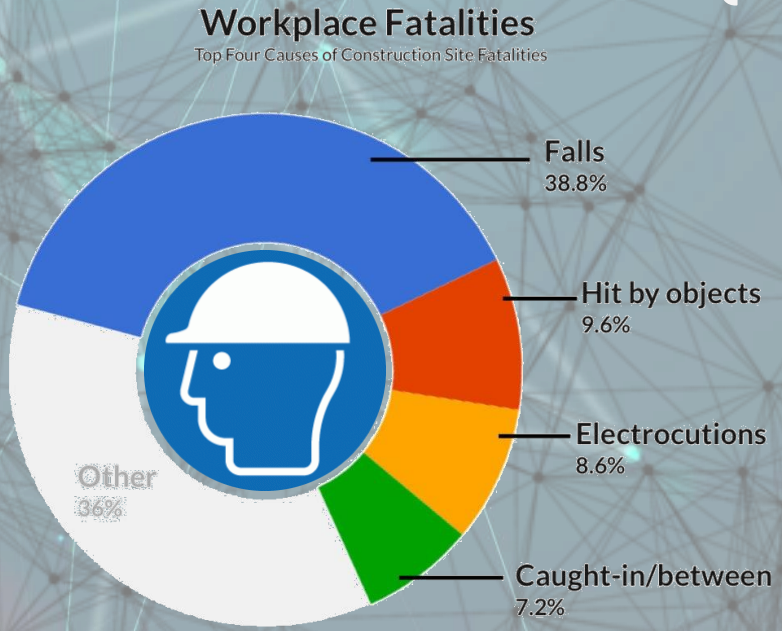
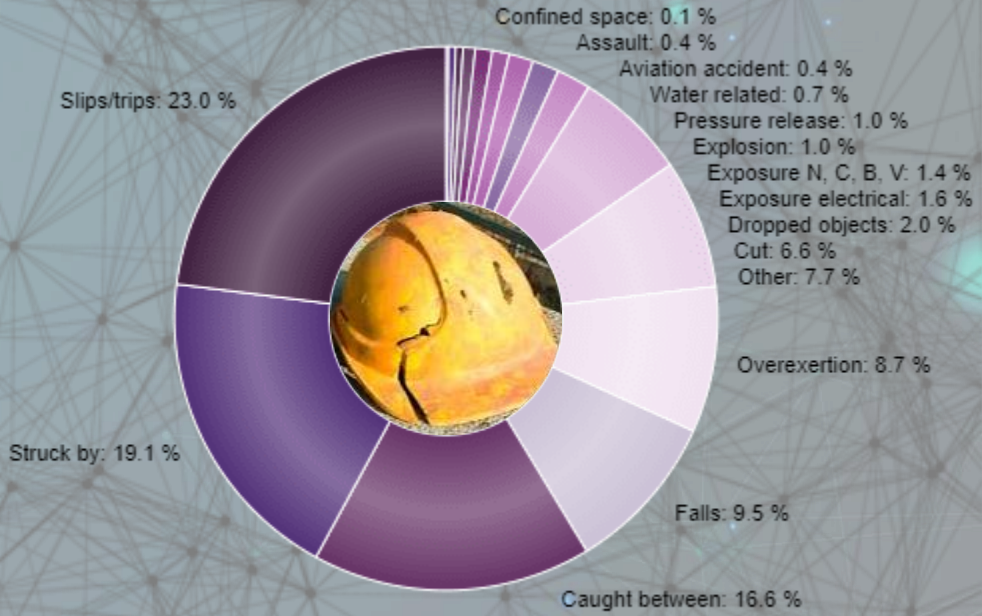
AI SAFETY OFFICER

Vitaliy Yashar

NAYA College Final ML Project
BootCamp, December 2021

WORK FATALITIES AND LOST DAY WORK CASE (LDWC)





MAIN GOALS

Our goal is to train a custom deep learning model to **detect whether a person is or is not wearing a safety helmet**

Hence, the problem is both
Detection/Binary Classification

In case of successful deployment, the safety helmet detector I am building here today could potentially be **used to help ensure your personal safety** rules is being responsibly followed

By the end of this product development, we should consider **deploying the model on portable detection device** which will be mounted on the entrance to highly hazardous production areas

AI SAFETY OFFICER

Deep Learning Based

FACE RECOGNITION

01

OpenCV Keras DNN Model will be used

FACE RECOGNITION PREFACE

Real-time face detection

The first real-time face detector was introduced with the **Viola-Jones** algorithm in **2001** and it segments the photo in multiple subsections and tries to find haar-like features inside each subarea. Performance can be improved by using cascade and adaptive boosting. Although this algorithm reveals a very high accuracy detecting faces, it still contains a lot of false positives and fortunately today there are better alternatives such as Support Vector Machines, Naïve Bayes Classifiers, Deep Neural Networks (DNN), etc.
I chose to explored both haarcascades and the DNN for facial recognition using the OpenCV library.

OpenCV's DNN Module

OpenCV is an open source computer vision and machine learning library made in C++.
It contains out-of-the-box haarcascades, but since version 3.3 there is pre-trained deep learning face detector. The DNN face detector is based on the Single Shot Detector (SSD) framework using a ResNet-10 like base Network.

FACE RECOGNITION PREFACE

01	02	03	04
Download the **face recognition** files from the OpenCV repository	The first step is to load the neural network **Deploy.prototxt** file defines the network architecture and **res10-300x300-ssd-iter-140000.caffemodel** has the weights of the layers	Run the input through DNN, check if there are any detections and for each detection the correspondent confidence is extracted. **Confidence>threshold** than we draw a box around the face and extract it's x,y	DNN face detector from OpenCV is highly accurate and easy to use. Real-time algorithm can be used in almost every scenario despite of the conditions of the source, the lighting and image resolution, face pose or facial expressions.

SAFETY HELMET CLASSIFIER

02

Keras/TensorFlow
Classifier will be trained

METHODOLOGY TO BE IMPLEMENTED

STAGE 1

- Load Safety Helmet DataSet
- Train Safety Helmet Detection Classifier with Keras/TensorFlow
- Save The Classifier Model to Computer

STAGE 2

- Load The Classifier Model from the Computer
- Detect Faces on Live Video Stream within OpenCV facility
- Extract Each Face ROI
- Apply Classifier Model to Each ROI
- Return the Notification and Alarm if the Safety Helmet is Off

DATASETS AND MODELS

A.

Cropped Dataset

800 Images : with helmet
1200 Images : without helmet

The without helmet images were different face pictures from wiki, it will allow us to further train the data enriched model

B.

Manipulated Dataset Enrichment Technique

1700 Images : with helmet
1900 Images : without helmet

DATASETS AND MODELS

C.

Data Enriched
Dataset
A+B

A. CROPING THE FACES OF FULL BODY PHOTOS WITH HARDHATS (Cropper Face.ipynb)

1400 Full body Images

»

723 Cropped Images

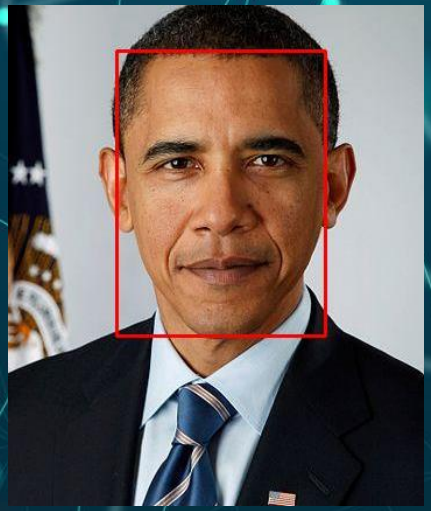
Full body hardhat Dataset found

HaarCascade based Algorithm which runs on the specified images folder

Saves the cropped faces

Wiki faces Dataset will be implemented as without safety helmet class for training

B. ARTIFICIALLY MANIPULATED DATASET
IMAGE PREPROCESSING



Start with wiki image without safety helmet

Face detection to find the **bounding box** of Face

Object Localization
Extract the face **Region of Interest (ROI)**

Detect **Facial Landmarks** using dlib, so that we know where to place the hardhat on top of the face

Transparent image of a helmet will be applied on each face

B. ARTIFICIALLY MANIPULATED DATASET

(Safety_hat_generator.ipynb)

5000 Wikipedia images

1692 Generated images

Start with wiki image without safety helmet

Face detection to find the **bounding box** of each face

Extract the face **Region of Interest (ROI)**

Detect Facial Landmarks using dlib so that we know where to place hardhat on the face

Transparent image of a helmet will be applied on each face and rubbish images removed manually

MODEL EVOLUTION

Learning Rate Adjustment

Various LR were implemented
1e-3 to 1e-5

Augmentation

Rotation_range=20/zoom_range=0.15/width_shift_range=0.2/
height_shift_range=0.2/shear_range=0.15/
horizontal_flip=True/fill_mode="nearest"

Adding more layers to HM

AveragePooling2D/Flatten/
Dense(128, activation="relu")
~~Dense(2, activation="softmax")~~
Dense(2, activation="sigmoid")

Dropout

Dropout(0.5)
Before the sigmoid last layer activation

CheckPointing

Only the best performed model
saved during each epoch runtime

CNN ARCHITECTURE CONSTRUCTION

BaseModel: MobileNetV2 is a light weighted convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers

```
# Load the MobileNetV2 network, ensuring the head FC layer sets are settled off
1. baseModel = MobileNetV2(weights="imagenet", include_top=False,
2. Input_tensor=Input(shape=(224, 224, 3)))
# Construct the head of the model that will be placed on top of the BaseModel
1. headModel = baseModel.output
2. headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
3. headModel = Flatten(name="flatten")(headModel)
4. headModel = Dense(128, activation="relu")(headModel)
5. headModel = Dropout(0.5)(headModel)
6. headModel = Dense(2, activation="sigmoid")(headModel)

# Place the head FC model on top of the base model (this will become the final Trained model
1. model = Model(inputs=baseModel.input, outputs=headModel)
```

Problem Type	Last Layer Activation	Loss Function
Binary Classification	Sigmoid Returns Probs of binary class	Binary_crossentropy
Multiclass, single-lable classification	**Softmax** Returns Probs of each class	Categorical_crossentropy
Multiclass, multilabel classification	Sigmoid	Binary_crossentropy

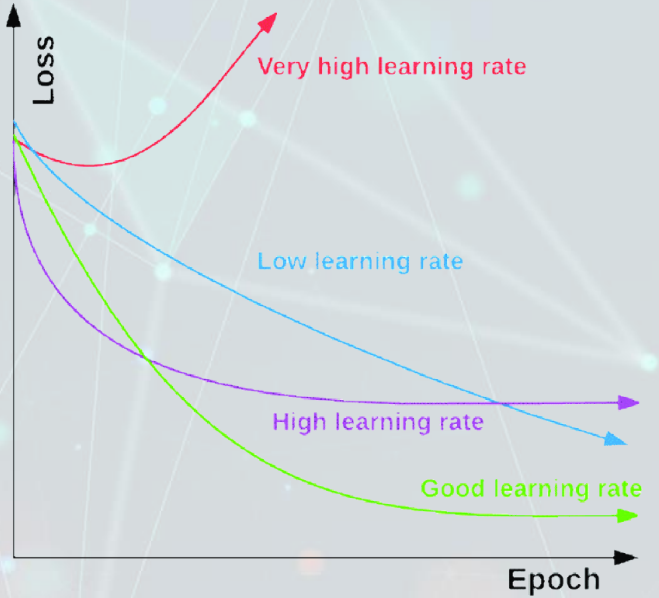
TRAINING PROCESS AND FURTHER DEPLOYMENT

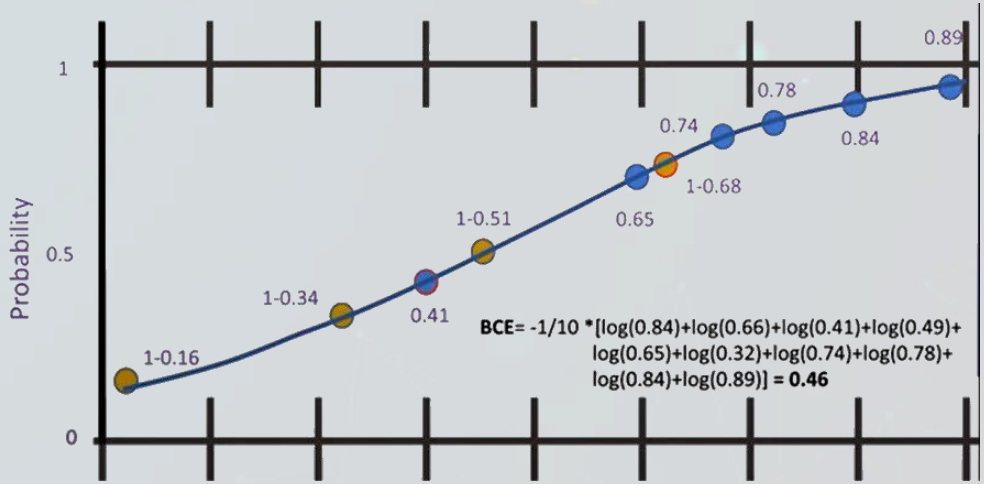
Training: Here we'll focus on loading our safety helmet detection dataset from GoogleDrive, training a model (using Keras/TensorFlow) on this dataset, and then saving the safety helmet detector model to disk.

Deployment: Once the safety helmet detector is trained, we can then move on to loading it to our OpenCV facility, performing face detection, and then classifying each face as hat or without hat class.

Keras .h5 model will be saved and than implemented within OpenCV to recognize the class.

LR ADJUSTMENT AND LOSS METRICS

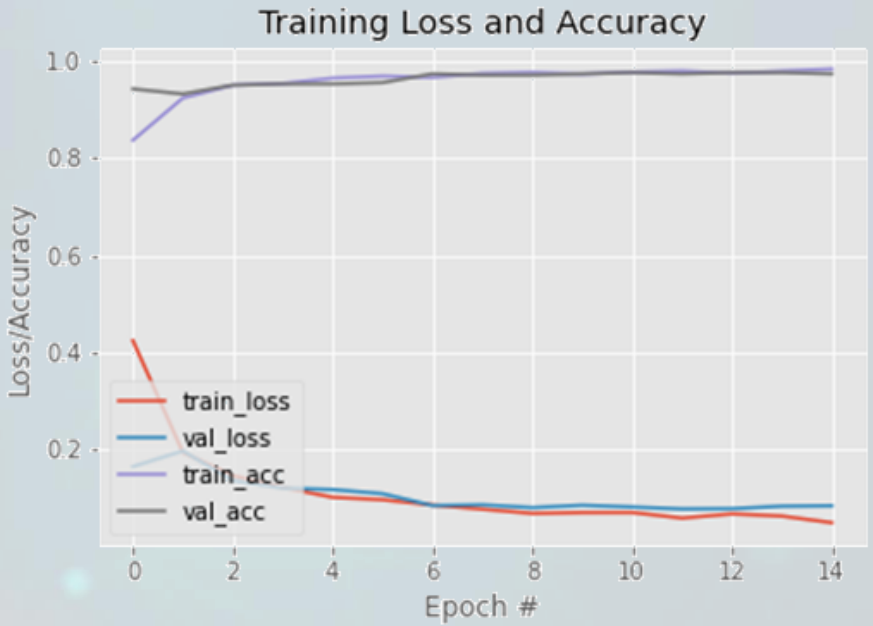


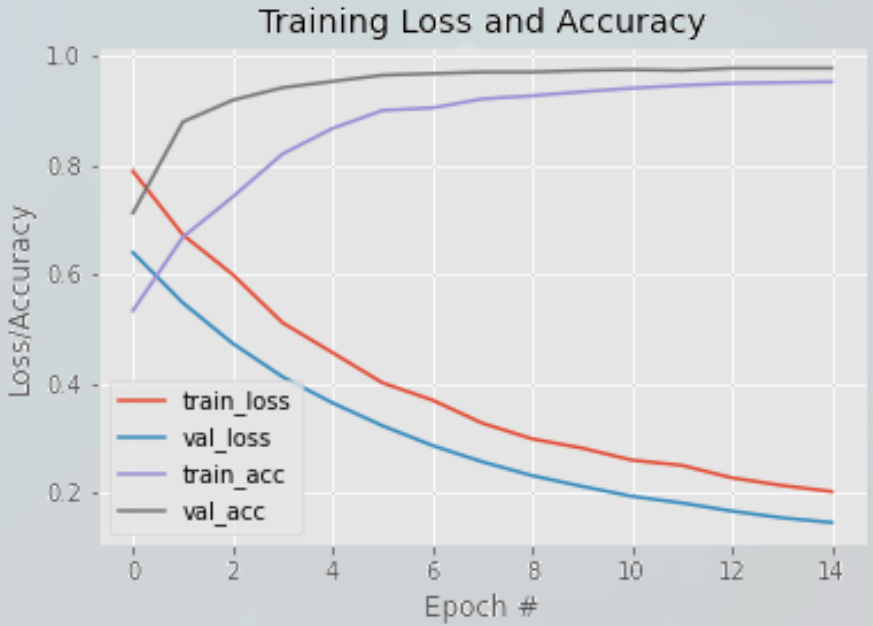


Binary_crossentropy = LogLoss

Accuracy/Loss	Low Loss	High Loss
Low Accuracy	Lot of Small errors	A lot of big errors
High Accuracy	Few Small errors	Few big errors

RESULTS AND EVALUATION





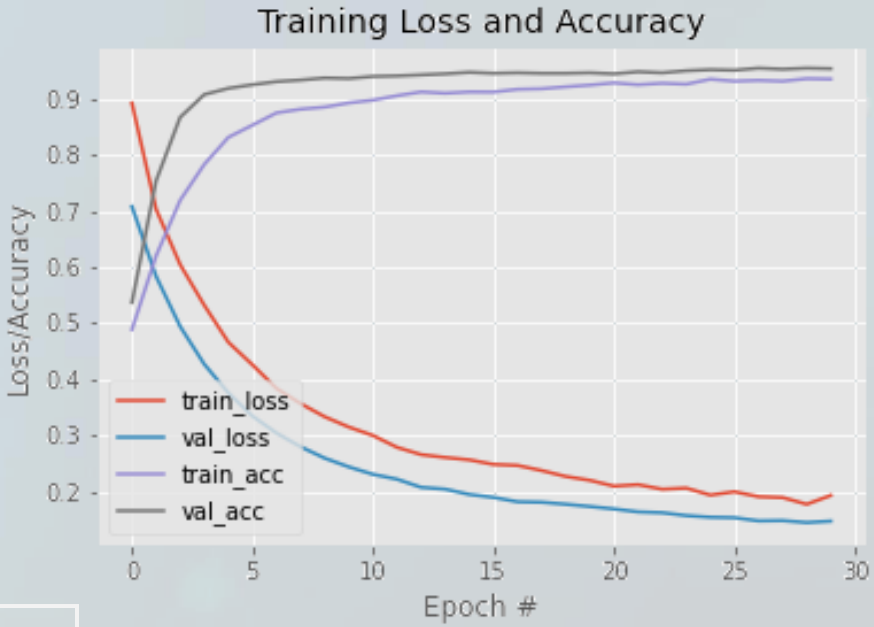
A. DataSet

LR	EP	Batch	Acc.	Loss
1e-3	15	100	0.9840	0.0476

B. DataSet

LR	EP	Batch	Acc.	Loss
⩔1e-5	15	⩔50	⩓0.9536	0.2019

FINAL MODEL – C. DATASET RESULTS



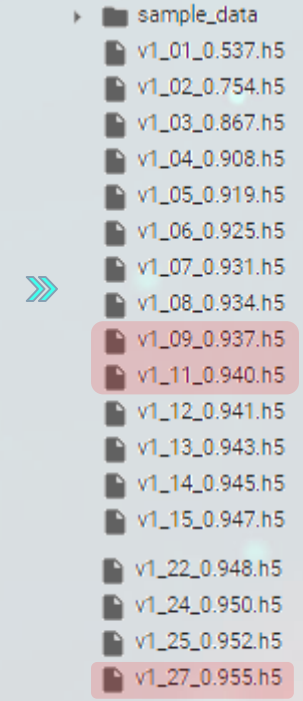
Training Accuracy	Validation Accuracy	Difference	Reason?
⇈	⇈	NO	Perfectly Fit
⇊	⇊	NO	Under Fit
⇈	⇊	High	Over Fit

LR	EP	Batch	Acc.	Loss
1e-5	30+CP	100	0.9361	0.1777

CheckPointing

```
86/86 [==============================] - 190s 2s/step - loss: 0.8926 - accuracy: 0.4890 - val_loss: 0.7083 - val_accuracy: 0.5373
Epoch 2/30
86/86 [==============================] - 190s 2s/step - loss: 0.7036 - accuracy: 0.6202 - val_loss: 0.5840 - val_accuracy: 0.7539
Epoch 3/30
86/86 [==============================] - 187s 2s/step - loss: 0.6048 - accuracy: 0.7193 - val_loss: 0.4946 - val_accuracy: 0.8673
Epoch 4/30
86/86 [==============================] - 188s 2s/step - loss: 0.5324 - accuracy: 0.7832 - val_loss: 0.4273 - val_accuracy: 0.9078
Epoch 5/30
86/86 [==============================] - 192s 2s/step - loss: 0.4665 - accuracy: 0.8315 - val_loss: 0.3762 - val_accuracy: 0.9189
Epoch 6/30
86/86 [==============================] - 190s 2s/step - loss: 0.4261 - accuracy: 0.8532 - val_loss: 0.3347 - val_accuracy: 0.9253
Epoch 7/30
86/86 [==============================] - 190s 2s/step - loss: 0.3839 - accuracy: 0.8752 - val_loss: 0.3045 - val_accuracy: 0.9309
Epoch 8/30
86/86 [==============================] - 196s 2s/step - loss: 0.3570 - accuracy: 0.8817 - val_loss: 0.2800 - val_accuracy: 0.9336
Epoch 9/30
86/86 [==============================] - 191s 2s/step - loss: 0.3335 - accuracy: 0.8854 - val_loss: 0.2597 - val_accuracy: 0.9373
Epoch 10/30
86/86 [==============================] - 191s 2s/step - loss: 0.3152 - accuracy: 0.8929 - val_loss: 0.2442 - val_accuracy: 0.9364
Epoch 11/30
86/86 [==============================] - 195s 2s/step - loss: 0.3002 - accuracy: 0.8978 - val_loss: 0.2309 - val_accuracy: 0.9401
Epoch 12/30
86/86 [==============================] - 191s 2s/step - loss: 0.2790 - accuracy: 0.9057 - val_loss: 0.2223 - val_accuracy: 0.9410
Epoch 13/30
86/86 [==============================] - 191s 2s/step - loss: 0.2661 - accuracy: 0.9125 - val_loss: 0.2078 - val_accuracy: 0.9429
Epoch 14/30
86/86 [==============================] - 190s 2s/step - loss: 0.2610 - accuracy: 0.9102 - val_loss: 0.2045 - val_accuracy: 0.9447
Epoch 15/30
```

```
Epoch 27/30
86/86 [==============================] - 191s 2s/step - loss: 0.1911 - accuracy: 0.9330 - val_loss: 0.1480 - val_accuracy: 0.9548
Epoch 28/30
86/86 [==============================] - 191s 2s/step - loss: 0.1896 - accuracy: 0.9316 - val_loss: 0.1484 - val_accuracy: 0.9530
Epoch 29/30
86/86 [==============================] - 191s 2s/step - loss: 0.1777 - accuracy: 0.9361 - val_loss: 0.1451 - val_accuracy: 0.9548
Epoch 30/30
86/86 [==============================] - 191s 2s/step - loss: 0.1933 - accuracy: 0.9354 - val_loss: 0.1476 - val_accuracy: 0.9539
```



SUMMARY

A. DATASET

Cropped Images

Best Performed

B. DATASET

Manipulated DataSet

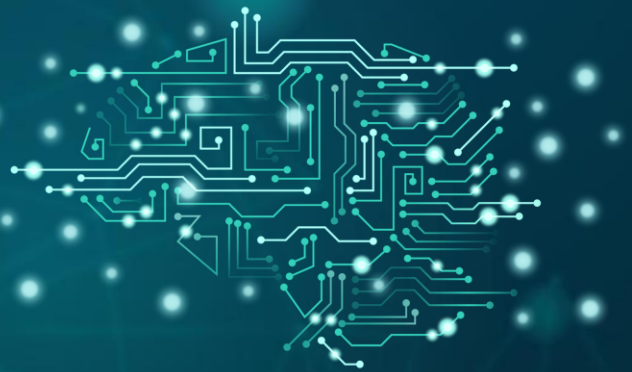
Average Performance, but is not feasable to apply it on live VideoStream

C. DATASET

United Images

93% Accuracy, alongside with enrichment algorithm to further score improovement

MODEL EVOLUTION AND BEST SELECTION



0.984%

Probably overfited model

0.936%

Also excellent performance and data enrichment algorithm for further work

ISSUES AND FURTHER RESEARCH

Having limited training data

Class being artificially generated

To improve our detection model further, we should gather *actual images* (rather than artificially generated images) of people with hardhats

Gather “confusing” image, which will bring the classifier into mode of recognizing the person is wearing a mask when in fact they are not — potential examples include back snaps or additional hat styles

2 steps detection is problematic, when hardhat can cover part of the face. If enough of the face is covered, the face cannot be detected, and therefore, the AI Safety Officer wouldn’t applied

SUMMARY AND PRODUCT BENEFITS

01

EASILY MOUNTED

Could be installed on portable machines such as NVIDIA Jetson

02

ONLINE RECOGNITION

Should be applied on the entrance to hazardous areas

03

ALERTS APPLIED

Identifies the person and sound alerts implemented whether the safety helmet is not there

04

UNBIASED

Trained on multinational Datasets

0,936+ Accuracy Achieved

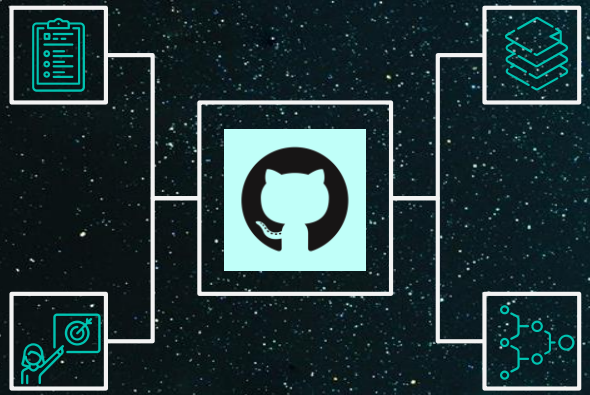
For the Best performed Model, applied on enriched DATA

JUPYTER NOTEBOOKS

Face Recognizer & Cropper
Safety helmet initializer
Keras Training Algorithm
OpenCv Deployment Notebook

Slides

The following presentation



Datasets

Links in Resources

Model

Model.h5 Keras best scored Model

RESOURCES

DATASETS

- https://data.mendeley.com/datasets/9rcv8mm682/3
- https://drive.google.com/file/d/1qWm7rrwvjAWs1slymbrLaCf7Q-wnGLEX/view
- https://makeml.app/datasets/hard-hat-workers

Thank You!

Do you have any questions?

vitttorio88@gmail.com
+972 545 464 615

https://github.com/YasharDS

https://youtu.be/LihUuENsJdI

Let’s see what we got…