing

Custom CNN w/ Variability in 'Pass' Data



Actual = 'Pass'

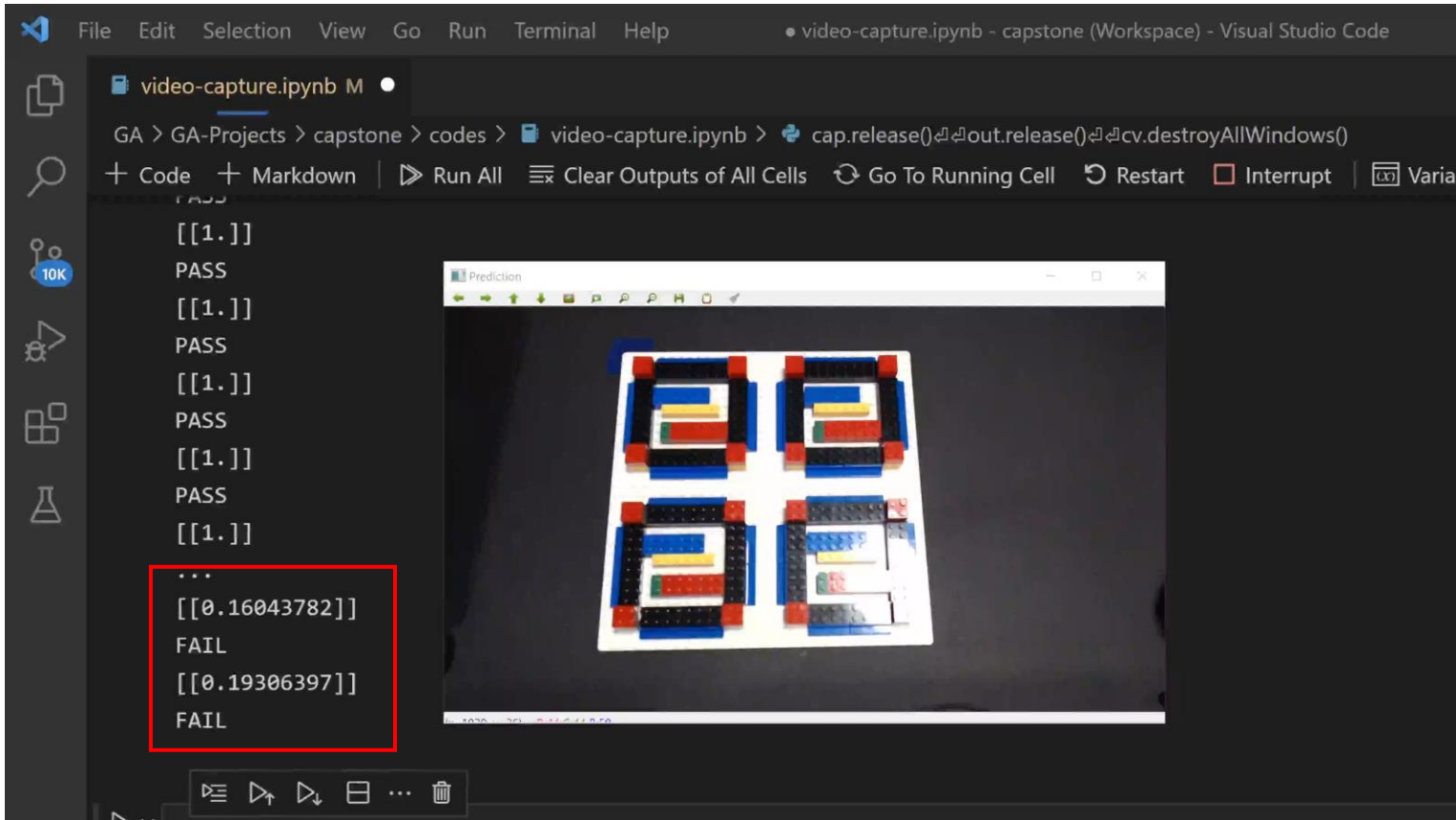
Video streamed from webcam
@ 720p, 5 fps

Fixed:

1. Correctly classifies despite different product orientation
2. Correctly classifies even with hand presence and movement
3. Correctly classifies at current lighting condition

Model Testing

Custom CNN w/ Variability in 'Pass' Data



```
GA > GA-Projects > capstone > codes > video-capture.ipynb > cap.release()cv.destroyAllWindows()

[[1.]]
PASS
[[1.]]
PASS
[[1.]]
PASS
[[1.]]
PASS
[[1.]]
PASS
...
[[0.16043782]]
FAIL
[[0.19306397]]
FAIL
```

Actual = 'Fail'

Video streamed from webcam
@ 720p, 5 fps

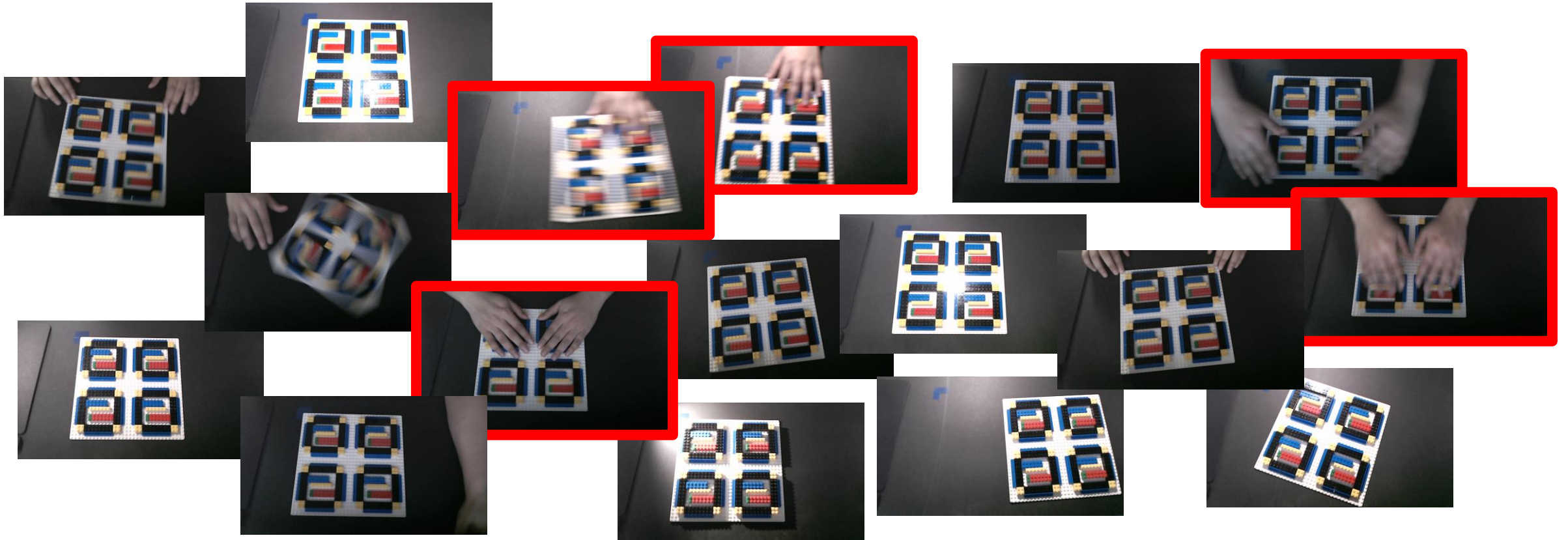
Issues:

1. Presence of hands = 'Pass'
2. Not sensitive to fails
 - Needed more than half of the corner blocks to be red before able to correctly detect 'Fail'.

Model Retraining

Variability in 'Pass' and 'Fail' Data

1. Images that were blocking the corner blocks were reclassified as 'Fail'
2. Performed similar variability poses/orientation for 'Fail' dataset.



Model Testing

Custom CNN w/ Variability in 'Pass' and 'Fail' Data

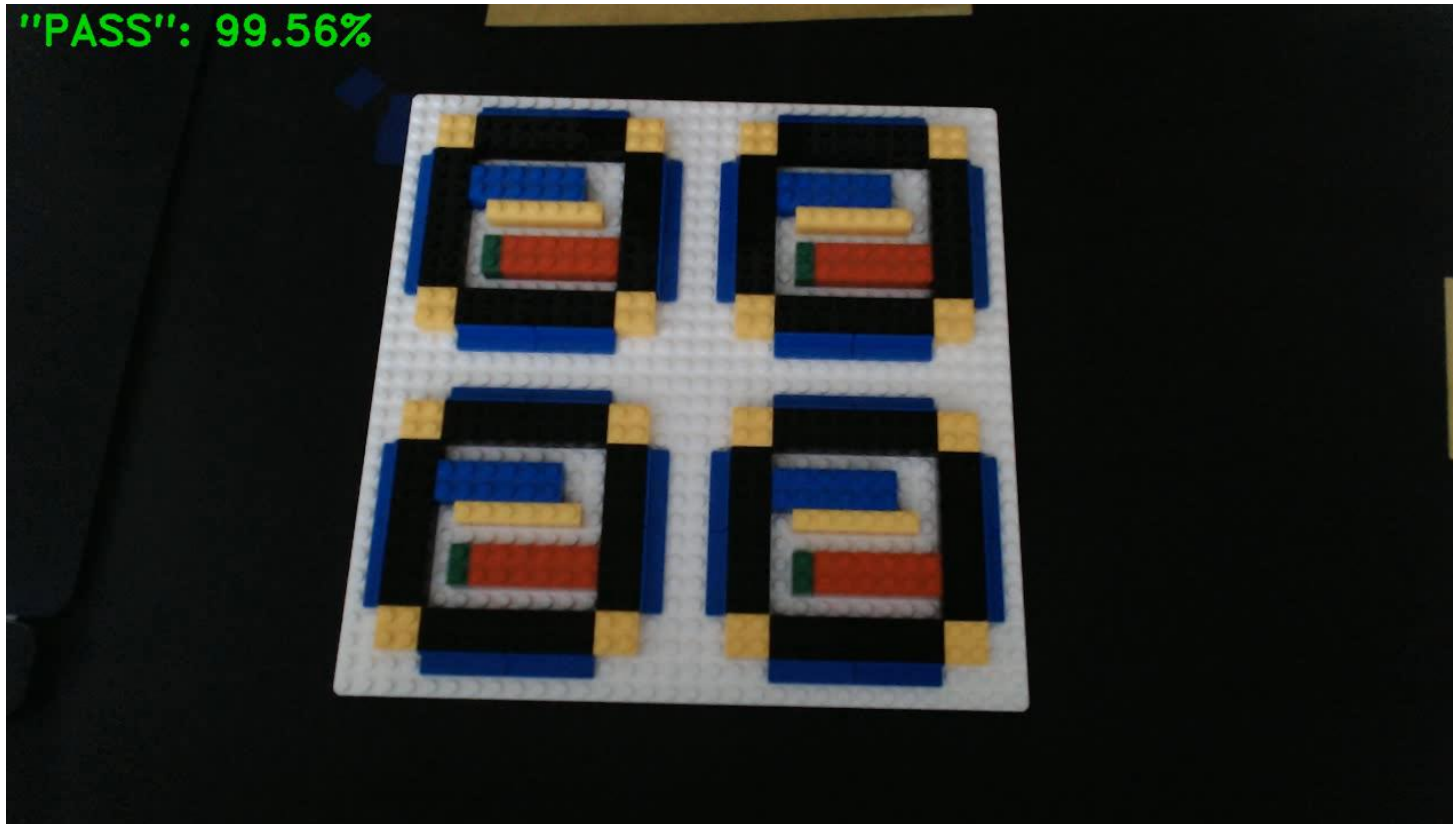
Video streamed from webcam
@ 720p, 5 fps

Fixed:

1. Hands blocking = 'Fail'
BUT, only in certain hand poses

Issues:

1. Too sensitive to orientation
2. Not sensitive to fails
3. Sensitive to lighting conditions

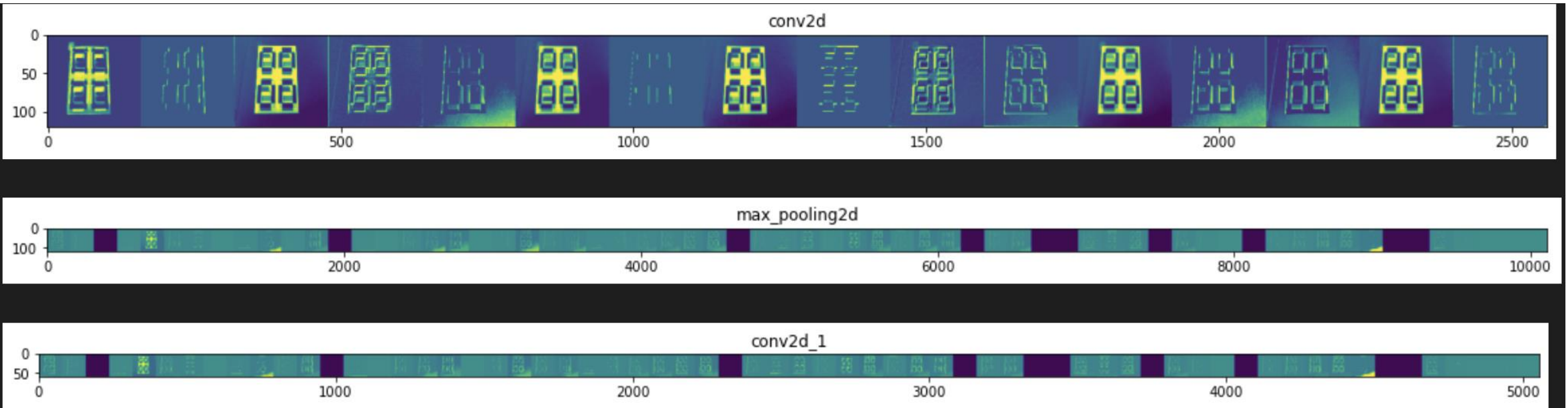
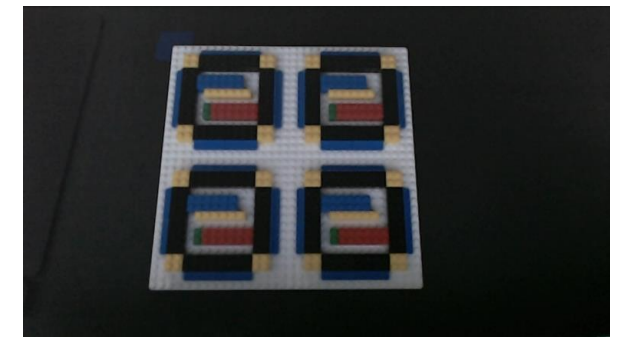


Actual = 'Pass'

Model Analysis

Model Analysis

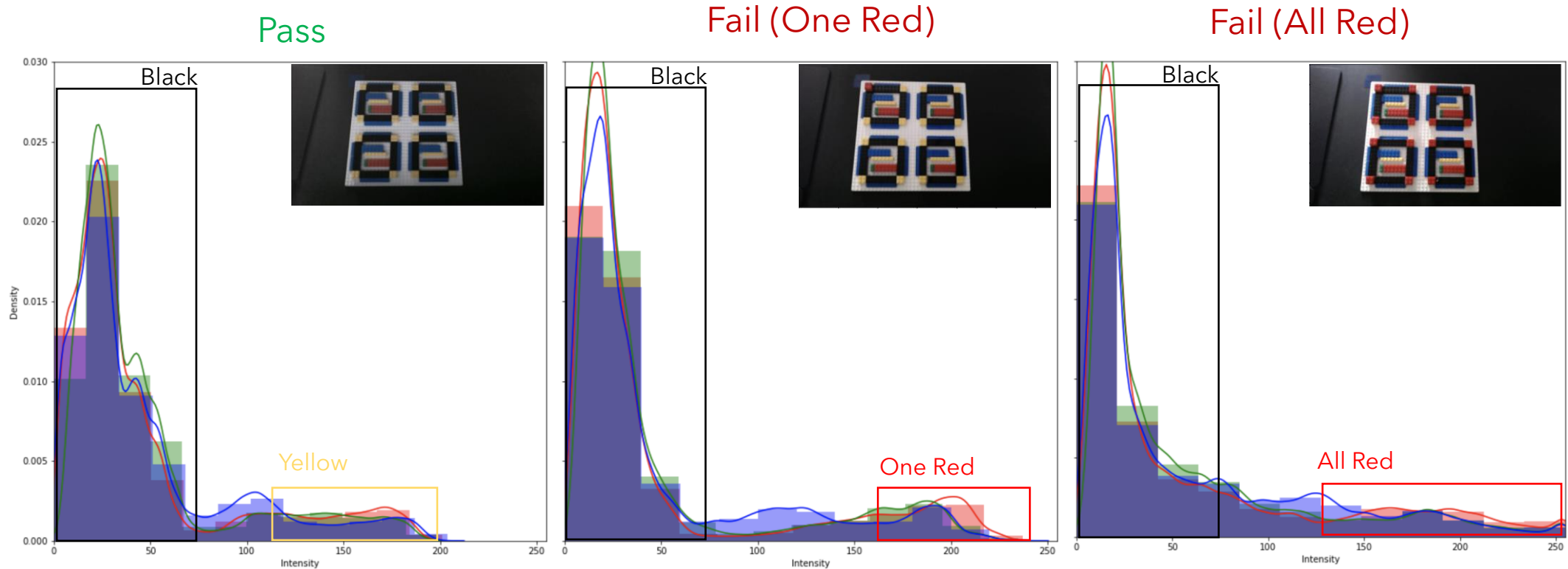
Feature Map



- Features extracted mostly consists of the white baseplate and the other components.
- Only ~3% of the image is required but whole image is used for training and classification.

Model Analysis

RGB Density Graphs



Note: Density vs. Intensity \approx Count vs. Colour Brightness

- Blacks are more pronounced as it takes up more than 50% of the image.
- Differences between classes are not as pronounced as they are of lower density.

Implementing Regions of Interest (ROIs)

Regions of Interest (ROIs)

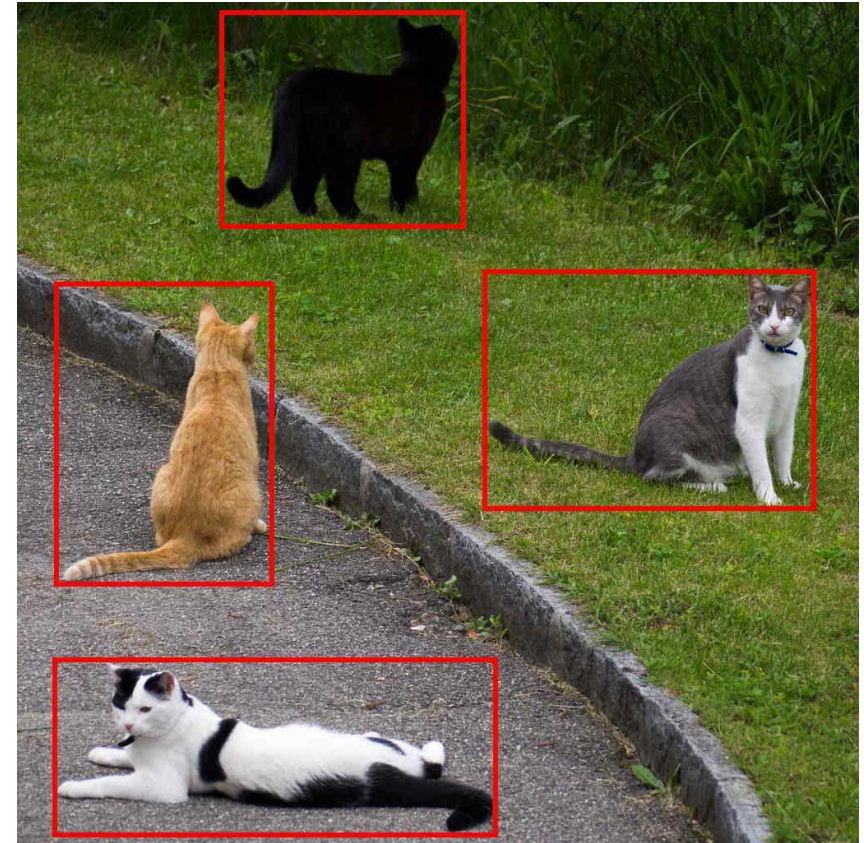
In configuring a region of interest (ROI), the image is reduced only to the area relevant for analysis.

Benefits:

- Reduces the image size
- Reduce memory required to evaluate each image
- Faster predictions
- Reduce noise/variability picked up from non-critical areas.

Model Implementation:

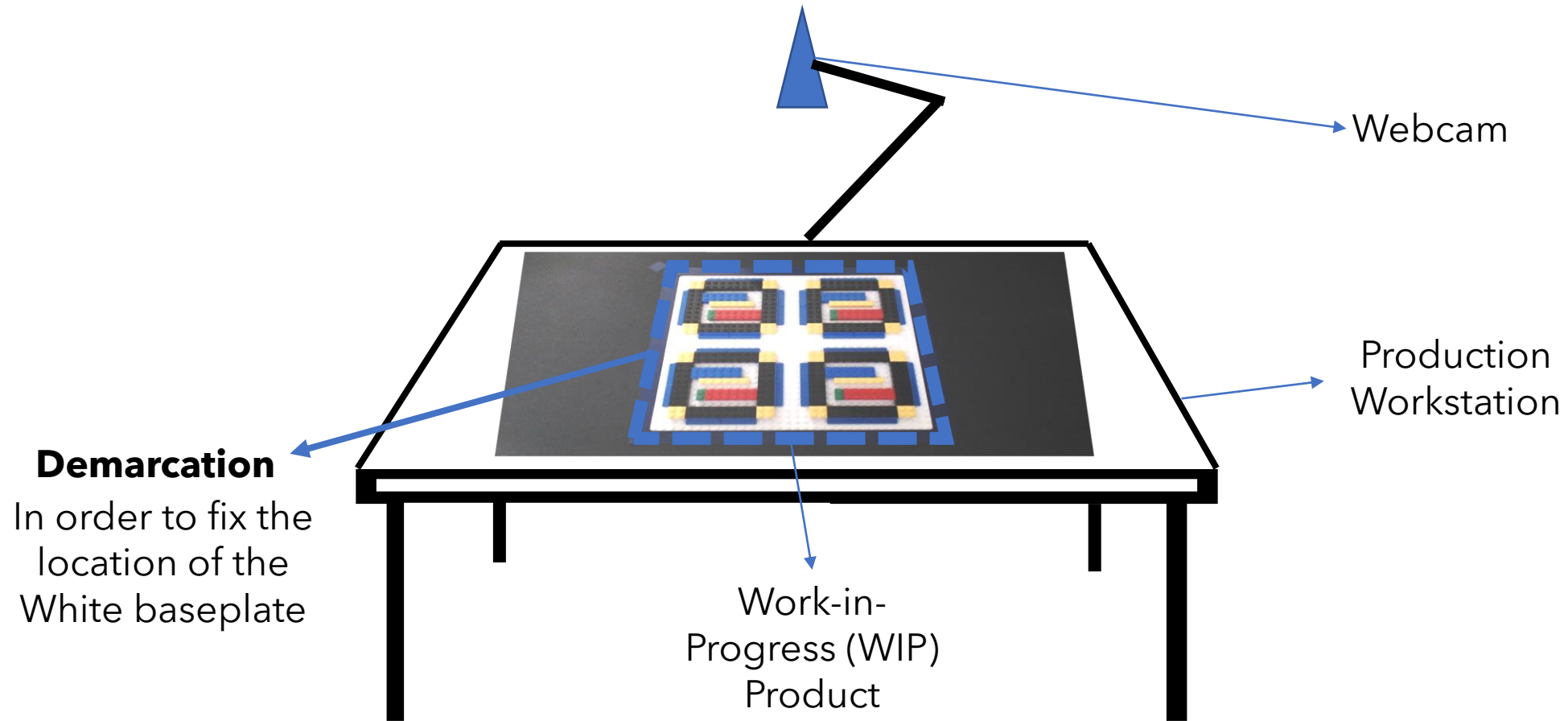
- Set fixed coordinates on the capture video for our model to analyse, given a fixed orientation of the product on the workstation.



Source: <https://erdem.pl/2020/02/understanding-region-of-interest-ro-i-pooling>

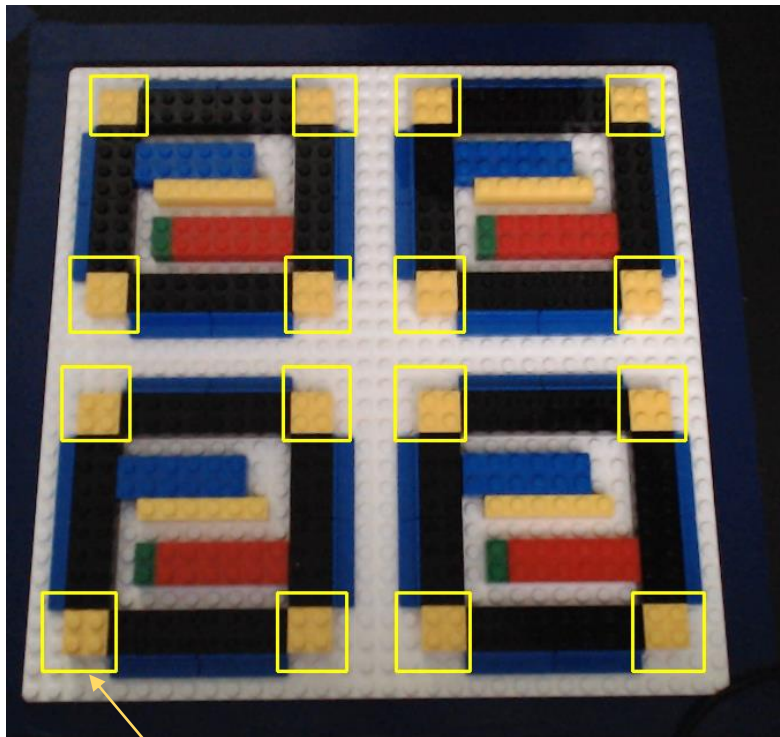
Regions of Interest (ROIs)

Setup Required

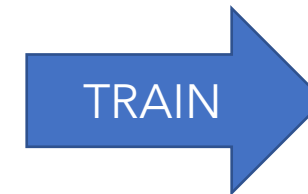
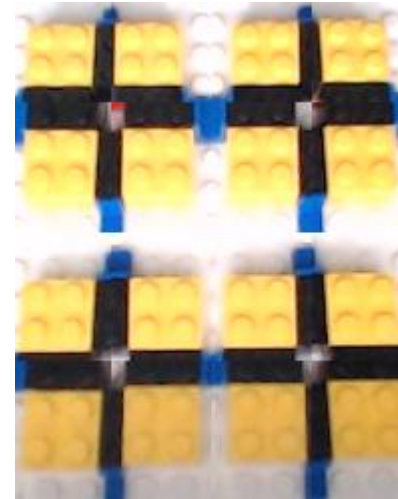
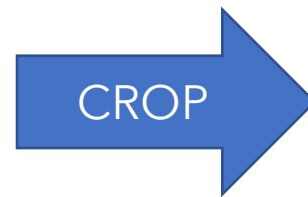


Model Preprocessing and Training

Fixed ROI



Set Target ROIs based on pixels



1: Pass - ~1.8k images

0: Fail - ~2.6k images

Model Training

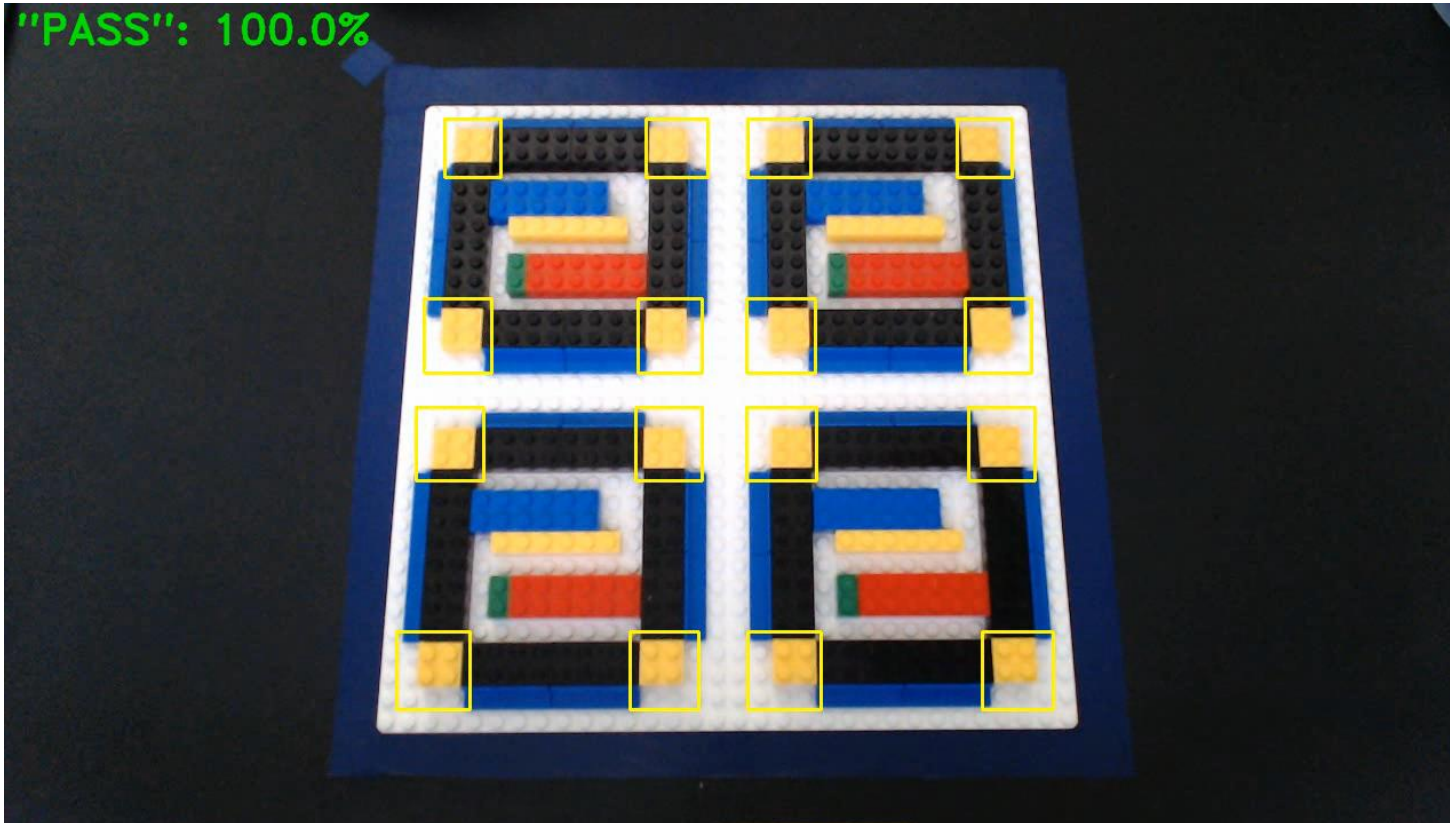
Fixed ROI Model Architecture

Lesser
Parameters!

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
conv2d_2 (Conv2D)	(None, 250, 200, 16)	448
max_pooling2d_2 (MaxPooling2D)	(None, 125, 100, 16)	0
conv2d_3 (Conv2D)	(None, 123, 98, 64)	9280
max_pooling2d_3 (MaxPooling2D)	(None, 61, 49, 64)	0
flatten_1 (Flatten)	(None, 191296)	0
dense_3 (Dense)	(None, 64)	12243008
dropout_2 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2080
dropout_3 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 1)	33
=====		
Total params: 12,254,849		
Trainable params: 12,254,849		
Non-trainable params: 0		

Model Testing

Fixed ROI Dataset



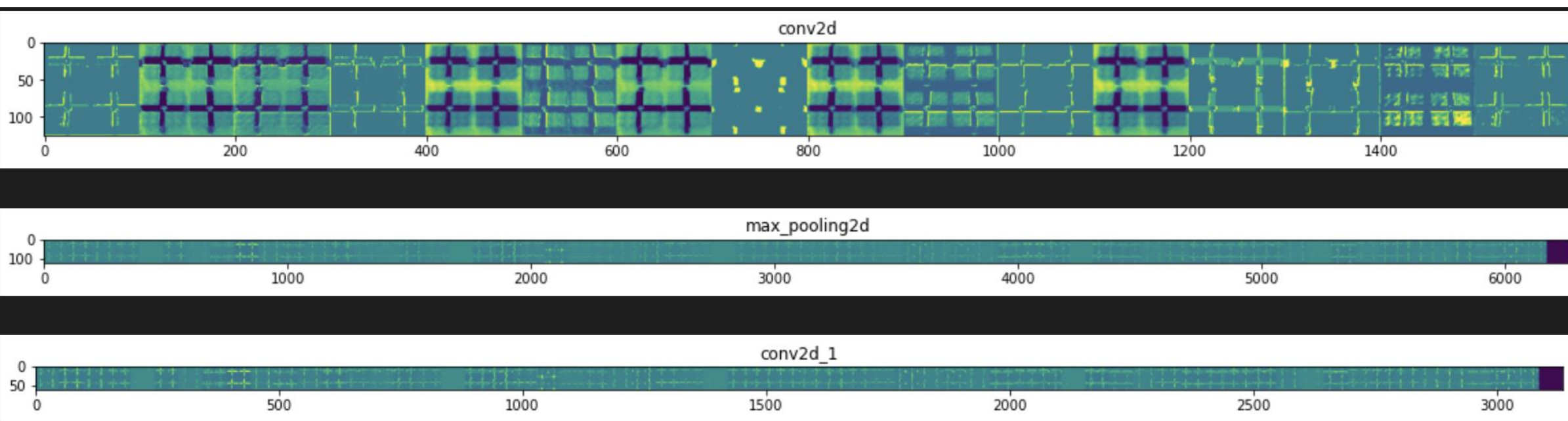
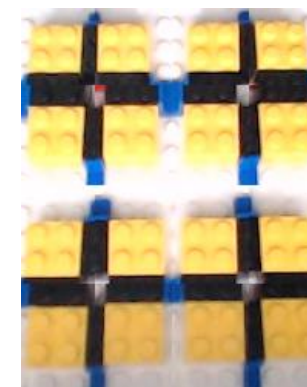
Video streamed from webcam
@ 720p, 5 fps

Fixed:

1. Easy to obtain accurate predictions.
 - Model will work so long as all the corner blocks are within ROIs.
2. Sensitive to fails
 - 1 red block = 'Fail'.

Fixed ROI Model Analysis

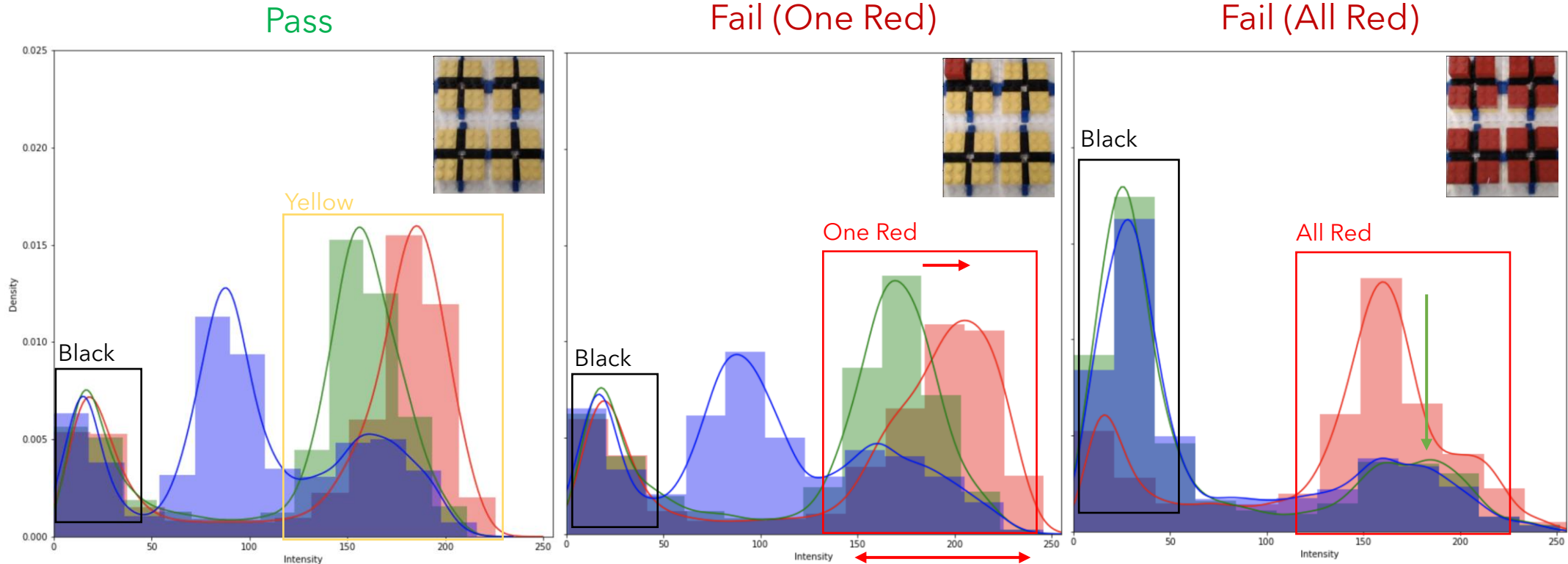
Feature Map



- Relevant features are extracted and preserved up till the last Feature Extraction layer
- A higher percentage (~50%) of the image is used for classification.

Fixed ROI Model Analysis

RGB Density Graphs

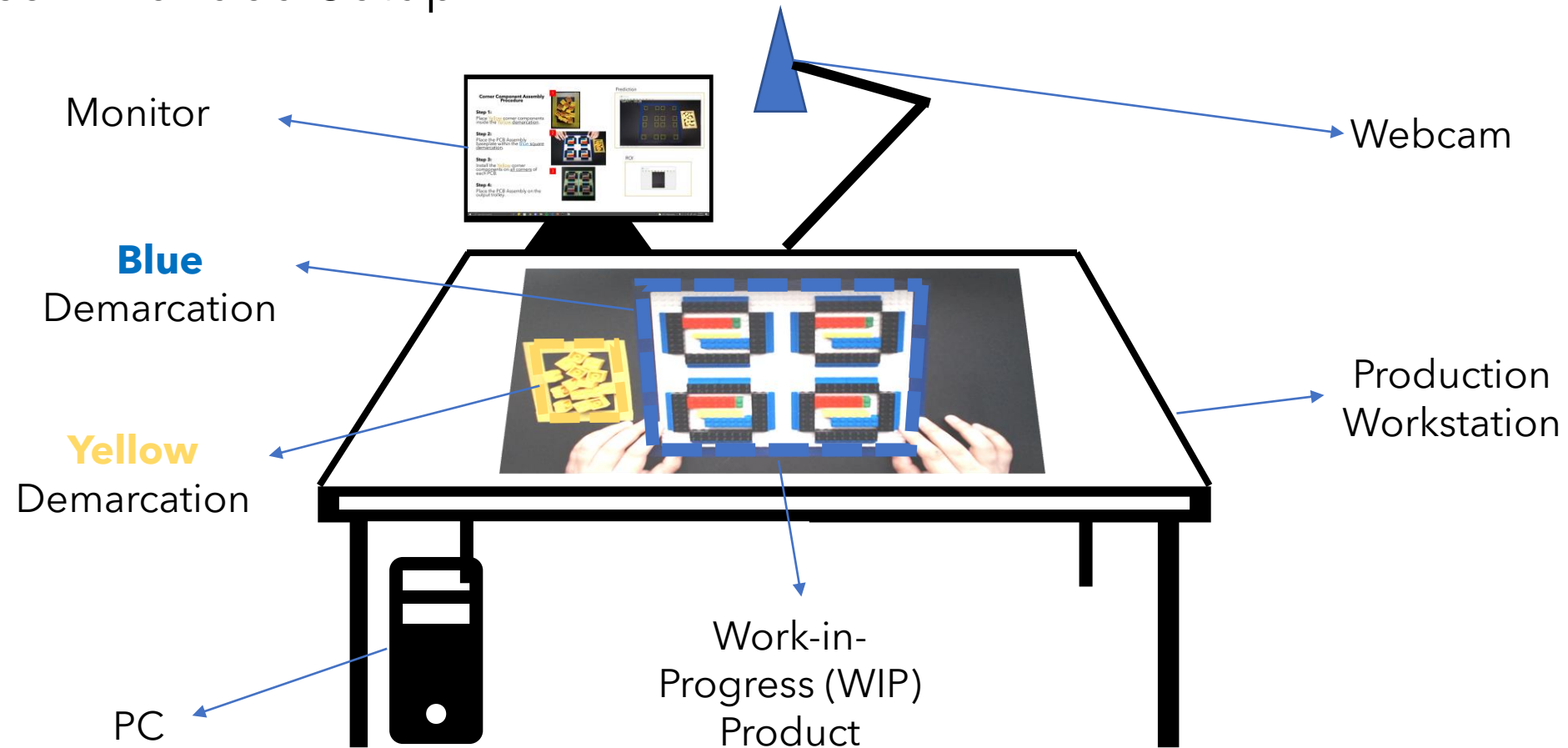


- Previous blacks are less pronounced (lower density).
- Model is now able to pick out corner block colours better.
 - E.g. when **One Red** is introduced, the red hue graph peak increases in intensity and has a larger area under the graph (higher overall density).

Production Model Implementation

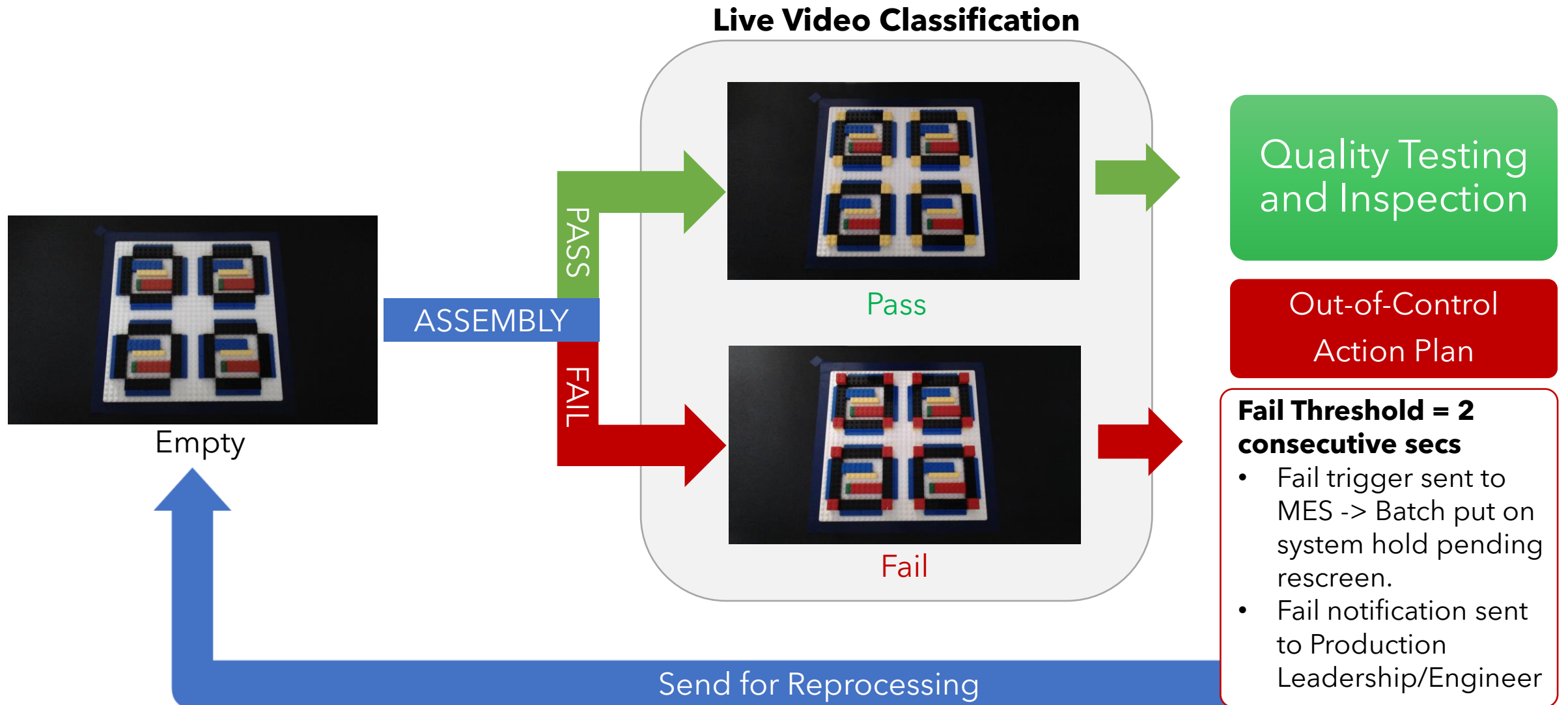
Production Model Implementation

Recommended Setup



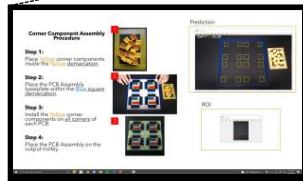
Production Model Implementation

Workflow



Production Model Implementation

Production Monitor



Corner Component Assembly Procedure

Step 1:

Place **Yellow** corner components inside the **Yellow** demarcation.



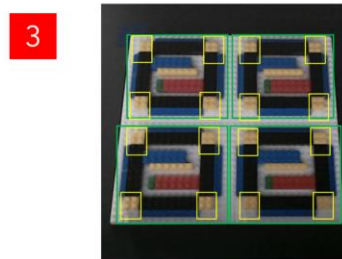
Step 2:

Place the PCB Assembly baseplate within the **Blue** square demarcation.



Step 3:

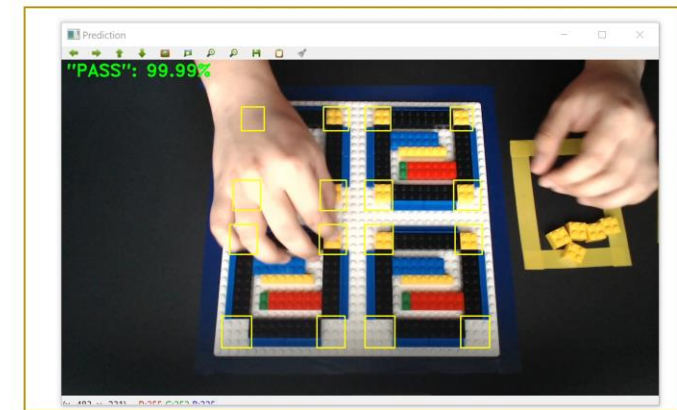
Install the **Yellow** corner components on all corners of each PCB.



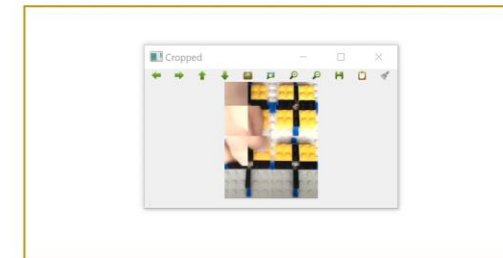
Step 4:

Place the PCB Assembly on the output trolley.

Prediction



ROI



Conclusions

Conclusion

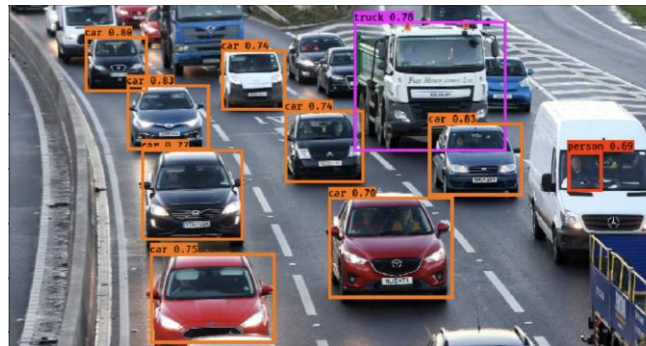
Current model is highly accurate and is sensitive to 'Fail'. This is sufficient to deploy to production to fix the current problem.

However, there are some downsides with the current model and its implementation:

1. Requires strict human adherence to procedure.
 - i.e. Putting the white baseplate exactly within the blue box, aligning corner components to ROI.

Proposed Future Improvement

- I. This can be improved on by **increasing the pixels for each ROI**, allowing for some flexibility in terms of baseplate orientation.
- II. Implement **object detection** that can capture the baseplate in a bounding box and subsequently, determine ROI based on the orientation of the baseplate.



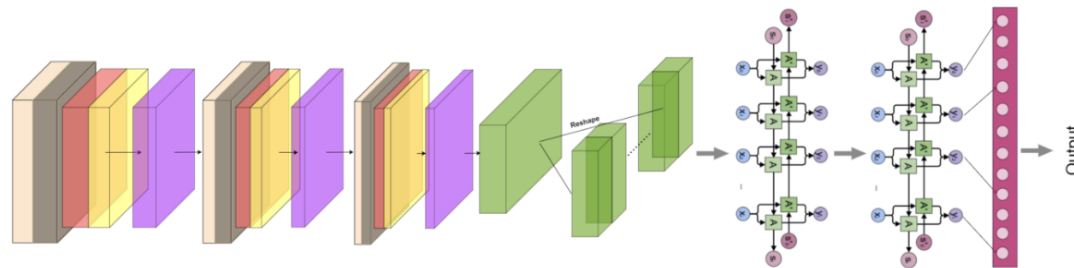
Conclusion

However, there are some downsides with the current model and its implementation:

2. Predictions flickers quickly from 'Pass' to 'Fail' in certain instances
 - i.e. When blocking multiple ROIs with hands during product assembly.

Proposed Future Improvement

- I. **Deploy a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN)**, also know as CRNN, model that takes into account the predictions of the previous few frames before predicting current frame
 - 'Fails' will be predicted only if it has happened over several frames instead of instantaneous.



Proposed CRNN

Source: <https://towardsdatascience.com/an-approach-towards-convolutional-recurrent-neural-networks-a2e6ce722b19>

Other Future Works

- Model can be pushed to the next level by implementing another class, "Ongoing", to represent unfinished product assembly.
- The images for the other components can also be fed to another model with a different ROI to detect defects from upstream processes.
- Implement object detection to detect foreign materials (red blocks in this case) within the workstation, thus, further reducing the risk of wrong assembly.

The background is a dark gray field filled with a complex, low-poly geometric pattern of various shades of gray, creating a 3D effect. A thin, white, hand-drawn style rectangular border frames the central text. Below the text, a single, slightly wavy white horizontal line is drawn.

Thank You