Eyes direction recognition

OpenCV

MediaPipe

Team Work

- Mele Alessandro
- Traini Davide

Master's degree students in Computer Engineering at UNIVPM[1].

Repository link[2]

Dataset link[3]

Workflow

- Dataset creation (60%)
- Classification model (30%)
- Regression model (10%)

Workflow

- Dataset creation
- Classification model
- Regression model

Dataset creation

Bot images → Data augmentation → Face Mesh

Dataset creation

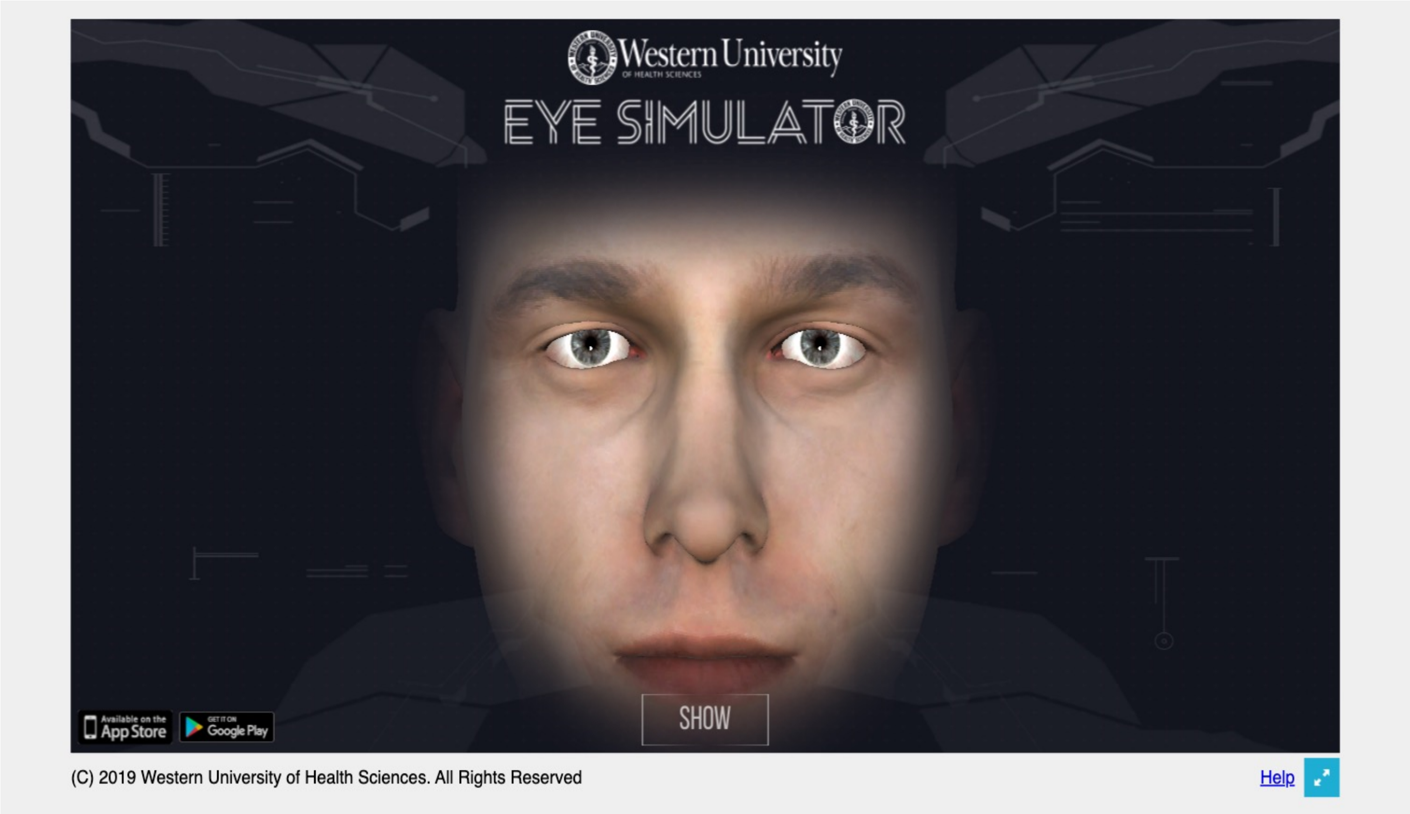
Bot images → Data augmentation → Face Mesh

Dataset creation

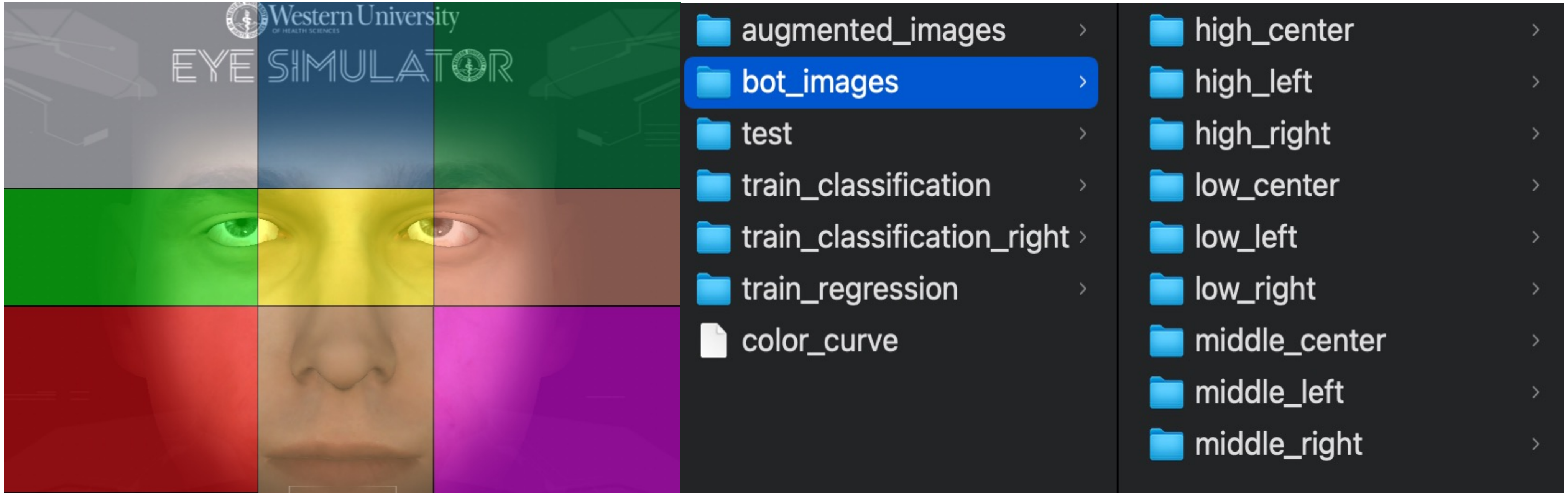
The dataset consists of images derived by an eye simulator of Wester University[4].

We developed a bot in Node.JS[5] that automatically takes screenshots about images moving mouse’s pointer.

Dataset creation



Dataset creation



Dataset creation

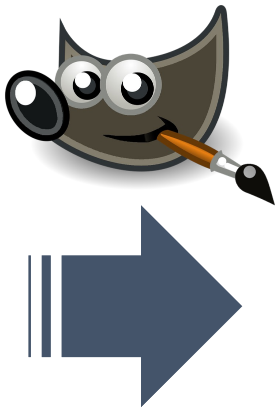
Bot images → Data augmentation → Face Mesh

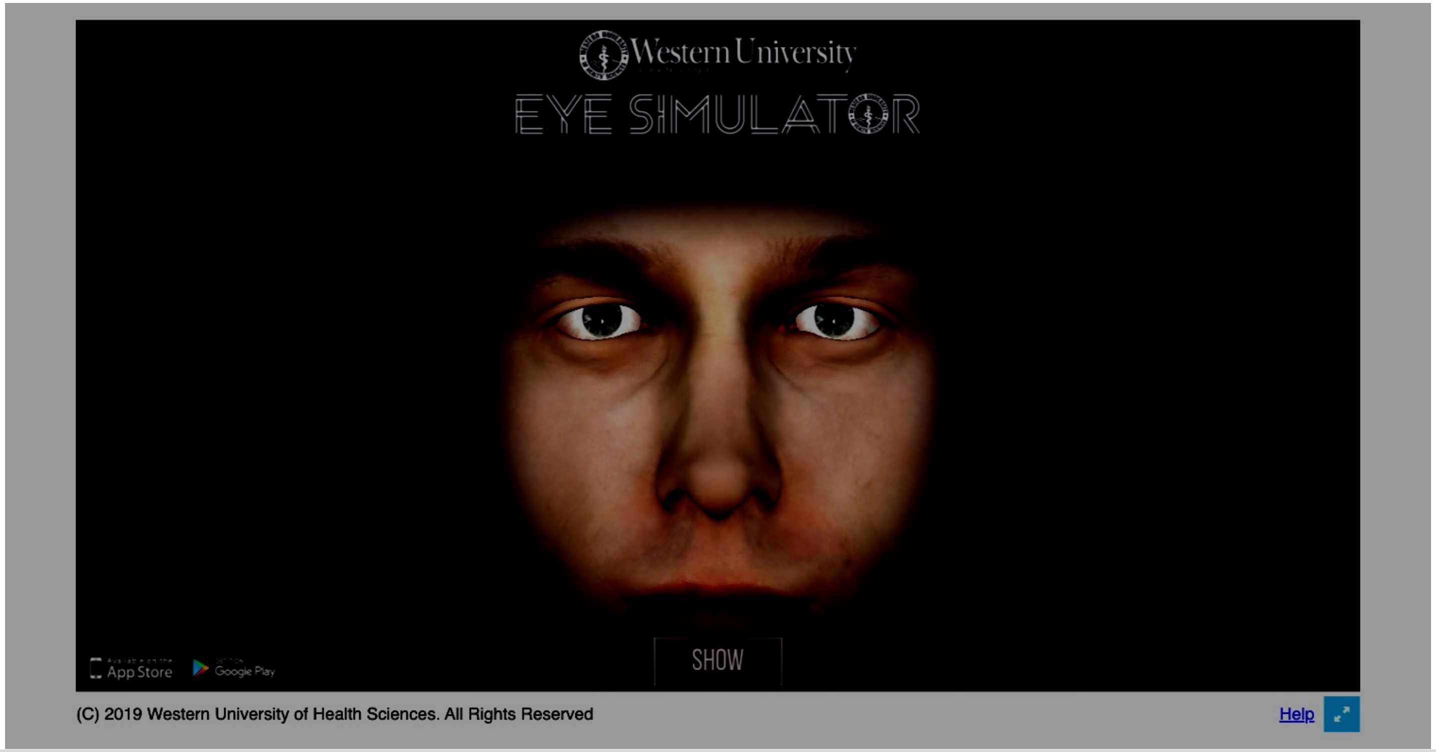
Data augmentation

Starting here

Data augmentation

Modified colour curves, through BIMP[6], a GIMP[7] plugin





Data augmentation

Rotation changes +-5°, through OpenCV[8]

Data augmentation

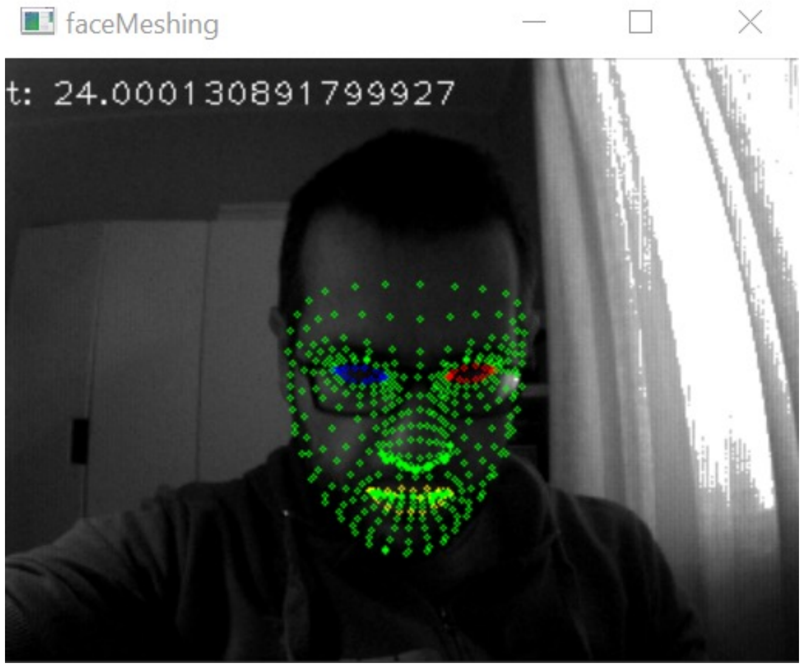
Perspective changes, through OpenCV

Dataset creation

Bot images → Data augmentation → Face Mesh

Eyes extraction with Facemesh

MediaPipe Face Mesh[9] is a Machine Learning solution that estimates 468 3D face landmarks: it uses lightweight model architectures and delivers real-time performance critical for live experiences.

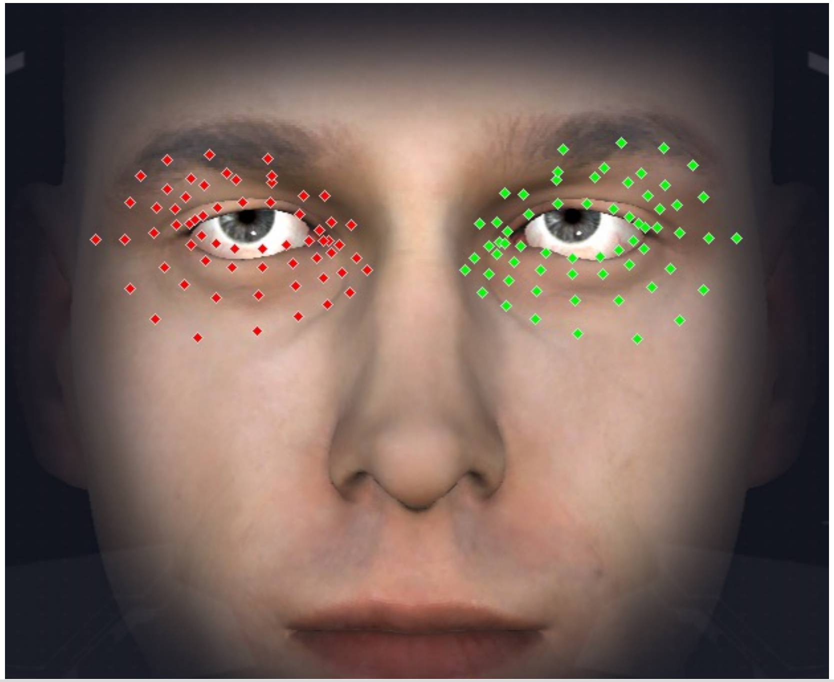


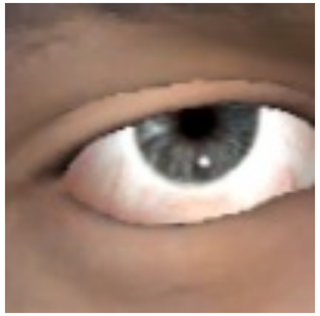
MediaPipe

AD SOLUTIONS

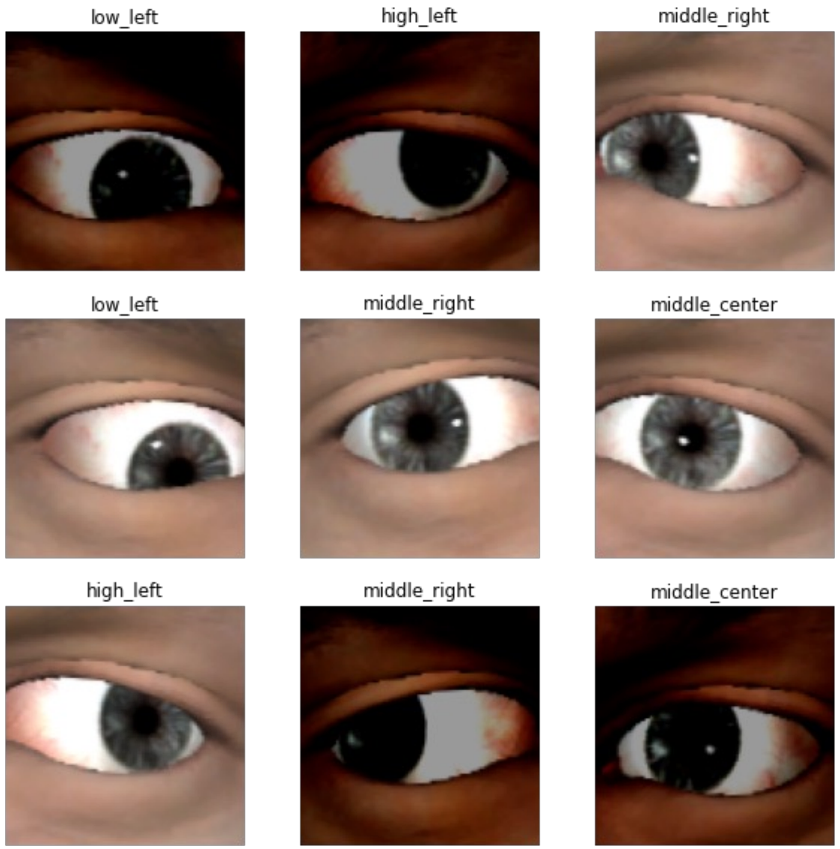
Eyes extraction with Facemesh

Then images have been cropped in two ones which contained right and left eye, and resized at 100x100px.





Dataset creation



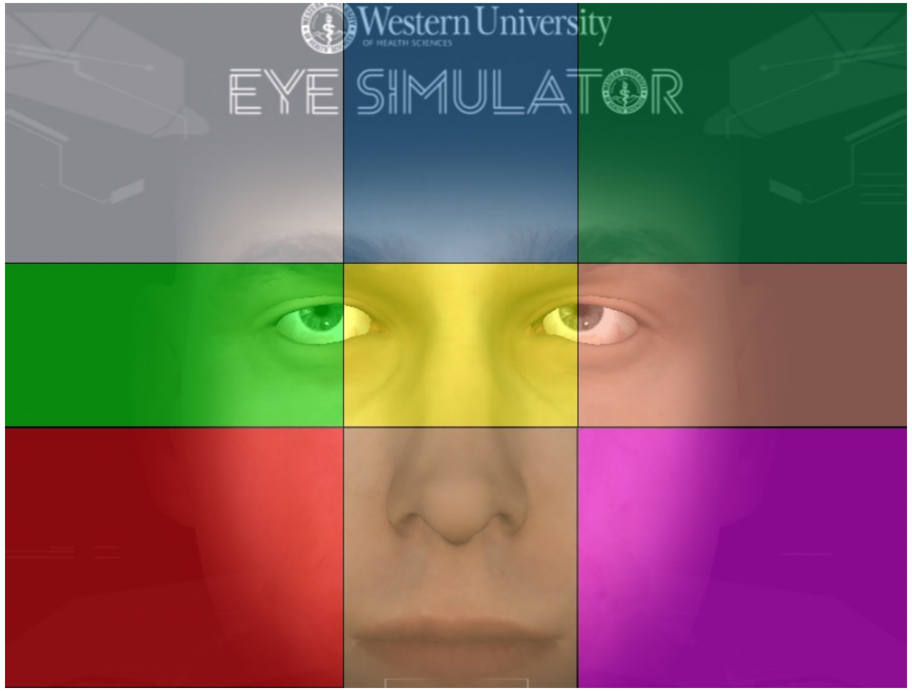
Workflow

- Dataset creation
- Classification model
- Regression model

Classification model

The classifier was trained to recognize nine classes, identified by the quadrants visible in the image.

The chosen network is MobileNet v2.



Classification model

Dataset: 5.376 images divided in:

- 80% Training set;
- 15% Validation set;
- 5% Test set;

Weights: Imagenet[10].

Every image has been rescaled, so pixels values are between 0 and 1.

Classification model: Structure

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100, 100, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 100, 100, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 100, 100, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 4, 4, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 9)                 11529

=================================================================
Total params: 2,269,513
Trainable params: 2,050,313
Non-trainable params: 219,200
```

Classification model: Train

```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
Loss = tf.keras.losses.SparseCategoricalCrossentropy(
       from_logits = False)
Metrics = ['accuracy']
```

```
BATCH_SIZE = 64
IMG_SIZE = (100, 100)
```

```
initial_epochs = 20
```

Classification model: Fine Tuning

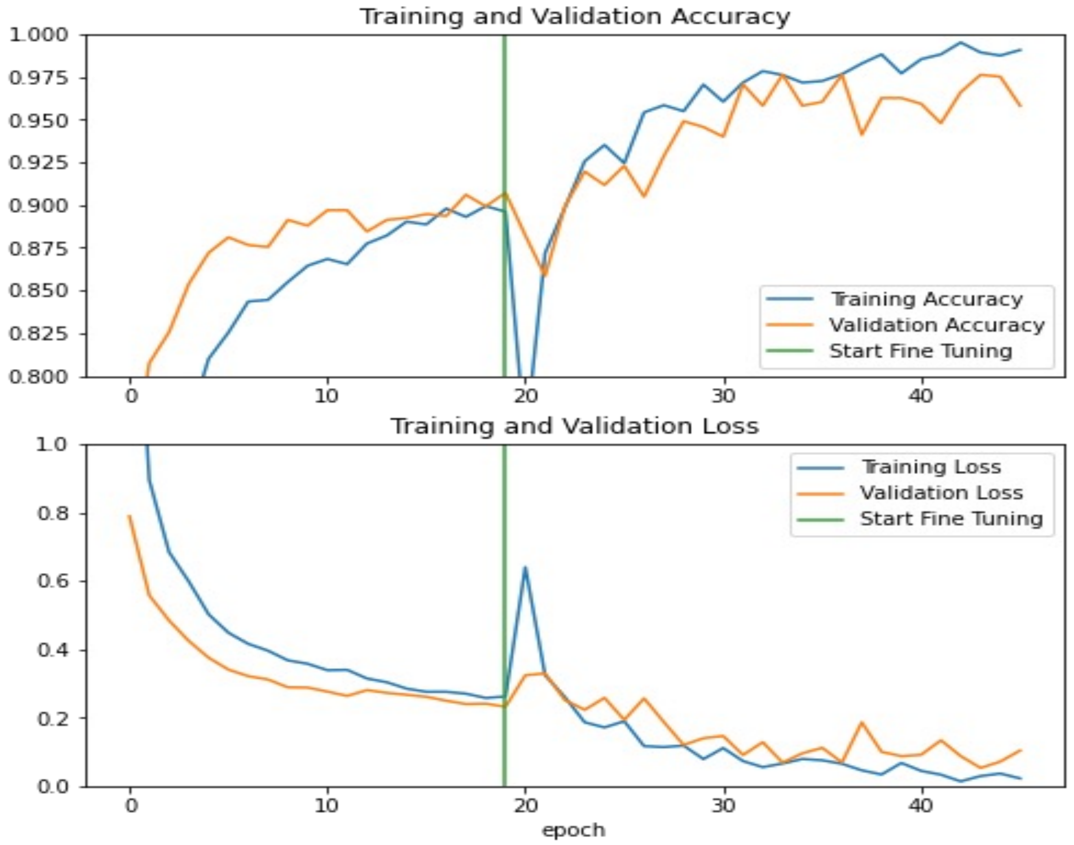
```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
Loss = tf.keras.losses.SparseCategoricalCrossentropy(
       from_logits = False)
Metrics = ['accuracy’]
```

```
BATCH_SIZE = 64
IMG_SIZE = (100, 100)
```

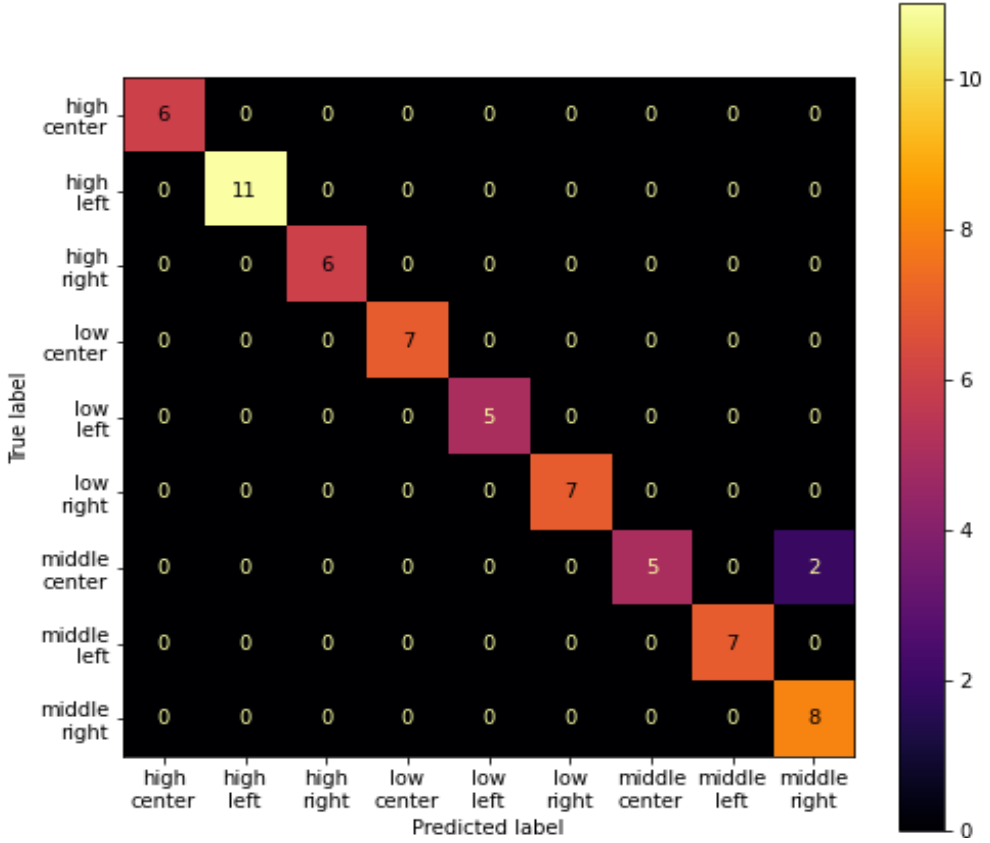
```
fit_epochs = 25
```

```
Fine_tune_at = 80
```

Classification model: Performance



Classification model: Confusion Matrix



Classification model: Real images

Using real images the model was not able to generalize,
so we tried to resolve with three methods:

- Train one model for left and another for right eye;
- Train the model with only middle_left and middle_right classes;
- Apply a cost matrix, to avoid unbalanced class problems.

The above listed methods were not sufficiently able to mitigate the generalization issue. A more complete data-set with synthetic and real images is a good candidate for that.

Workflow

- Dataset creation
- Classification model
- Regression model

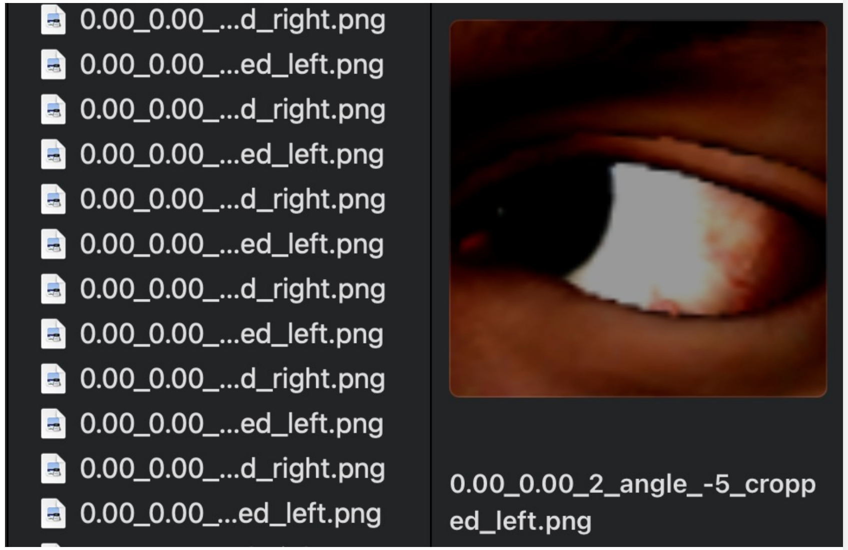
Regression model: Dataset

Same as classification, but without folders (classes).

Images has been saved as ***horPerc_verPerc***

where ***Perc(s)*** are normalized than maximum

and minimum (x,y) cursor movement.



Regression model: Structure

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100, 100, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 100, 100, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 100, 100, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 4, 4, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,041,346
Non-trainable params: 219,200
```

Regression model: Train

```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
loss = 'mse'
Metrics = ['mse','mae','mape']
```

```
BATCH_SIZE = 64
IMG_SIZE = (100, 100)
```

```
initial_epochs = 20
```

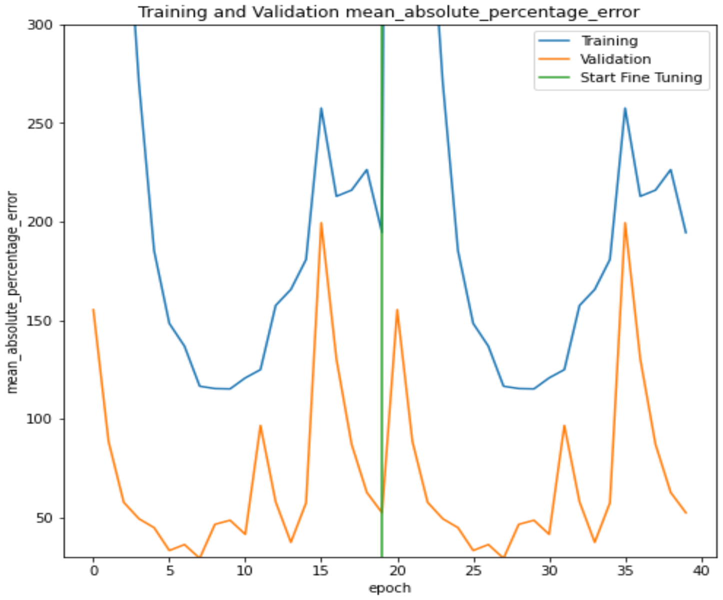
Regression model: Fine Tuning

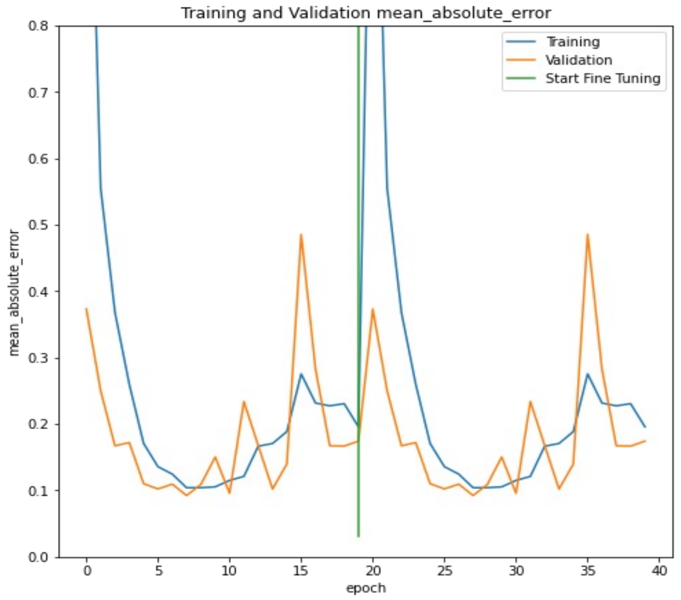
```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
loss = 'mse'
Metrics = ['mse','mae','mape']
```

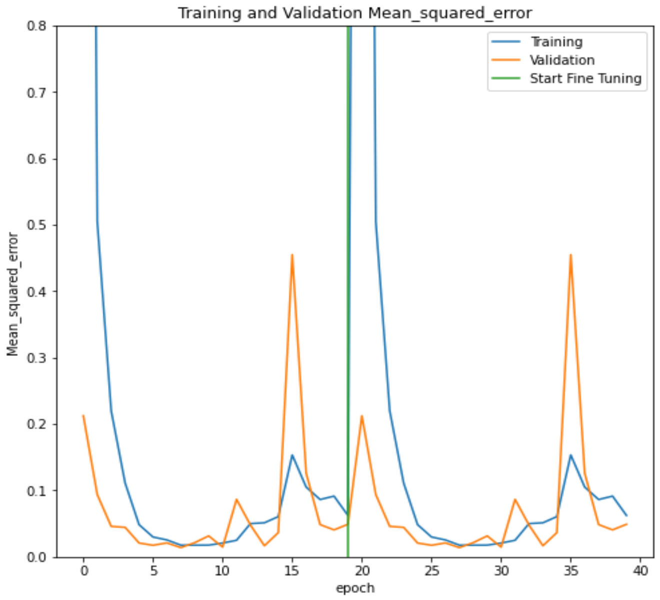
```
BATCH_SIZE = 64
IMG_SIZE = (100, 100)
```

```
fit_epochs = 25
Fine_tune_at = 80
```

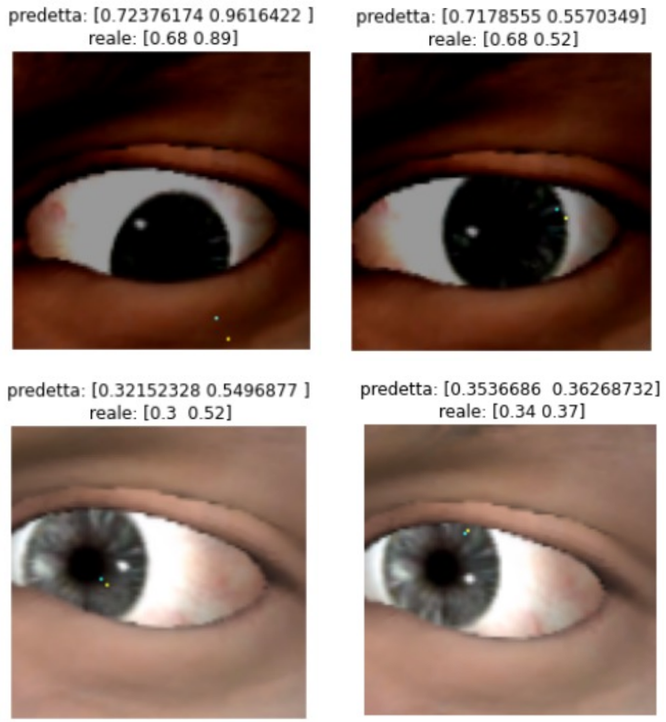
Regression model: Performance







Regression model: Apply model



AD SOLUTIONS

Conclusions

Both classification and regression models are really accurate on synthetic dataset, but have poor performances on real world data.

A further approach would be to train the network on both synthetic and real data.

References

UNIVPM: https://www.univpm.it/Entra/

GitHub Repository: https://github.com/AlessandroMele/eyes_direction_recognition

Drive folder for dataset: https://drive.google.com/drive/folders/1CYh07UE_v58td-wKqa1jO1x-bAOt65gS?usp=sharing

Western University eye simulator: https://edtech.westernu.edu/3D-eye-movement-simulator/

Node.JS: https://nodejs.org/it/

BIMP: https://alessandrofrancesconi.it/projects/bimp/

GIMP: https://www.gimp.org

OpenCV: https://opencv.org

Face Mesh: https://google.github.io/mediapipe/solutions/face_mesh.html

ImageNet: https://www.image-net.org/