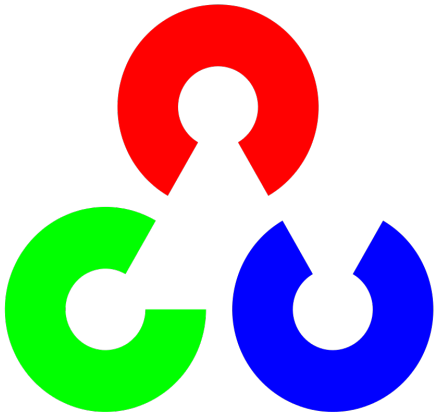
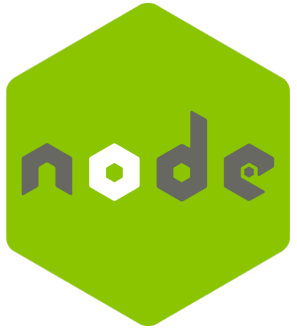




Deep Eyes Recognition

Team Work

- Mele Alessandro
- Traini Davide

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

Repository link[2] 

Dataset link[3] 

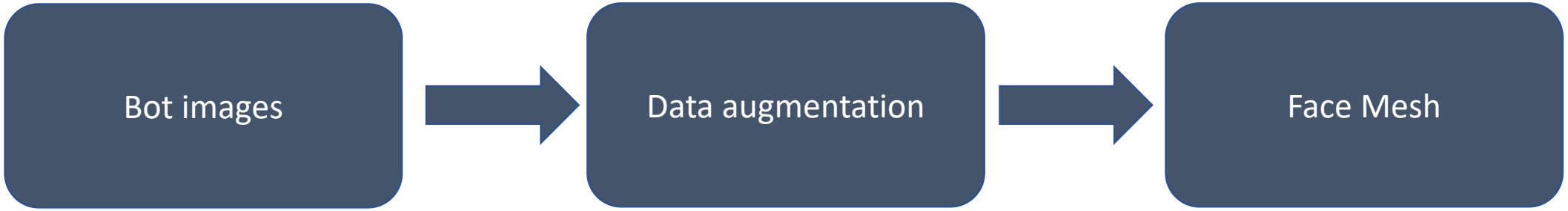
Workflow

- Dataset creation
- Classification model
- Regression model

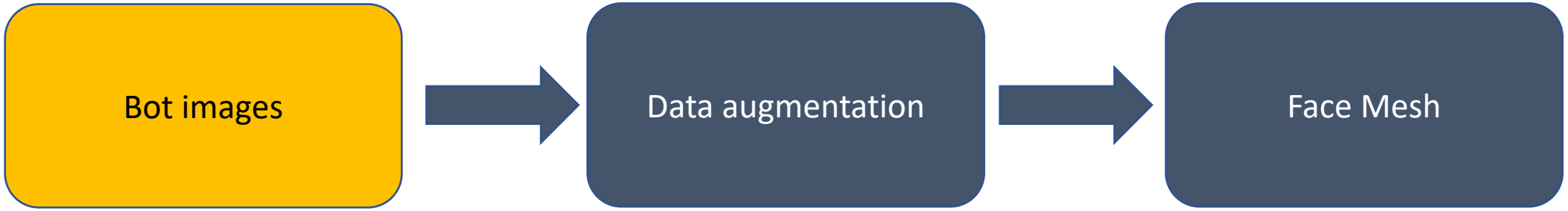
Workflow

- Dataset creation
- Classification model
- Regression model

Dataset creation



Dataset creation



Dataset creation

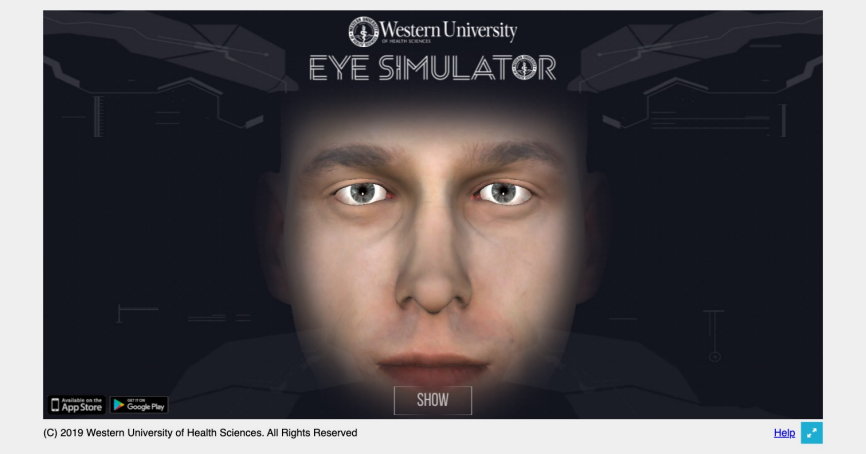
The dataset consists of images got by an eye simulator of Wester University[4].

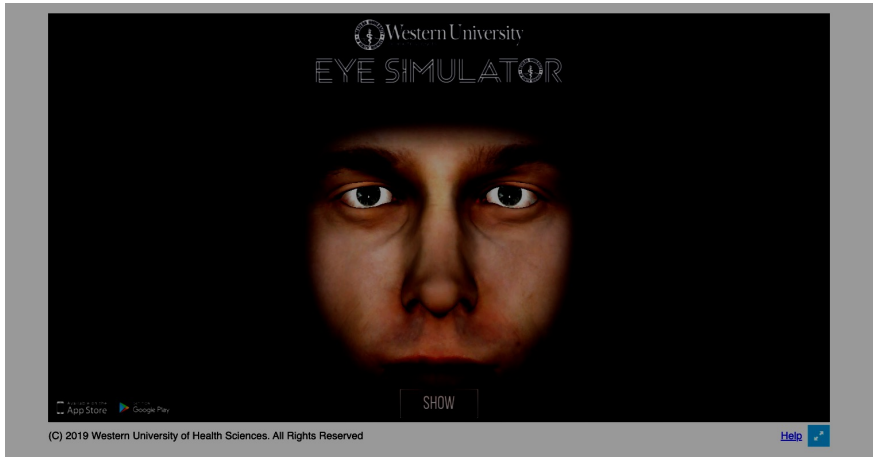
We realised it by making a bot in Node.JS that automatically takes screenshots about images moving mouse's pointer.

Dataset creation

Data augmentation

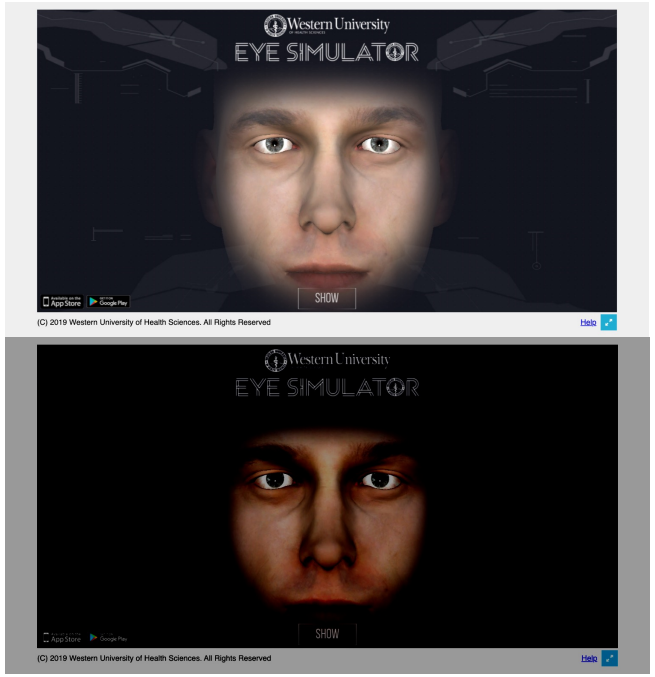
Modified colour curves

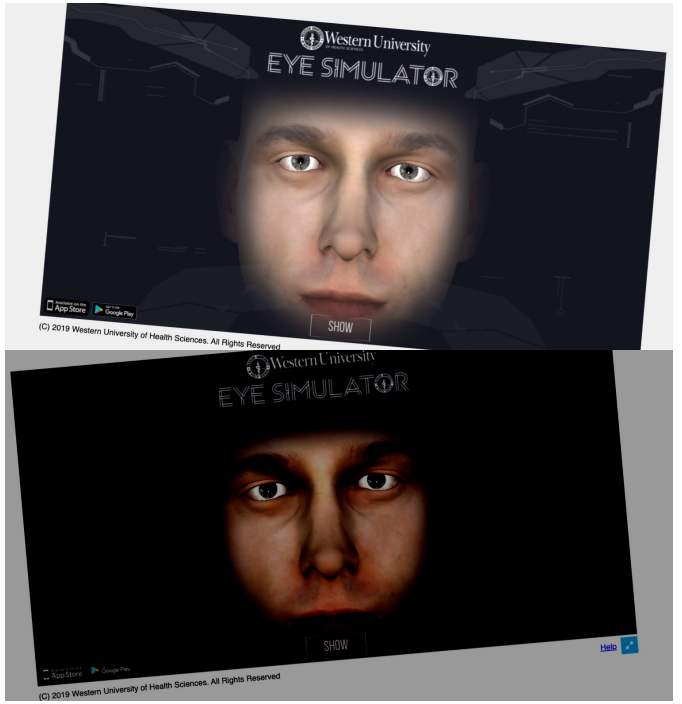




Data augmentation

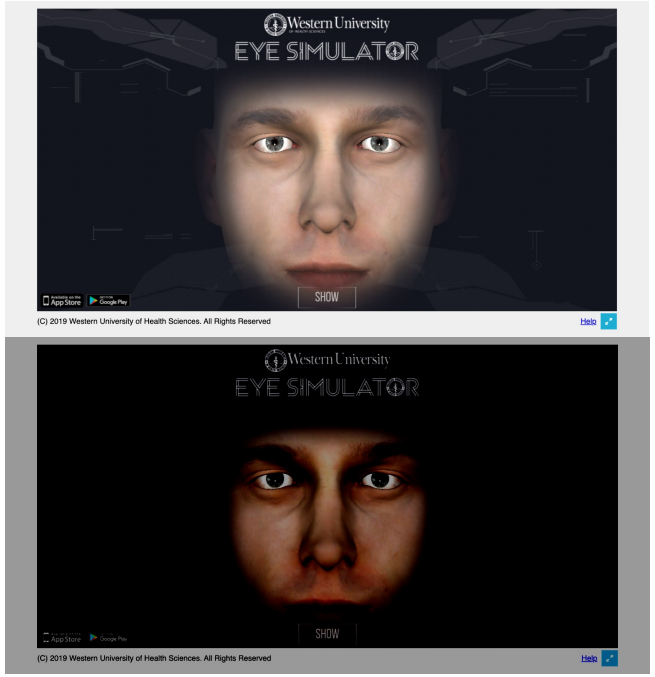
Rotation changes (+-5°)

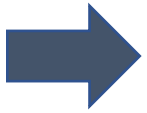


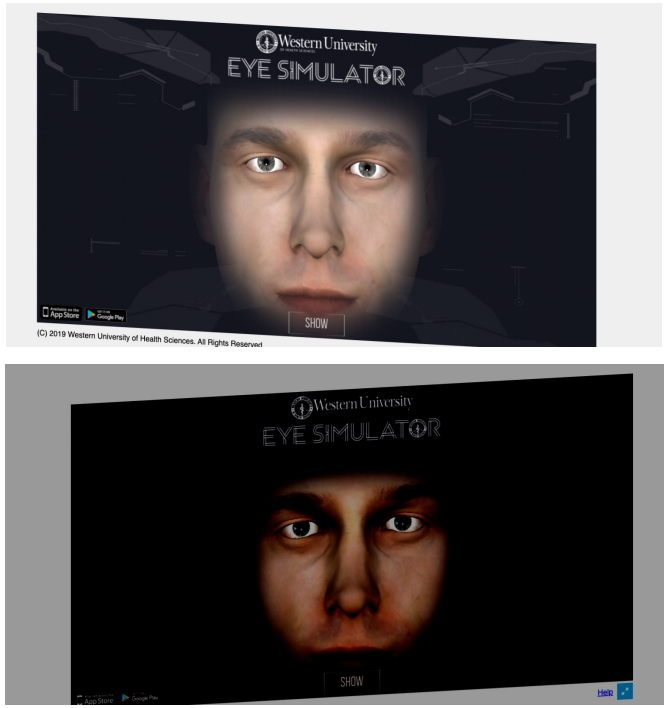


Data augmentation

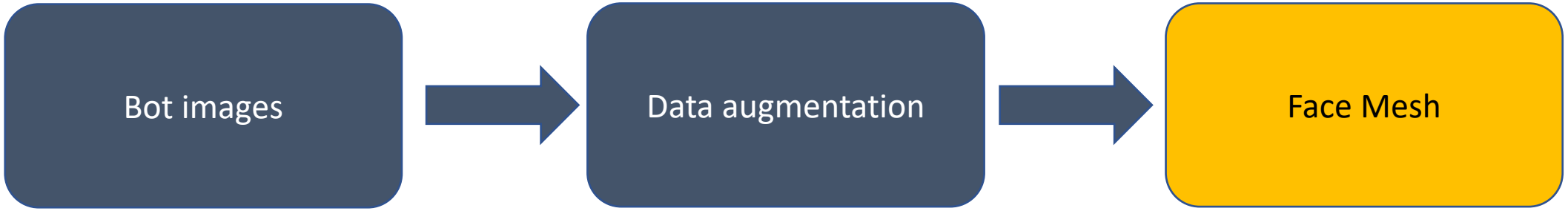
Perspective changes





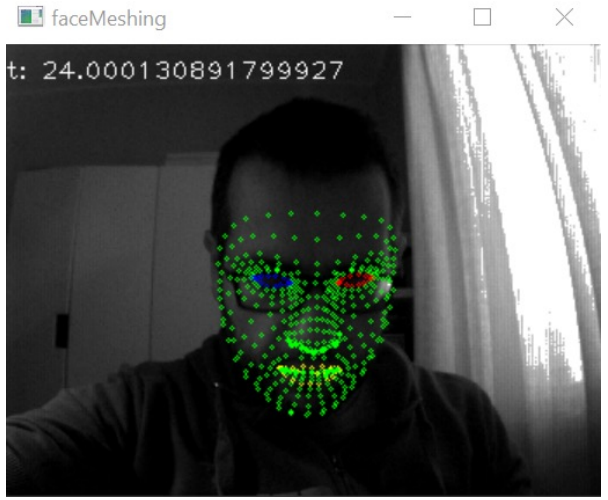


Dataset creation



Eyes extraction with Facemesh

MediaPipe Face Mesh[5] 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.

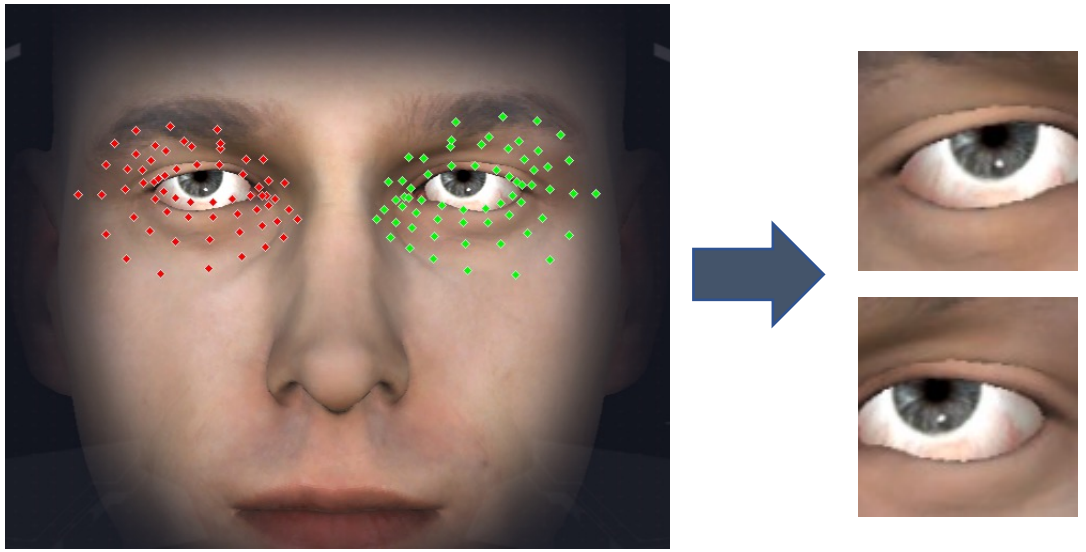


MediaPipe

Eyes extraction with Facemesh

Focus has been on eyes' keypoints, calculating bounding boxes.

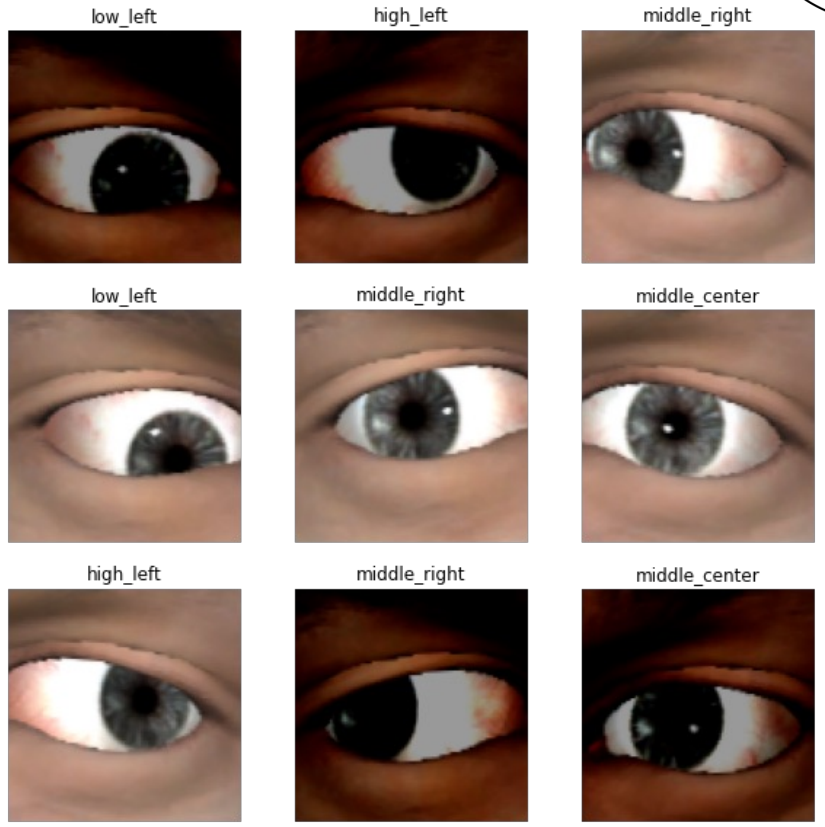
Then images have been cropped in two ones which contained right and left eye.



MediaPipe

Dataset creation

Then, images have been cropped at 100x100px. The final result is



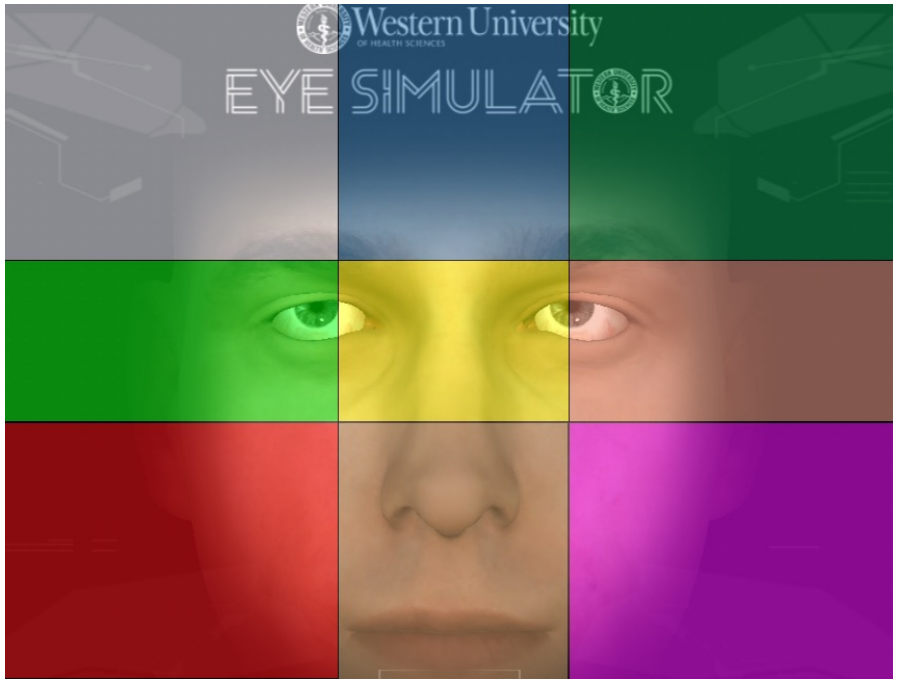
Workflow

- 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

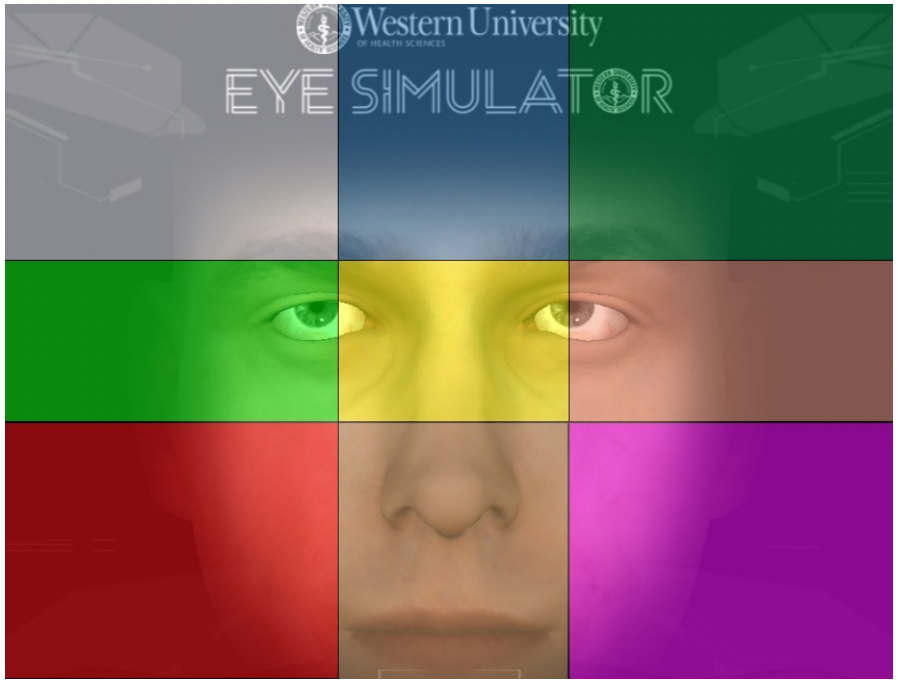
Dataset: 5376 images divided in:

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

with shuffle selection;

Weights: Imagenet[6].

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 (TFOpLambda)	(None, 100, 100, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 4, 4, 1280)	2257984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
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

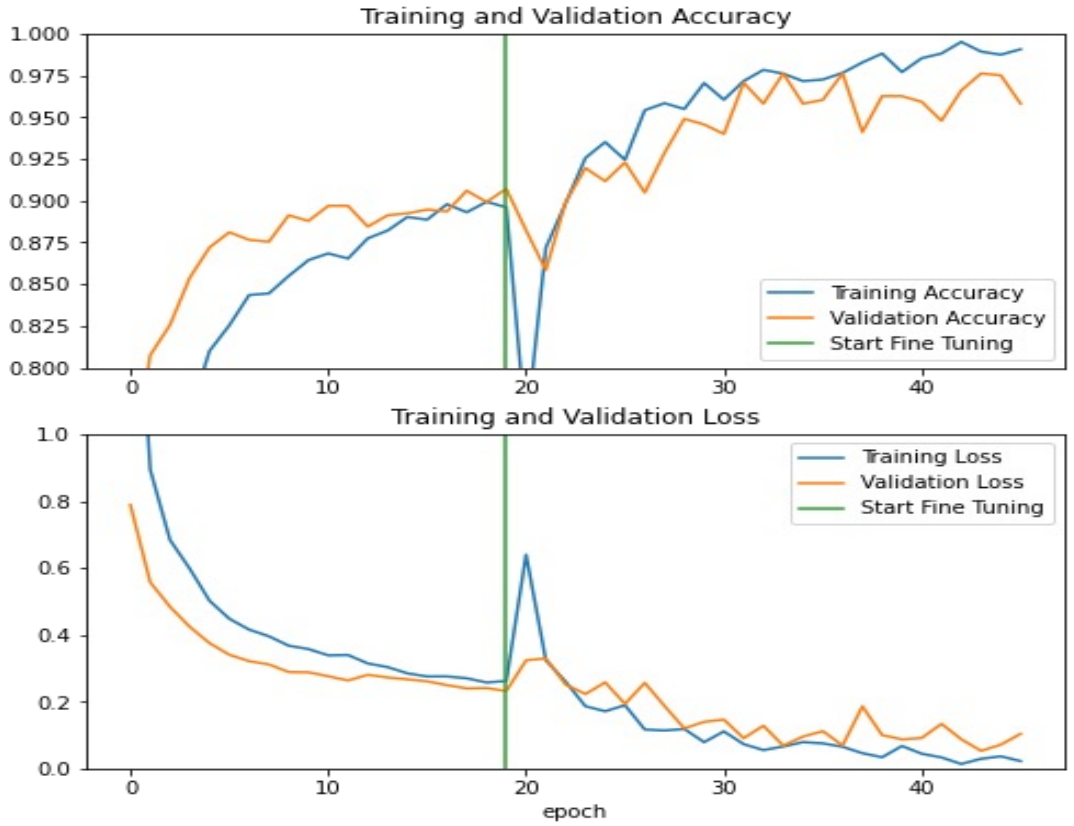

```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
Loss = tf.keras.losses.SparseCategoricalCrossentropy(
       from_logits = False)
Metrics = ['accuracy']
```

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

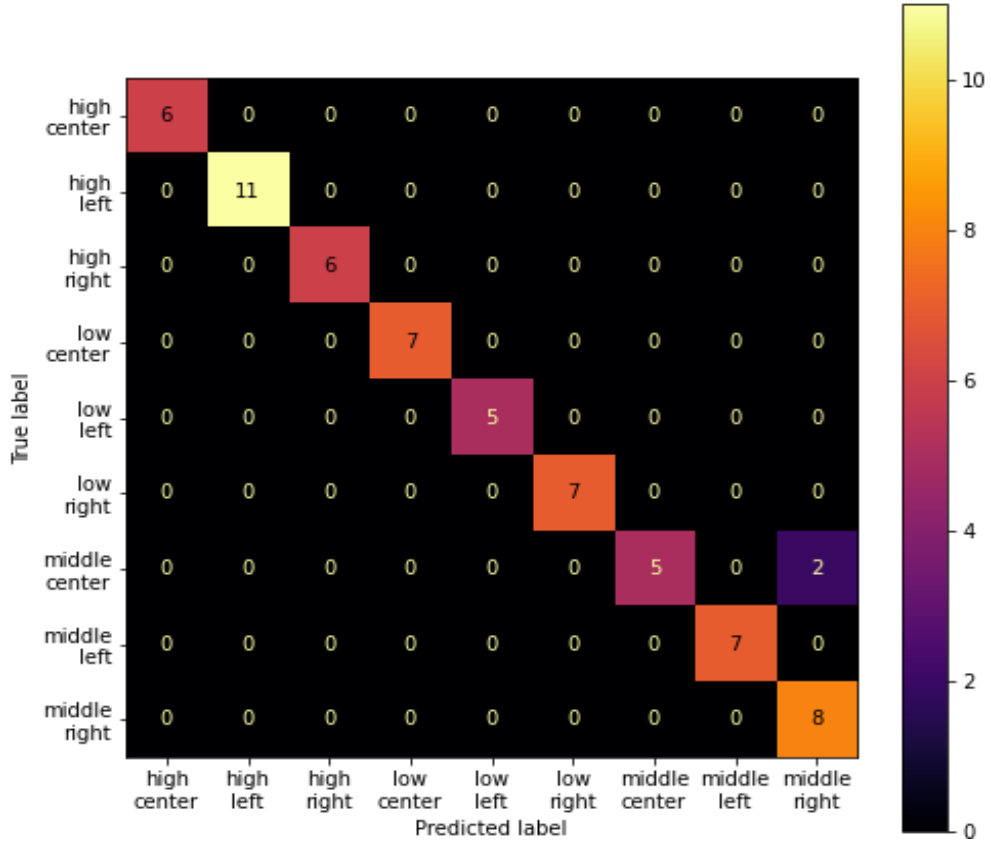
```
fit_epochs = 25
```

```
Fine_tune_at = 80
```

Classification model: Performance



Classification model: Confusion Matrix



Classification model: Real images

In real images the model has bad accuracy, 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.

None of the three methods improved performance. ☹

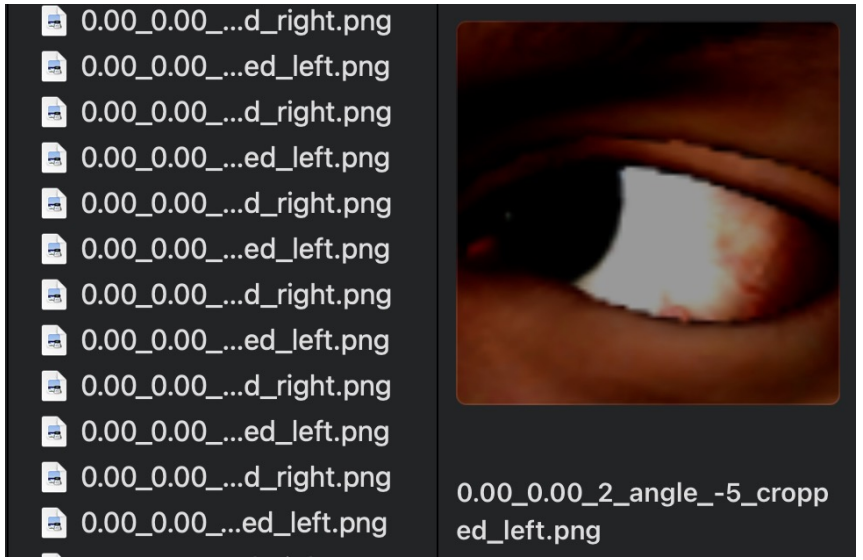
Workflow

- Dataset creation
- Classification model
- Regression model

Regression model: Dataset

Same as classification, but without folders (classes).

Images has been saved with *hor_perc_ver_perc*.png 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 (TFOpLambda)	(None, 100, 100, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 4, 4, 1280)	2257984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
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

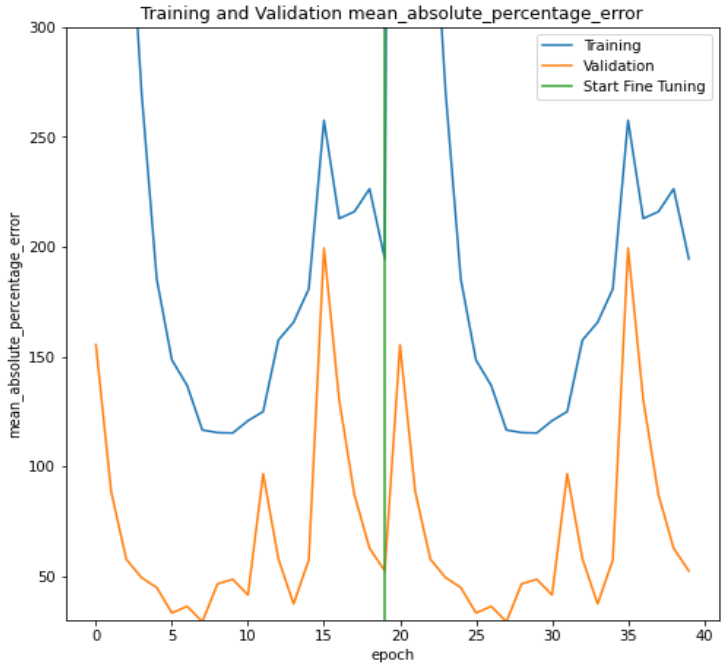

```
Dropout = 0.3
Optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
loss = 'mse'
Metrics = ['mse','mae','mape']
```

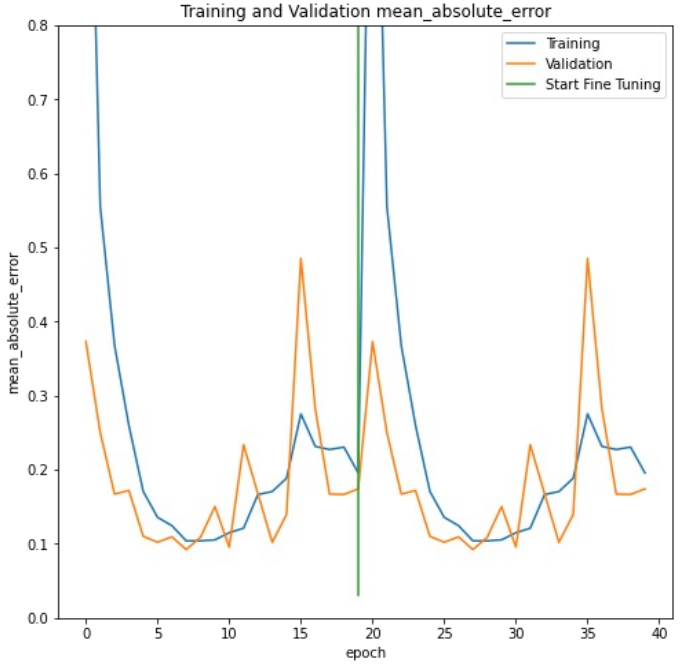
```
BATCH_SIZE = 64
IMG_SIZE = (100, 100)
```

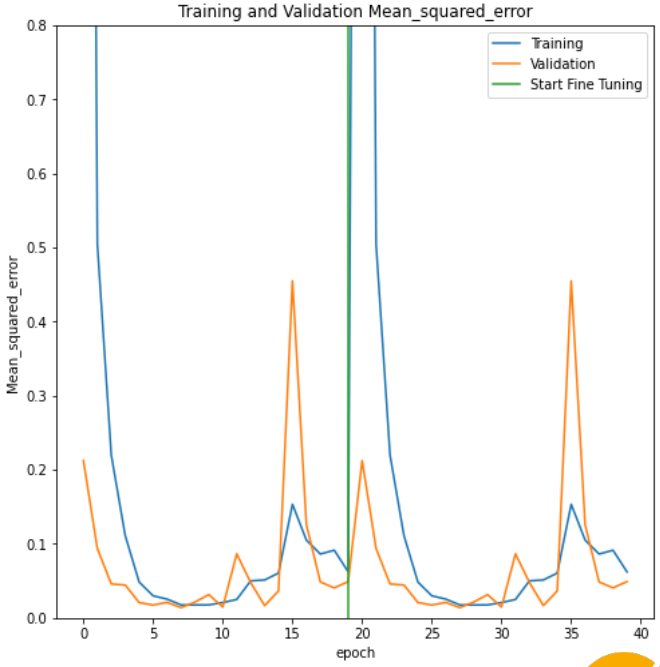
```
fit_epochs = 25
```

```
Fine_tune_at = 80
```

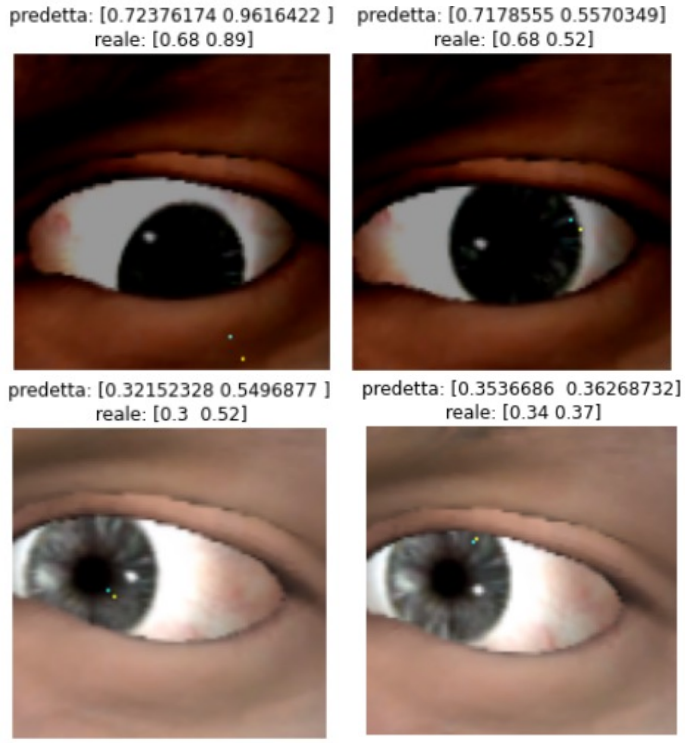
Regression model: Performance

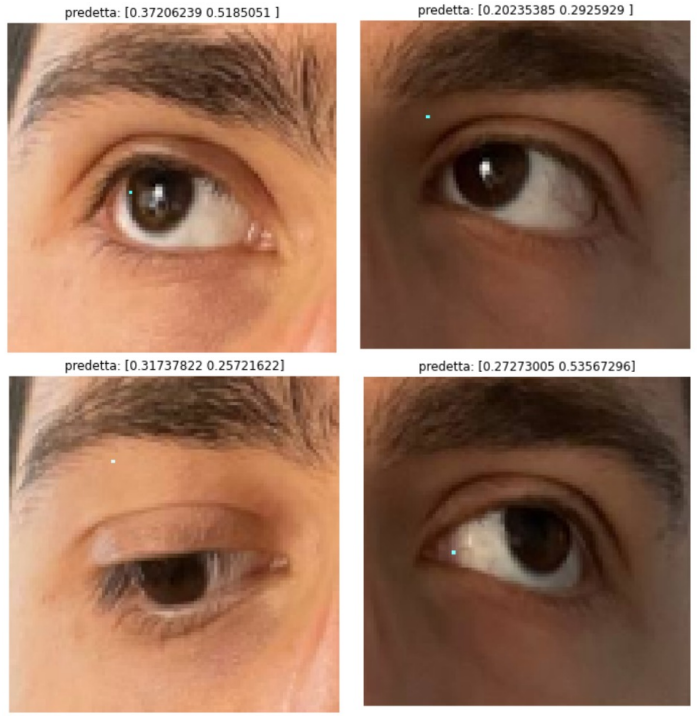






Regression model: Apply model





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.