EXPOSITION DISPLAY GAZE TRACKING SYSTEM

ROSS MONAGHAN – G00376556

INSTITUTION: GALWAY MAYO INSTITUTE OF TECHNOLOGY
COURSE: BENG(H) IN SOFTWARE AND ELECTRONIC ENGINEERING - YEAR 4
MODULE: PROJECT ENGINEERING
SUPERVISOR: NIALL O'KEEFFE
YEAR: 2020/2021

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Signed: Ross Monaghan

Acknowledgements

I would like to express my gratitude and appreciation to the academic staff at GMIT who have provided continuous support and guidance to me throughout the duration of the academic year.

Specifically, my thanks go out to my project supervisor Niall O'Keeffe, who has been extremely helpful in providing technical feedback and direction to areas of my project which required support.

I would also like to thank my Project Engineering lecturers, Brian O'Shea, Michelle Lynch and Paul Lennon, who have been present and available to provide help and guidance throughout the year.

Table of Contents

1 Summary

The goal of the Exposition Display Gaze Tracking System was to develop a computer vision system that estimates the gaze of attendees at an exposition, so that data may be gathered of what component of a display most frequently caught a person's eye. The result of this would be a more detailed understanding of how a display performs, so that its design may be altered to grab attention more effectively.

This projects scope focused on the computer vision aspect of the system, with the main deliverable being the development of a gaze following program, which can detect the most likely area in a frame that an individual is looking.

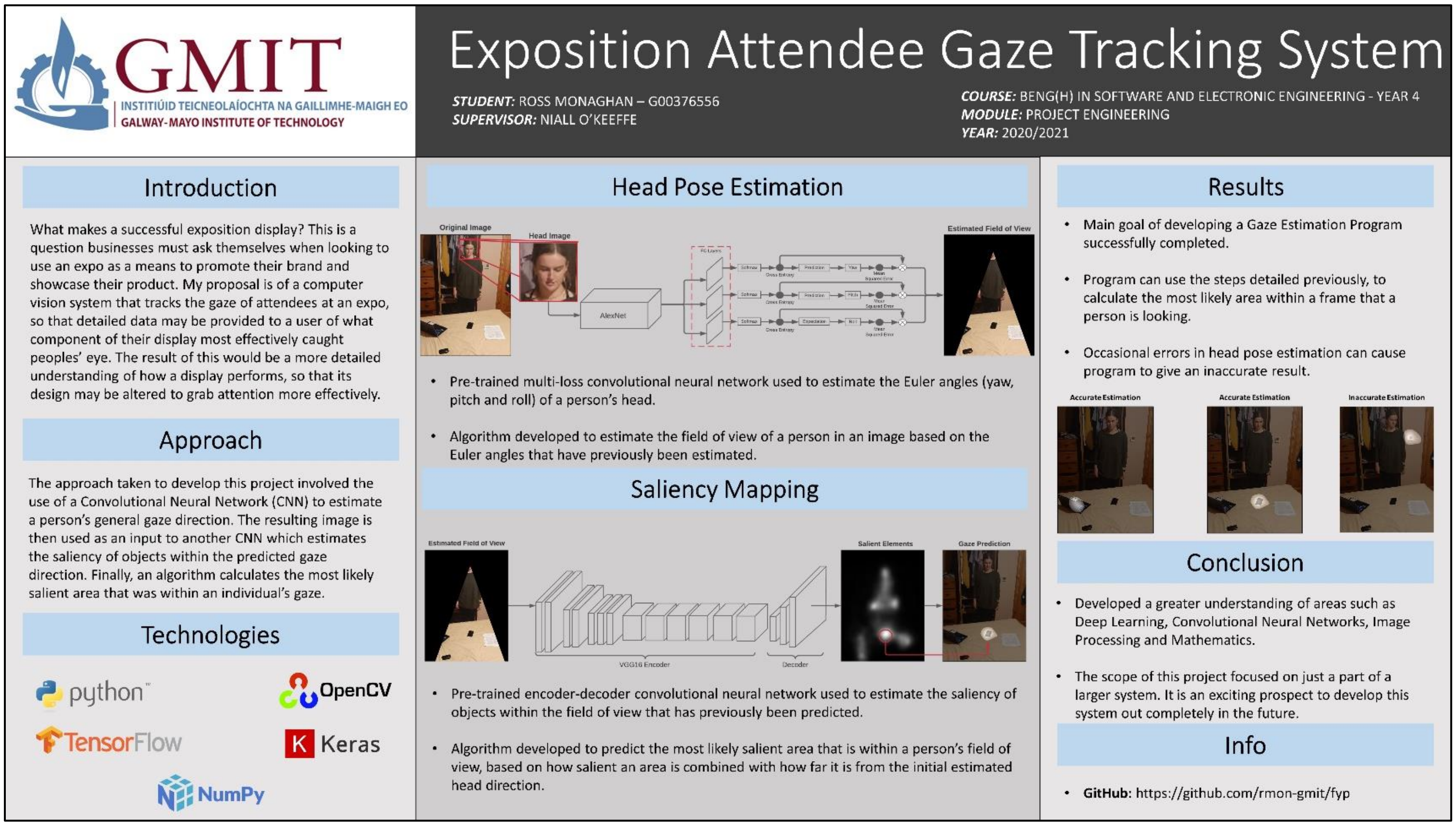
The approach taken to implement this functionality involved the use of a Convolutional Neural Network (CNN) to estimate a person's general gaze direction. The resulting image then being used as an input to another CNN which estimates the saliency of objects within said gaze direction. An algorithm would then calculate the most likely salient area that was within an individual's gaze.

The Project has been developed using Python. It has involved the use of TensorFlow/Keras libraries for its Deep Learning and CNN areas and has used the NumPy library for the mathematical aspects of the code. OpenCV has been used for the various image processing tasks that the project requires.

Much was accomplished during the duration of The Project, with the main goal of developing a Gaze Estimation Program successfully completed. This program can use the steps detailed previously, to calculate the most likely area within a frame that a person is looking.

Conclusions made during the development of this project have led to a greater understanding of areas such as Deep Learning, Convolutional Neural Networks, Image Processing and Mathematics. The resulting program has much opportunity for further development, and this will undoubtably be pursued in the future. As was stated previously, the scope of this project focused on just a part of a larger system, and it is an exciting prospect to develop this system out completely in the future.

2 Poster


GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY
Original Image
Head Image
AlexNet
Estimated Field of View
Salient Elements
Gaze Prediction
VGG16 Encoder
Decoder
Accurate Estimation
Accurate Estimation
Inaccurate Estimation
python
OpenCV
TensorFlow
Keras
NumPy

Exposition Attendee Gaze Tracking System

STUDENT: ROSS MONAGHAN – G00376556
SUPERVISOR: NIALL O'KEEFFE

COURSE: BENG(H) IN SOFTWARE AND ELECTRONIC ENGINEERING - YEAR 4
MODULE: PROJECT ENGINEERING
YEAR: 2020/2021

Introduction

What makes a successful exposition display? This is a question businesses must ask themselves when looking to use an expo as a means to promote their brand and showcase their product. My proposal is of a computer vision system that tracks the gaze of attendees at an expo, so that detailed data may be provided to a user of what component of their display most effectively caught peoples' eye. The result of this would be a more detailed understanding of how a display performs, so that its design may be altered to grab attention more effectively.

Approach

The approach taken to develop this project involved the use of a Convolutional Neural Network (CNN) to estimate a person's general gaze direction. The resulting image is then used as an input to another CNN which estimates the saliency of objects within the predicted gaze direction. Finally, an algorithm calculates the most likely salient area that was within an individual's gaze.

Technologies

Head Pose Estimation

- Pre-trained multi-loss convolutional neural network used to estimate the Euler angles (yaw, pitch and roll) of a person's head.
- Algorithm developed to estimate the field of view of a person in an image based on the Euler angles that have previously been estimated.

Saliency Mapping

- Pre-trained encoder-decoder convolutional neural network used to estimate the saliency of objects within the field of view that has previously been predicted.
- Algorithm developed to predict the most likely salient area that is within a person's field of view, based on how salient an area is combined with how far it is from the initial estimated head direction.

Results

- Main goal of developing a Gaze Estimation Program successfully completed.
- Program can use the steps detailed previously, to calculate the most likely area within a frame that a person is looking.
- Occasional errors in head pose estimation can cause program to give an inaccurate result.

Conclusion

- Developed a greater understanding of areas such as Deep Learning, Convolutional Neural Networks, Image Processing and Mathematics.
- The scope of this project focused on just a part of a larger system. It is an exciting prospect to develop this system out completely in the future.

Info

- **GitHub:** https://github.com/rmon-gmit/fyp

Figure 1: Project Poster

3 Introduction

The motivation for pursuing this project stems largely from the increasing use of Deep Learning algorithms to improve processes or tasks which are benefited by the access to more sources of information. Computer Vision is a field which has on many occasions implemented Deep Learning in this way, providing a user or system with useful and accurate information which has been gathered from an image or images.

Gaze following is an established computer vision topic which is used to estimate the most likely area that an individual in a frame is looking. Different from eye tracking, which relies on a clear view of a person's eye to calculate the point which they are looking. Gaze following instead relies on deducing the direction a person is looking based on an image of their head, followed by a saliency prediction on the objects in that direction [1]. What results will not be an exact point in an image, but instead a general area.

The goal of this project is to develop a gaze following program which will be used to estimate the general area in an image that a person is looking. In the context of a larger Exposition Display Gaze Tracking System, this program would be fundamental to that systems functionality.

The work required to develop the gaze following program will involve extensive research into both deep learning and convolutional neural networks. The knowledge gained from this research will be used to develop a head pose estimation deep learning algorithm which will estimate gaze direction based on a person's head. This estimation will be used to generate a field of view for that person, which can then be subject to a saliency prediction using another deep learning algorithm. The result of this saliency prediction will then be overlayed on the original image, creating a heat map of the area that a person is looking.

This report will detail the underlying technologies and concepts involved in building a gaze following program, followed by the steps taking in completing the project. What will follow is a description of the results, and a comparison with existing systems.

4 Convolutional Neural Networks

When dealing with computer vision applications of deep learning, Convolutional Neural Networks (CNN's) are almost universal in their usage. The robustness in which CNN's learn and recognise patterns, combined with their ability to break larger data down into smaller chunks, make them excellent for deep learning tasks that involve images [2].

4.1 Convolution Operation

The core operation of a Convolutional Neural Network is that of the convolution. In essence, a convolution involves taking two matrices and performing an element-by-element multiplication on them, the resulting values being summed together [3].

When applied to images, convolutions involve the creation of a 'kernel', which is to say a matrix of fixed value and of a size that is usually much smaller than that of the original image. This kernel is then applied to a section of the original image, creating an output for that section. When this process is repeated throughout the image, an output image can be created which will vary greatly depending on the values which have been assigned to the kernel. A diagram of a convolution is contained below:

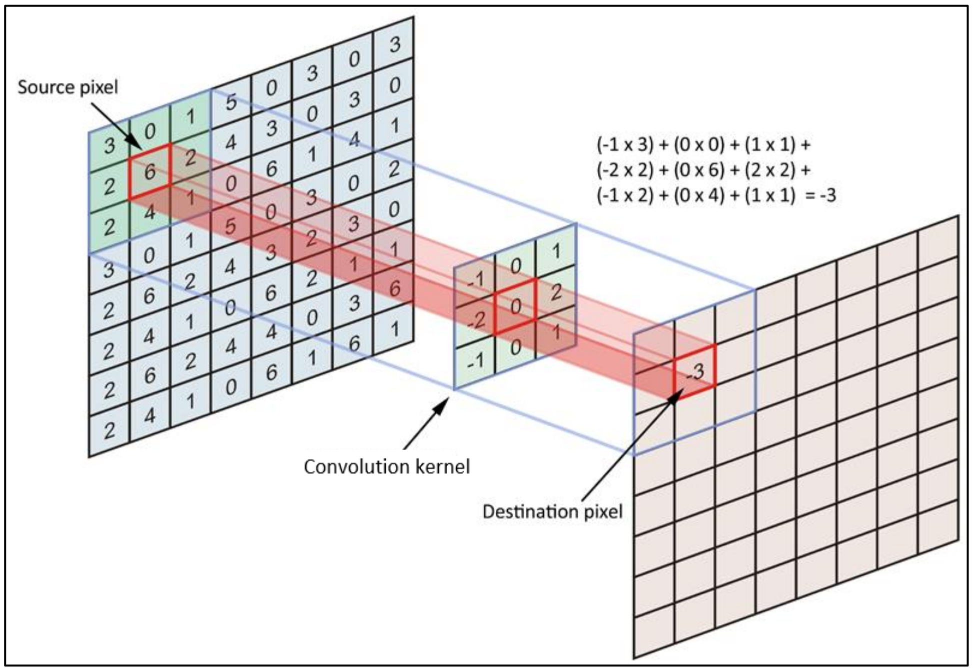

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution kernel
Destination pixel

Figure 2: Convolution Diagram [4]

4.2 Convolutions for Pattern Recognition

The fact that different kernel values can have drastically different effects on the resulting image is extremely powerful and results in the ability to use kernels of specifically selected values to carry out operations such as filtering or edge/contour detection.

Take for example the kernel present in the previous figure, which is used as part of the common Sobel operator used in edge detection and will have the effect of emphasising vertical lines in an image. The entire Sobel operator will output a result of both vertical and horizontal kernels applied to an image and can be compared with just the vertical kernel in the figures below:

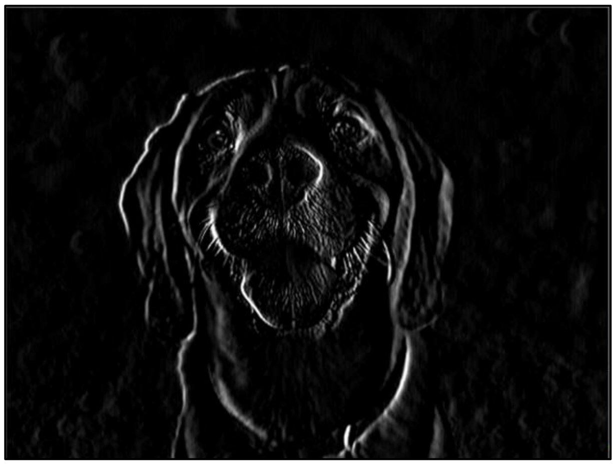


Figure 3: Emphasized Vertical Lines

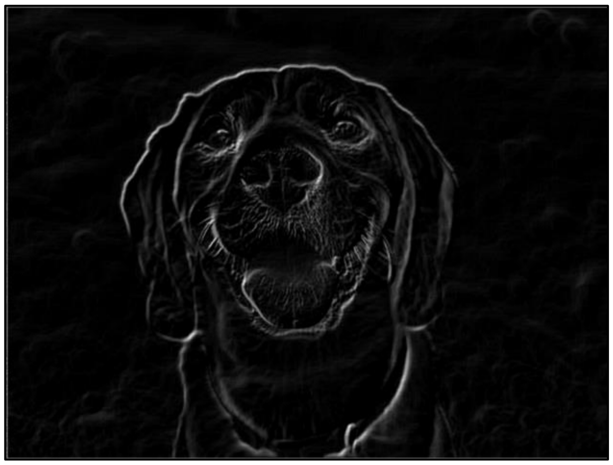


Figure 4: Full Sobel Operator

In the context of CNN's, the ability to extract very basic details (such as edges and contours) from an image is essential to the effectiveness of the network. This is because the underlying functionality of a CNN is that these very basic features may be combined to create more complex patterns which the network can recognise and use to form its prediction [5].

4.3 Neural Networks

Drawing inspiration from neuroscience and biology, neural networks are a collection of artificial neurons that 'fire' or activate depending on what is input to the network. Similarly to how different stimuli will fire different collections of neurons in an organic brain, the data that is input to a neural network will activate different collections of artificial neurons. This property of neural networks can be used to make predictions on the data that has been input. Though the uses of neural networks vary widely, when applied to computer vision tasks neural networks are used because of their inherent ability to recognise patterns [2].

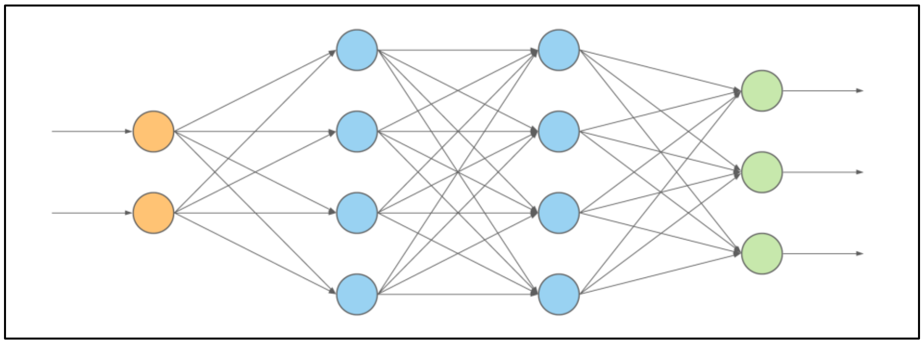


Figure 5: Simple Neural Network

4.3.1 Artificial Neuron

The most basic component of a neural network is that of the artificial neuron. Artificial neurons are mathematical functions whose inputs are obtained from the outputs of previous neurons or the initially input data. Their outputs are in turn, linked to other artificial neurons further on in the network. Artificial neurons are linked together in layers to create the underlying structure of a neural network.

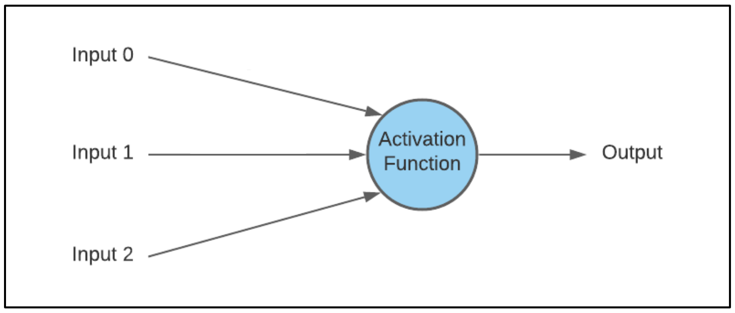


Figure 6: Artificial Neuron Model

4.3.2 Activation Functions

The determining factor to whether a neuron is activated or not is decided by that neuron's activation function, that is a mathematical function that normalises the input data to some value range [5]. It is the result of this activation function that is passed to subsequent neurons as their input(s).

The chosen activation function can have quite a large difference to the value that is output from a neuron. Take for example the output of the commonly used Rectified Linear Unit (ReLU) function when compared to the nonlinear sigmoid function:

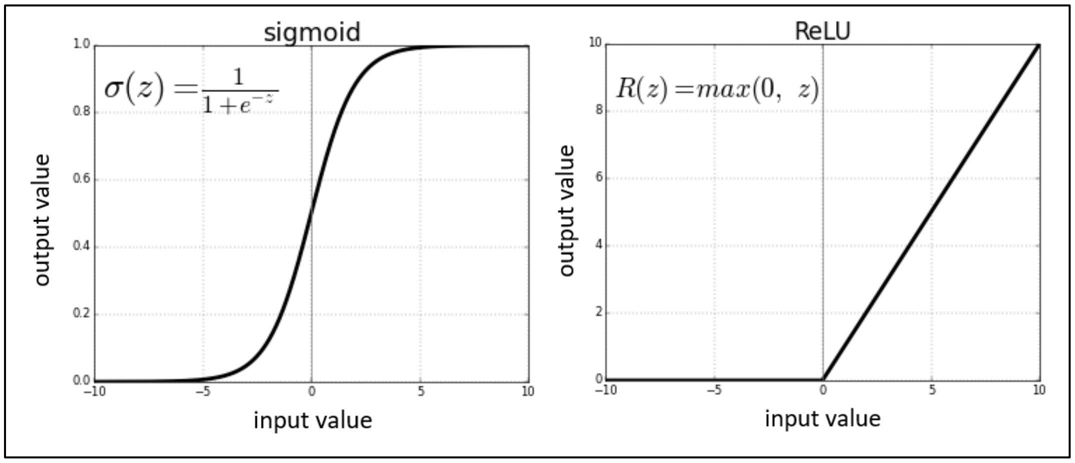


Figure 7: Activation Function Comparison

Comparing the above diagrams, it is possible to see the difference in what will be output from each activation function. Where the sigmoid activation will normalize the input value to always be a number between 0 and 1, the ReLU activation will normalize the input value to be between 0 and the input value. Additionally, the ReLU activation will assign all negative inputs to a value of zero, whereas the sigmoid activation will accommodate some negative input values by assigning them a non-zero value.

4.3.3 Convolutional Layer

Going back now to components unique to convolutional neural networks, convolutional layers utilise convolution operations instead of activation functions in their neurons. Recalling the point made previously about how convolution operations are effective at extracting simple features (lines, curves) from an image, convolutional layers use this property to extract these features and pass them to subsequent layers to be subject to more convolutions. This has the effect of building up increasingly complex features that the network can recognise and use to come up with its prediction on the contents of an image [5].

The fact that the inputs to a convolutional layer is an image means that these inputs are expressed as 3D tensors, the height and width of the image giving the input its first two dimensions and the colour channels of the image giving the input its depth (3rd dimension).

The output of a convolutional layer is also expressed as a 3D tensor. However, in the case of the output tensor, the 3rd dimension no longer represents the colour channels of the original image, but instead the number of filters(kernels) that have been assigned to that convolutional layer [2]. The diagram below represents this visually:

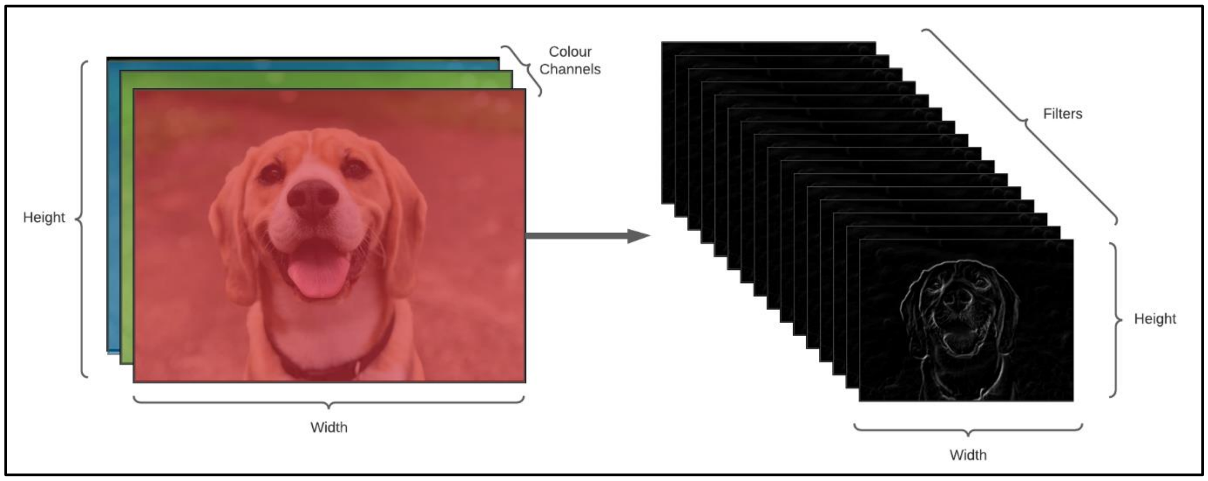


Figure 8: Filter Depth

4.3.4 Pooling Layer

Pooling layers are commonly used in convolutional neural networks for the purpose of easing the processing power a network requires. They function by subsampling a tensor, thereby reducing its spatial size.

A common method of pooling is achieved by creating a 2x2 sized kernel, and extracting only the maximum value of a 2x2 area of the original tensor [6]. This pooling is carried on every layer of the tensor being subject to this operation. This will have the effect of reducing the size of the original tensor by 75%.

4.3.5 Fully-Connected Layer

A fully connected layer is a layer that is fundamental to all neural networks. Every neuron in a fully connected layer is connected to every neuron in the previous layer. Drawing on the fact that each node in a convolutional layer outputs a feature, it therefore makes sense that at some stage in the network, it would be desirable to have a layer whose neurons can receive inputs from every feature that has been detected. This will mean that an individual neuron in a fully connected layer will be able to represent some specific combination of these features [6].

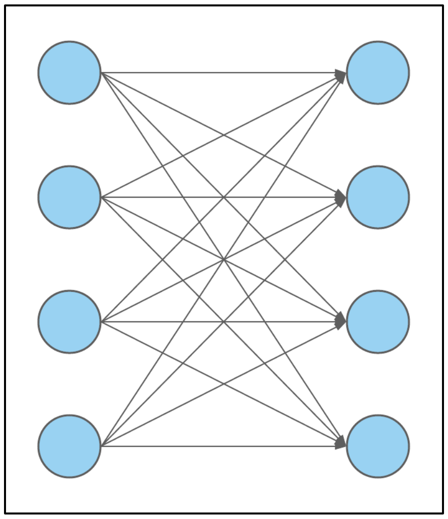


Figure 9: Fully Connected Layer

4.3.6 Output Layer

The output layer of a neural network will be a fully connected layer with a set number of output neurons representing all possible predictions of the network. In the case of a classification model, the output will be a series of classification bins whose values will be the probability of a correct prediction. In the case of a regression model the output layer will be a single or series of neurons of continuous values where each neuron represents a prediction of some specific part of the input.

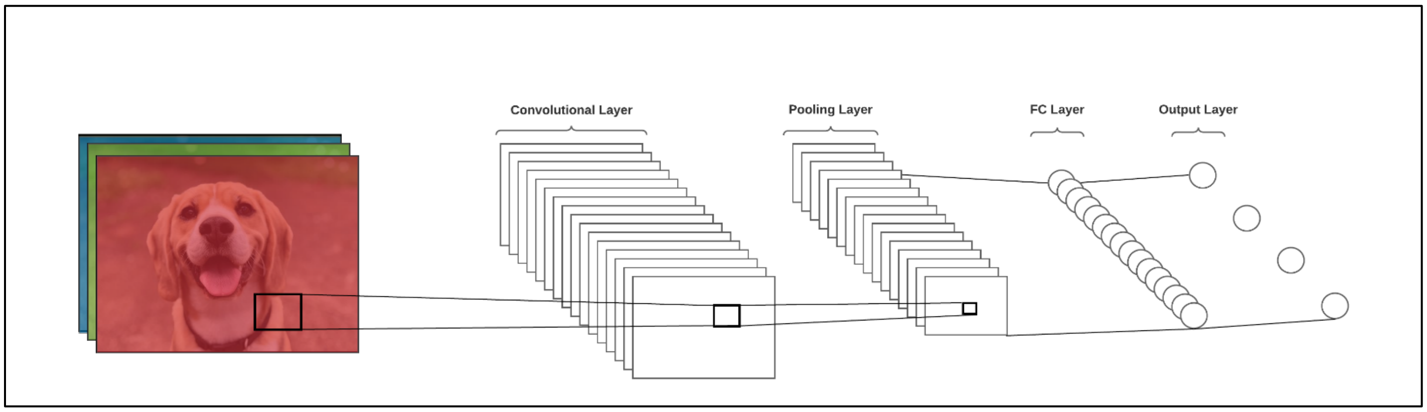


Figure 10: Convolutional Neural Network

5 Supervised Deep Learning

Of the many methods in which deep learning can be used to train a neural network, supervised learning is by far the most common [2]. When attempting to train a neural network to perform a prediction on some input data, supervised learning involves providing that neural network with both the input data and the expected result. With access to both the input data and the expected result, the model can learn from incorrect predictions and change something about the way in which it has reached its incorrect prediction, in the hopes of becoming closer to the correct answer the next time around.

5.1 Labelling

Labelling is the process in which raw data which will be used to train a neural network is assigned meaningful information, so that the network being trained has correct labels, known as ground truth labels, to refer to when deducing the accuracy of its prediction [7].

An example of data labelling for classification could be taking a dataset of images of cats and dogs, and then assigning each image a label, be that '1' for cat and '0' for dog. This way when a classifier makes its prediction, be that '1' or '0', it can compare that prediction to the correct result.

An example of data labelling for regression could be labelling an image of a house with a set price, for example €250,000. When a model makes its prediction it can, again, compare its prediction with the correct result and use that information to adjust its prediction.

5.2 Loss Functions

The method in which a deep learning model quantifies the accuracy of its predictions is via the use of a loss function [6]. A loss function is a mathematical function that causes a value (known as the ‘loss’) to increase or decrease with regards to how different two values are, the higher the loss the more inaccurate the prediction. Applied to supervised learning, the values which are compared are the prediction and the ground truth label. The total loss calculated for a model will be averaged over the size of the entire training dataset [6].

5.2.1 Classification Loss (Cross Entropy)

Neural networks, when used in classification tasks, essentially categorises the input data into several ‘bins’, each of these bins representing a specific category. In the case of a model that predicts whether an image is of a cat, mouse, or dog, there will be a total of three bins, with each one representing one of the animals [5].

Classification models will output a vector of values between one and zero, this vector is of equal size to the number of category bins, with each value in the vector representing the probability that that bin is the correct prediction.

Cross Entropy is a loss function that is commonly used for classification tasks. It calculates negative log of the predicted result multiplied by the expected result. This is performed on each prediction for all items in a training set, so that their results may be summed together and averaged over the size of the set. The equation is contained below:

$$L_i = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

When considering cross entropy for a loss function, it is important to note that for many classification tasks there will be just one correct value. This makes appropriate the usage of one-hot encoding, which is to say, of a vector of possible output bins, only the correct bin will have a value of 1, whereas all the other bins will have a value of 0. This will affect the loss function in that, there will be no summation of incorrect values, as they will be multiplied by 0.

5.2.2 Regression Loss (Mean Squared Error)

Neural networks, when used for regression tasks, are used to predict continuous values, such as in the case of a model that is used to predict number of attendees at a football match and how much they may spend.

The output of models trained for regression will be a vector of size one or greater, with each element of that vector containing a continuous value corresponding to the prediction for the attribute of the input data that you are trying to predict.

Mean Squared Error (MSE) is a common loss function used in regression tasks. For every item in a training set, MSE takes the square of the difference between the prediction and ground truth label, the result of each of these being summed together and averaged over the size of the entire dataset. The equation is contained below:

$$L_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Due to the squaring of the loss in mean squared error, incorrect predictions are punished increasingly the further they are from the ground truth.

5.3 Trainable Parameters

Neural networks on their own would not be much use without the possibility of changing something about how a prediction was achieved. It is the weights and biases associated with a neural network that are subject to change depending on how small or large that networks loss is, these are known as a networks 'trainable parameters'.

Weights are in essence just numbers that are multiplied by each of a neurons input values, whereas bias are numbers which are used to shift an activation function by means of them being added to the entire input. These changes will greatly change the value of the activation of that neuron and will in turn, greatly effect the prediction that is output [2].

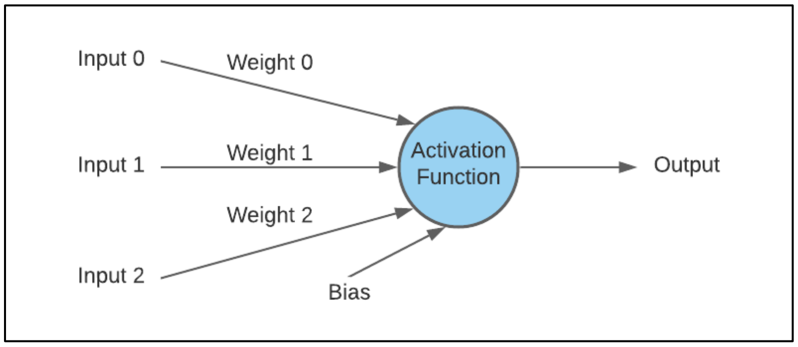


Figure 11: Artificial Neuron with Weights and Bias

5.4 Gradient Descent

Upon initialization, a neural networks trainable parameters are set to 'small-random values', which are randomly chosen values that are close to but not quite zero [6]. To elaborate on what was stated previously; the loss that has been calculated from a networks prediction must be used to alter its trainable parameters with the goal of improving the accuracy of its next prediction. When this process is repeated over successive predictions, the networks accuracy improves greatly.

Gradient-Based Optimization is the method in which the network alters its trainable parameters in the appropriate way, specifically, in the case of attempting to gain a more accurate model, gradient descent is the method that is used to deduce the correct change to make to a parameter.

6 Project Plan

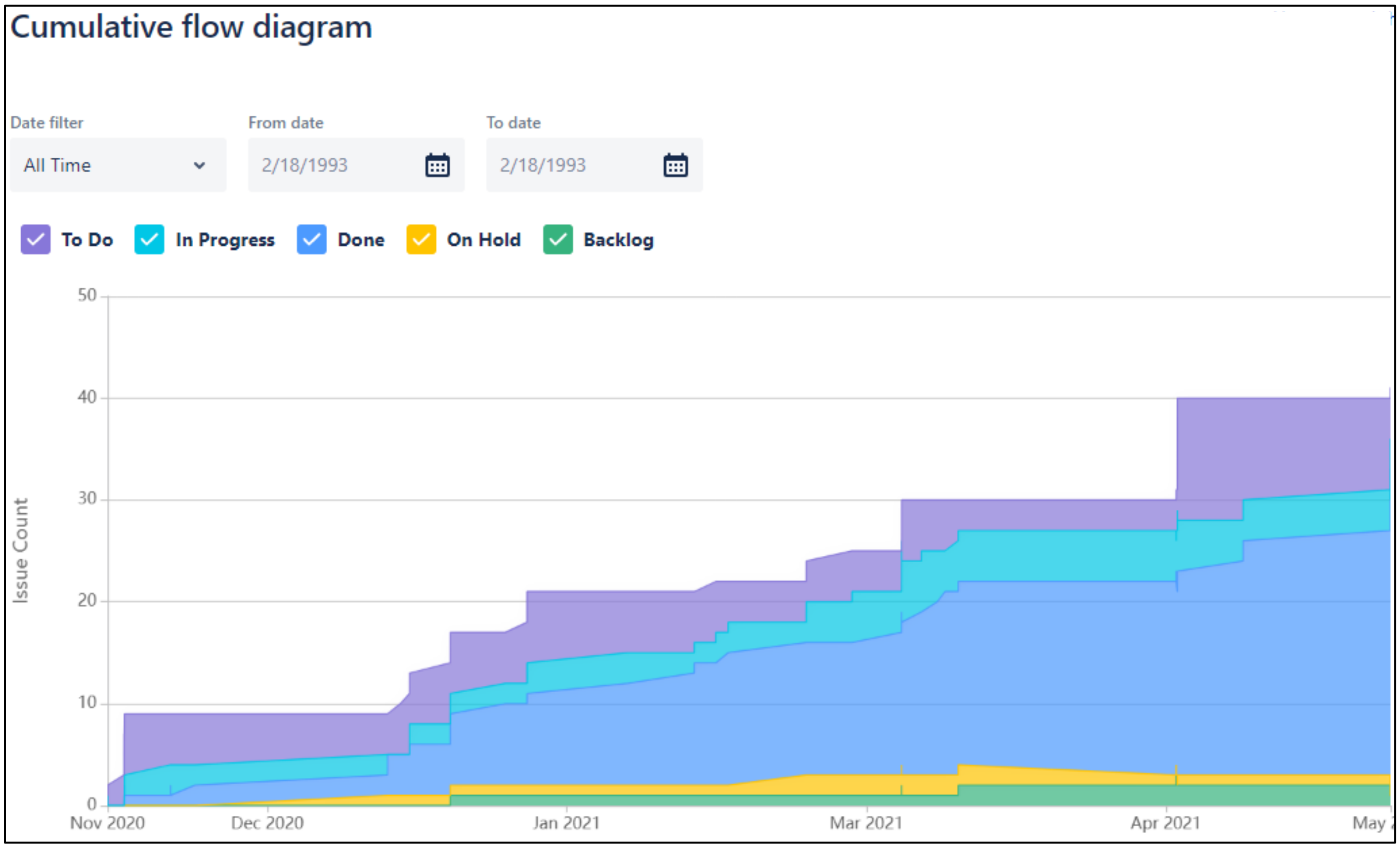


Figure 12: Cumulative Flow Diagram of Goals That Were Set Out Vs Goals Achieved

7 Stages Involved in Image Processing

The method in which image processing is achieved follows a five-stage approach. Two of those stages involve the use of pretrained convolutional neural networks, with the remaining three stages involving the use of methods and algorithms which are used to transform the input data in some way. The five stages are as follows:

1) **Face detection and extraction**

   When an image is first processed by the project, a face detection algorithm is used to detect a face from that image. The face that has been detected is then cropped out of the original image and resized to be used as input data to the convolutional neural network in stage two.

2) **Head Pose Estimation**

   As mentioned in stage 1, the face which has been previously cropped out of the original image is then used as the input data to a pre-trained convolutional neural network. The purpose of this convolutional neural network is to determine the head pose of a person in the original image. The head pose, once determined, will be used as an estimate as to the general direction that a person is looking.

3) **Field of View Masking**

   Now that the general direction that a person is looking has been determined, that information is used to create a field of view which extends a set amount of degrees from the estimated gaze direction. This field of view is then used to create an image mask around the original image. This mask will cover everything that is not in that person's field of view.

4) **Saliency Estimation**

   With the field of view mask created. The masked image will be used as the input to another pre-trained convolutional neural network. This convolutional neural network will be used for saliency estimation of the elements within a person's field of view. This stage will output a saliency map of those elements, which is to say, the areas that the network has deemed most liable to catch the eye will be highlighted in increasing intensities the more salient that area is.

5) **Most salient area calculation**

Finally, seeing as how there are often more than one salient areas in the saliency map of the previous stage, an algorithm has been developed to decide on the most likely of those areas that a person is looking. This is determined by the average intensity of a salient area multiplied by the inverse of the angle between the salient area and the predicted gaze direction calculated in stage two.

8 Project Architecture

An architectural diagram containing the entire process for the project is contained below:

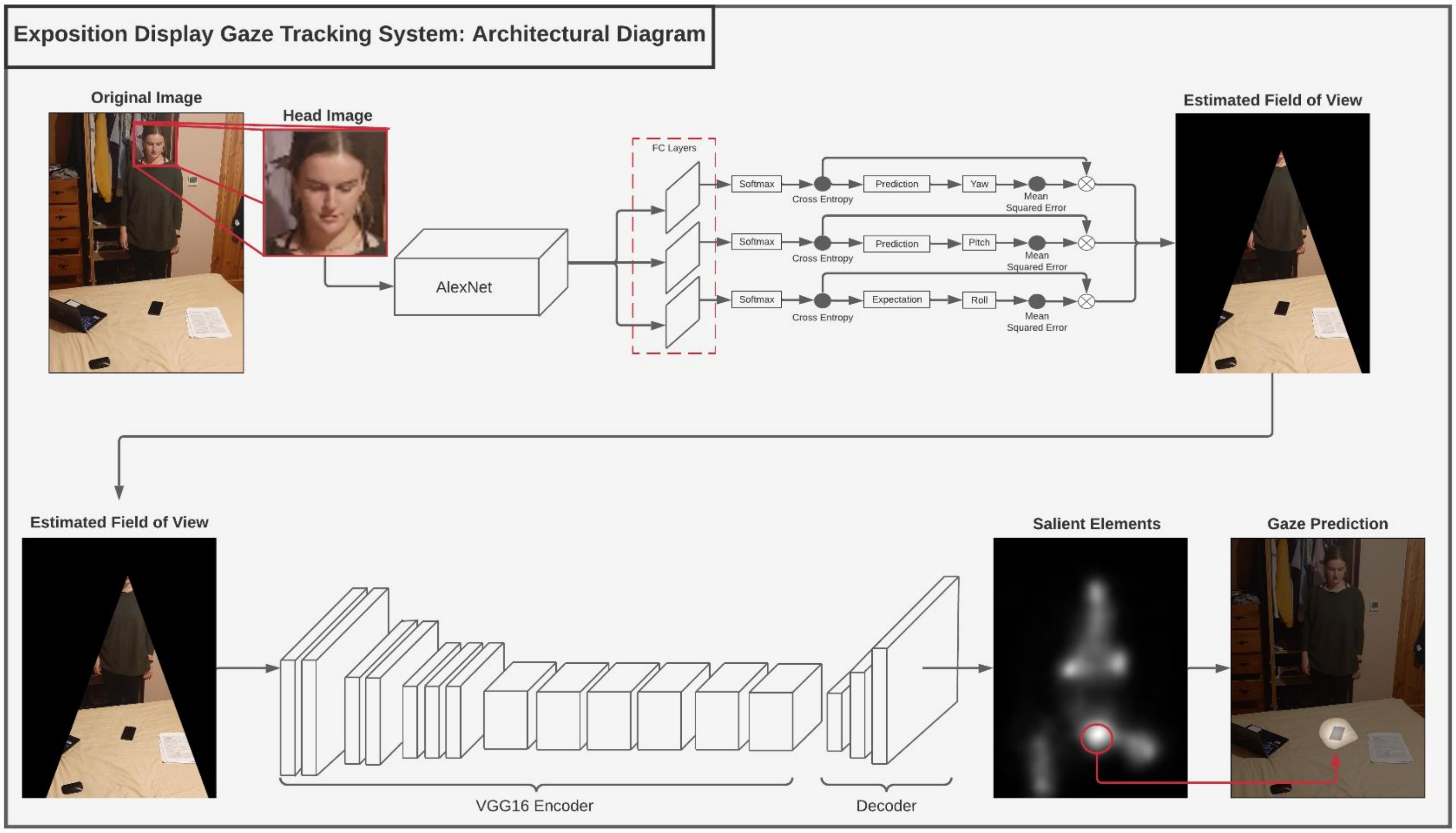


Figure 13: Image Processing Architecture

9 Viola-Jones Face Detection Method

The method in which a face is detected from an image uses fast and robust Viola-Jones face detection algorithm. The principle of this face detection method relies on simple features being used to classify the contents of an image. These features are nothing more than combinations of rectangles, which when applied over an image being analysed, sum the intensity values of the pixels within adjacent rectangles [8]. The difference is then calcualted between the values in adjacent values, with this result being used to determine if a feature is present. This process is then repeated, with each iteration becoming increasingly confident of a face being present. This is what is known as a cascade of classifiers [8].

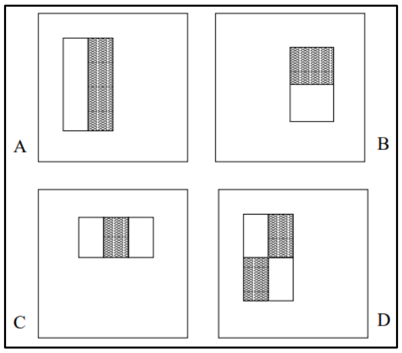


Figure 14: Haar-Like Features [8]

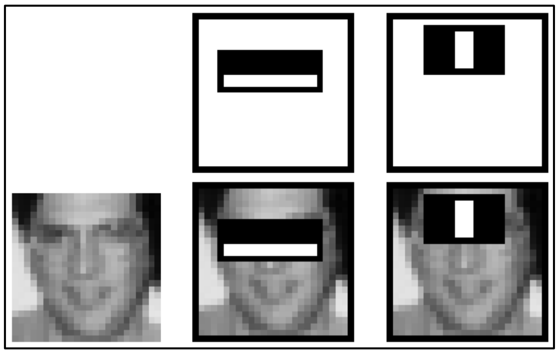


Figure 15: Features Applied to a Face [8]

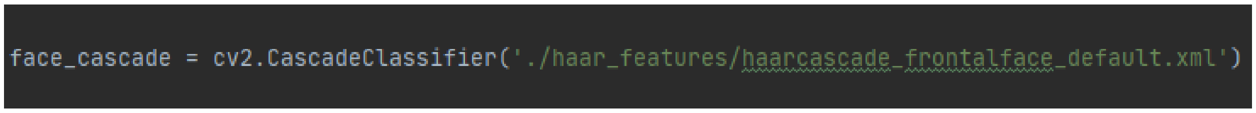

```
face_cascade = cv2.CascadeClassifier('./haar_features/haarcascade_frontalface_default.xml')
```

Figure 16: Creating a Cascade Classifier in Python

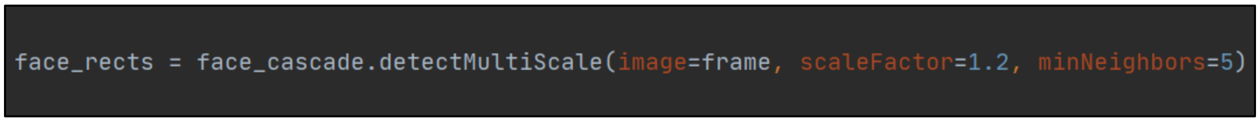

```
face_rects = face_cascade.detectMultiScale(image=frame, scaleFactor=1.2, minNeighbors=5)
```

Figure 17: Detecting a Face Using Previously Initialized Classifier

What makes this algorithm so fast and effective is the fact that upon each iteration of the process described above, the area in an image which is being analysed is reduced greatly. Additionally, the Viola-Jones method incorporates a smart way of quickly calculating the intensities af areas of an image, via the use of what is called an ‘integral image’. The integral image will be the same width and height as the original image, however, each pixel value will equal the sum of all pixel values above and to the left of that pixel [8].

10 Head Pose Estimation

Though there are several methods in which deep learning may be used to achieve head pose estimation, it was decided that the approach to be taken would use the work of Ruiz et al in their Hopenet head pose estimation model [9]. Contrary to using keypoints for this task as is traditionally the case, Hopenet calculates head pose directly from image intensities. According to Ruiz et al. their method results in a more robust way of detecting head pose from an image [9].

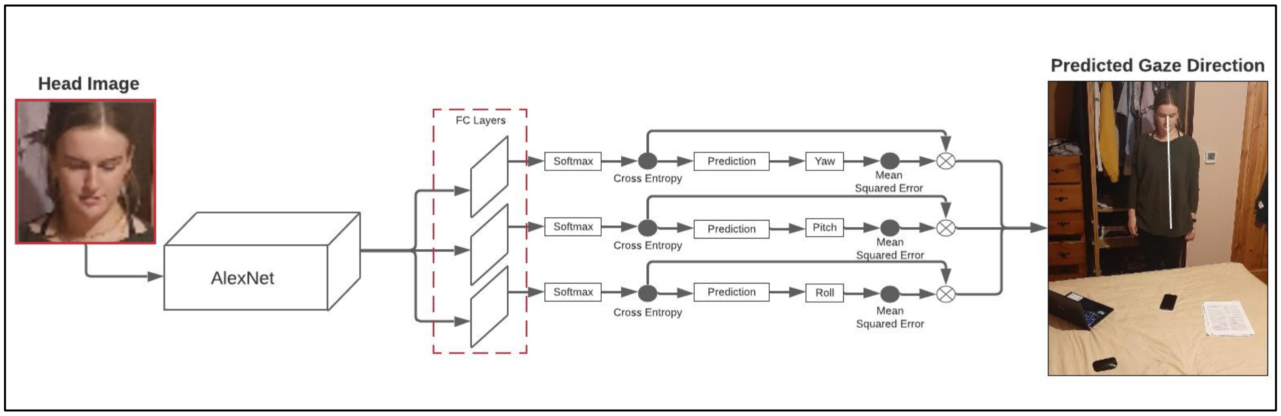


Figure 18: Hopenet Architecture

Hopenet is trained using two datasets, the first of which being the BIWI dataset which contains images and labels that have been taken in a laboratory setting by a Kinect device. The BIWI dataset contains continuous and classification labels for the head pose, as well as depth labels which are not used in this instance. The second dataset is the AFLW2000 dataset, which contains much more difficult ‘in the wild’ images and labels which have a great deal of variation.

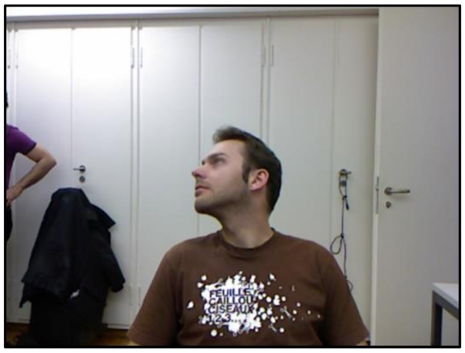
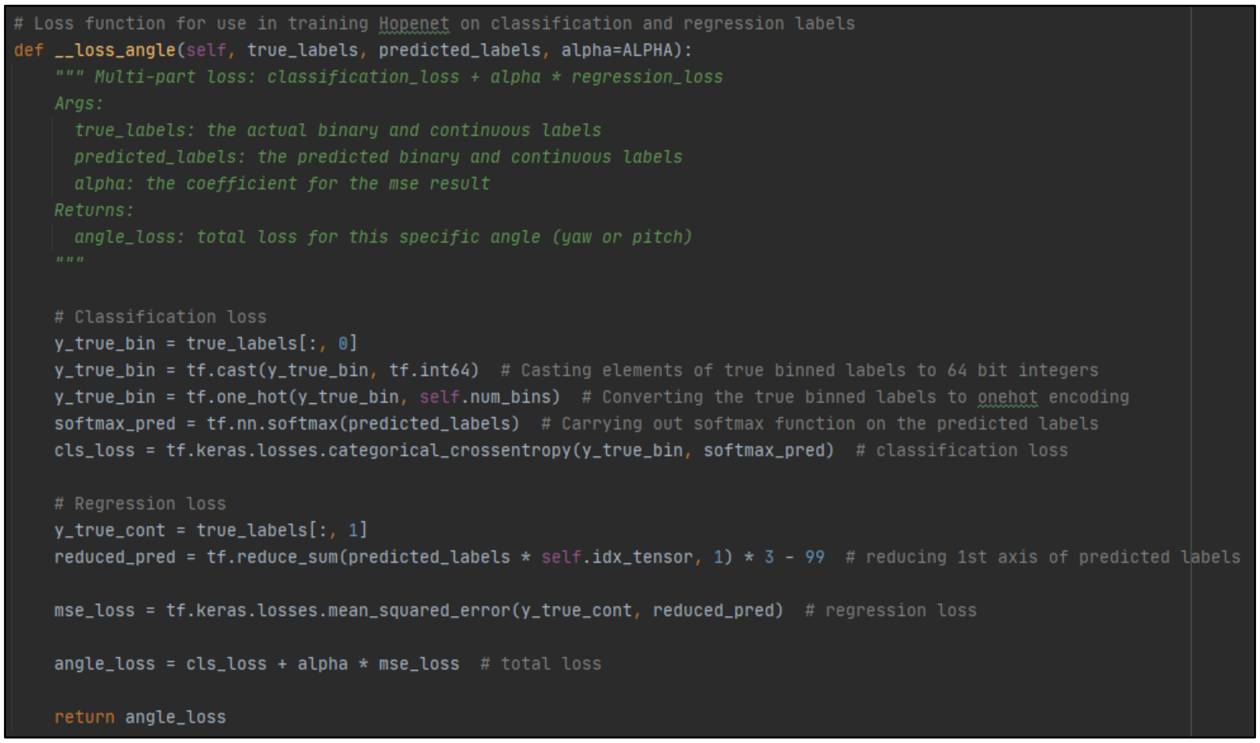


Figure 19: BIWI

Figure 20: AFLW2000

Hopenet achieves its estimation by creating a convolutional neural network with three individual losses, one for each Euler angle (yaw, pitch and roll). Additionally, Hopenet utilises a combination of a classification and regression loss to calculate its final loss, which will be a continuous value representing the predicted angle. The reasoning behind this is that by first performing a classification loss with one-hot encoding, the loss function will output a classification bin representing the neighbourhood of a predicted angle (±3 degrees). This can then be combined with the regression loss to improve the accuracy of a more fine-grained head pose estimation. [9]

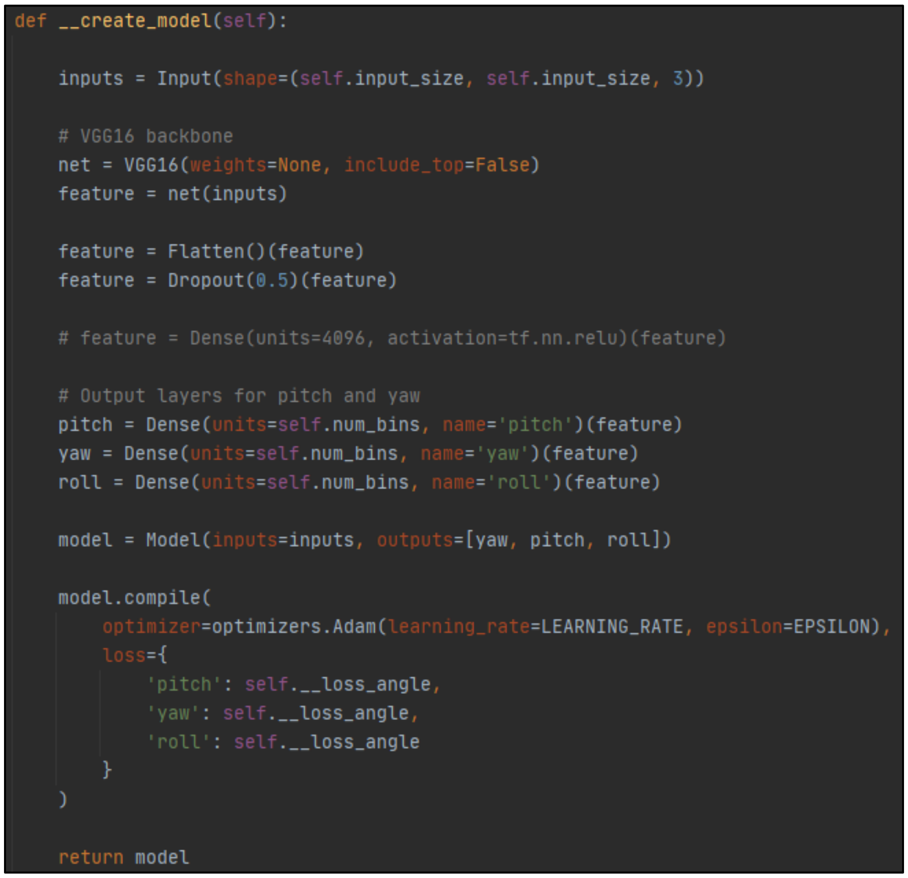


```python
# Loss function for use in training Hopenet on classification and regression labels
def __loss_angle(self, true_labels, predicted_labels, alpha=ALPHA):
    """ Multi-part loss: classification_loss + alpha * regression_loss
    Args:
      true_labels: the actual binary and continuous labels
      predicted_labels: the predicted binary and continuous labels
      alpha: the coefficient for the mse result
    Returns:
      angle_loss: total loss for this specific angle (yaw or pitch)
    """

    # Classification loss
    y_true_bin = true_labels[:, 0]
    y_true_bin = tf.cast(y_true_bin, tf.int64)  # Casting elements of true binned labels to 64 bit integers
    y_true_bin = tf.one_hot(y_true_bin, self.num_bins)  # Converting the true binned labels to onehot encoding
    softmax_pred = tf.nn.softmax(predicted_labels)  # Carrying out softmax function on the predicted labels
    cls_loss = tf.keras.losses.categorical_crossentropy(y_true_bin, softmax_pred)  # classification loss

    # Regression loss
    y_true_cont = true_labels[:, 1]
    reduced_pred = tf.reduce_sum(predicted_labels * self.idx_tensor, 1) * 3 - 99  # reducing 1st axis of predicted labels

    mse_loss = tf.keras.losses.mean_squared_error(y_true_cont, reduced_pred)  # regression loss

    angle_loss = cls_loss + alpha * mse_loss  # total loss

    return angle_loss
```

Figure 21: Multi-Loss Function Adapted from [10]

The backbone network of Hopenet may be any convolutional neural network. What is required to create a Hopenet model, however, is for three fully connect layers to be branched off from that original backbone network. These fully connected layers will form a prediction of their respective Euler angle (yaw, pitch or roll).

In their paper, Ruiz et al. state the losses of their model to be roughly 3 per Euler angle, with a total loss approaching 10. However, in trying to replicate their model using Tensorflow instead of PyTorch, the models trained over the course of this project did not achieve the desired accuracy. Attempts were made to alter the loss function and model structure in the hopes that these changes would improve the performance of the model, it was however the case that the total loss of models rarely reached below 100. Logs from training attempts can be seen in the Appendix.



```
def __create_model(self):

    inputs = Input(shape=(self.input_size, self.input_size, 3))

    # VGG16 backbone
    net = VGG16(weights=None, include_top=False)
    feature = net(inputs)

    feature = Flatten()(feature)
    feature = Dropout(0.5)(feature)

    # feature = Dense(units=4096, activation=tf.nn.relu)(feature)

    # Output layers for pitch and yaw
    pitch = Dense(units=self.num_bins, name='pitch')(feature)
    yaw = Dense(units=self.num_bins, name='yaw')(feature)
    roll = Dense(units=self.num_bins, name='roll')(feature)

    model = Model(inputs=inputs, outputs=[yaw, pitch, roll])

    model.compile(
        optimizer=optimizers.Adam(learning_rate=LEARNING_RATE, epsilon=EPSILON),
        loss={
            'pitch': self.__loss_angle,
            'yaw': self.__loss_angle,
            'roll': self.__loss_angle
        }
    )

    return model
```

Figure 22: Model Used in Attempting to Train Hopenet

11 Field of View Masking

The algorithm developed for field of view masking relies on the gaze direction which has been calculated in the previous stage, as well as the location of the head with regards to its position in the original image. Also important to this function is the theta value, which represents the degree in which the field of view branches each side of the predicted gaze direction.

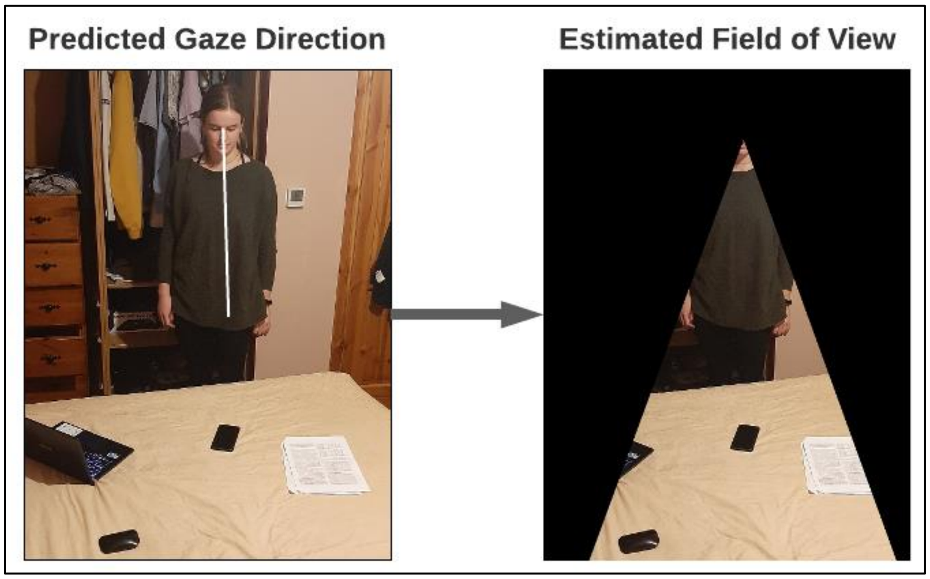
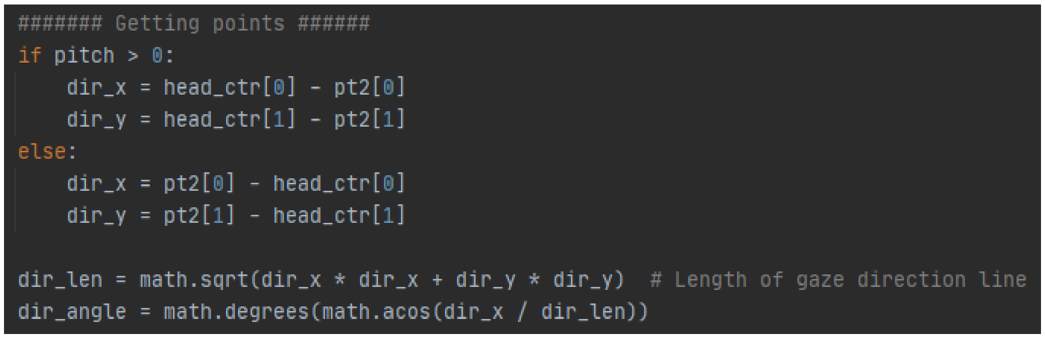


Figure 23: Field of View and Masking

Below is detailed the steps to obtaining the field of view of a person and applying the mask to the areas outside of that field of view:

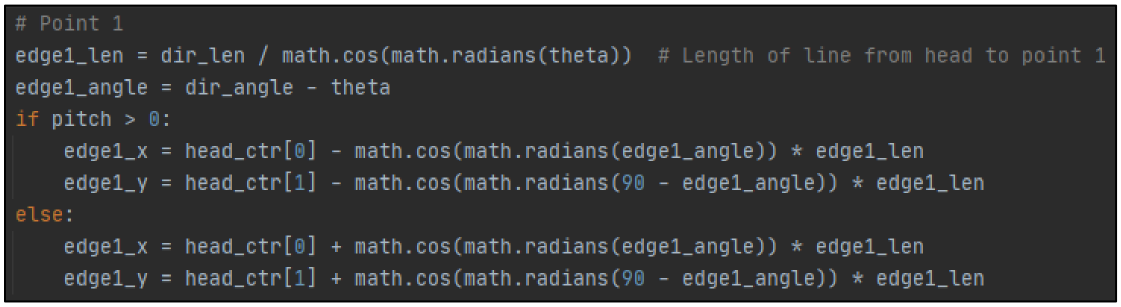
1. The length of the gaze direction line (seen in white in *Figure 23*) must first be calculated so that a triangle may be drawn of the predicted field of view.



```
####### Getting points ######
if pitch > 0:
    dir_x = head_ctr[0] - pt2[0]
    dir_y = head_ctr[1] - pt2[1]
else:
    dir_x = pt2[0] - head_ctr[0]
    dir_y = pt2[1] - head_ctr[1]

dir_len = math.sqrt(dir_x * dir_x + dir_y * dir_y)  # Length of gaze direction line
dir_angle = math.degrees(math.acos(dir_x / dir_len))
```

Figure 24: Calculating Gaze Direction Line

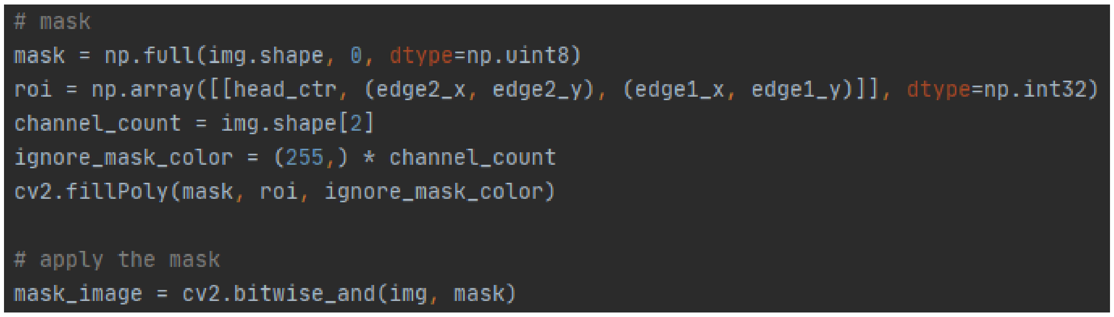
2. Whereas the center of a person's head is one point of the triangle used for the field of view mask, the second two points of the triangle must also be calculated before being able to apply the mask.



```
# Point 1
edge1_len = dir_len / math.cos(math.radians(theta))  # Length of line from head to point 1
edge1_angle = dir_angle - theta
if pitch > 0:
    edge1_x = head_ctr[0] - math.cos(math.radians(edge1_angle)) * edge1_len
    edge1_y = head_ctr[1] - math.cos(math.radians(90 - edge1_angle)) * edge1_len
else:
    edge1_x = head_ctr[0] + math.cos(math.radians(edge1_angle)) * edge1_len
    edge1_y = head_ctr[1] + math.cos(math.radians(90 - edge1_angle)) * edge1_len
```

Figure 25: Calculating Point of FOV Triangle

3. The final step in this stage of the project is to apply the mask to the original image.



```
# mask
mask = np.full(img.shape, 0, dtype=np.uint8)
roi = np.array([[head_ctr, (edge2_x, edge2_y), (edge1_x, edge1_y)]], dtype=np.int32)
channel_count = img.shape[2]
ignore_mask_color = (255,) * channel_count
cv2.fillPoly(mask, roi, ignore_mask_color)

# apply the mask
mask_image = cv2.bitwise_and(img, mask)
```

Figure 26: Creating the FOV Mask

12 Saliency Mapping

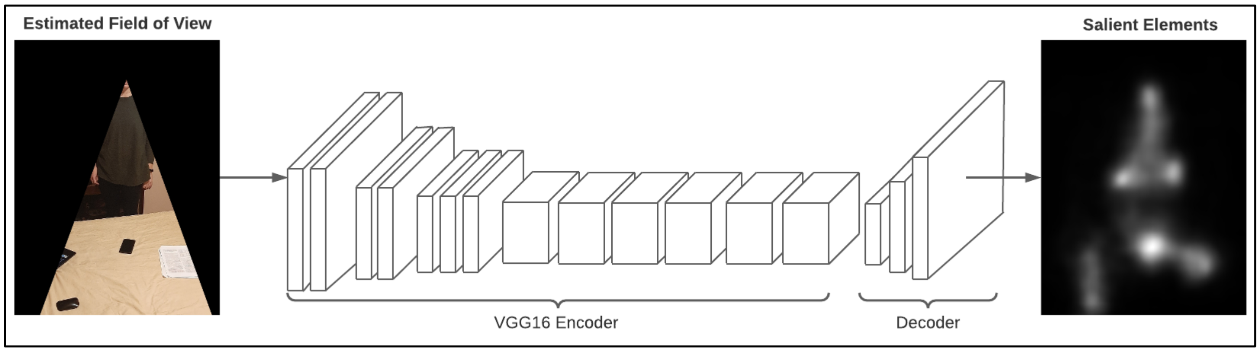


Figure 27: Encoder-Decoder Network for Salienct Mapping

For saliency mapping, it was decided that another convolutional neural network would be used, this time incorporating a pretrained encoder-decoder network called MSI-Net and designed by Kroner et al. [11].

Multiple pre-trained MSI-Net models were available, each of them having been trained on a different dataset. Upon comparison of each of the saliency models available, it was decided that the model trained on the OSIE dataset would be most appropriate for use in the project. This is due to there being more clearly defined salient areas in the saliency map that was output by this model. Below are comparisons of the output of the OSIE trained model, when compared to models trained on other datasets:

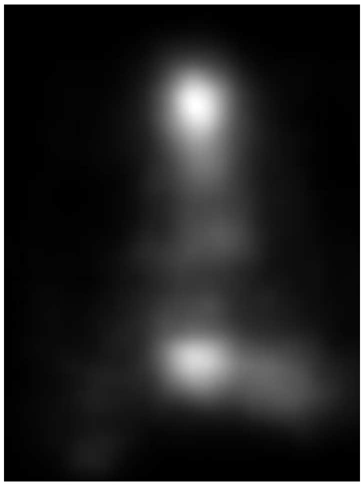


Figure 28: Pascals

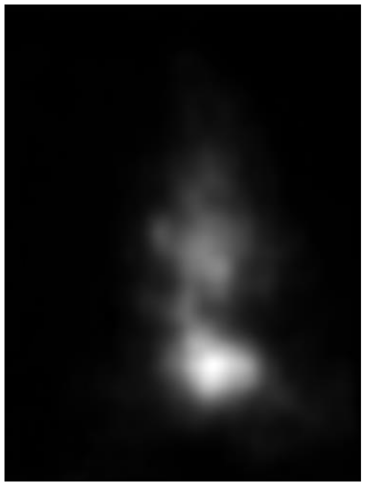


Figure 29: Salicon

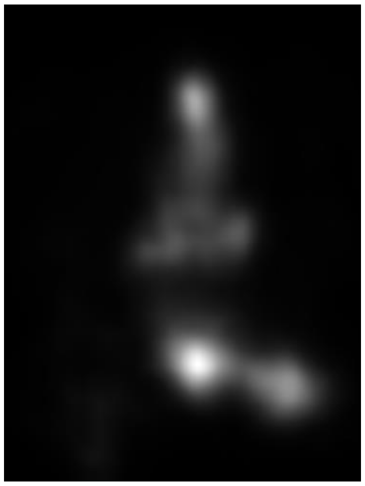


Figure 30: OSIE

13 Calculating Most Salient Area

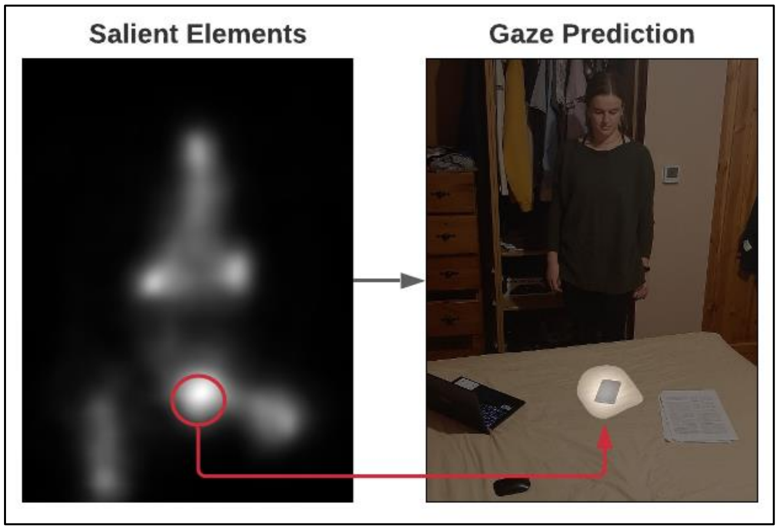
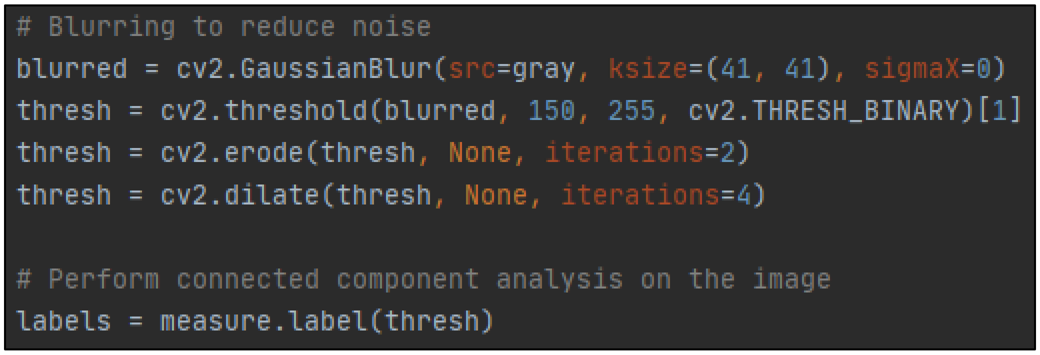


Figure 31: Most Salient Area Applied to Original Image

The saliency map calculated previously undergoes a binary thresholding function in which pixels below a brightness value of 150 are set to black, with pixels above that value being set to white.

A labelling function is then applied to the resulting image which will cause areas that are adjacent to each other and of the same value to be assigned a single label. Due to the fact that sufficiently salient areas of the saliency map have been set to white, the labelling function will result in a list of labels representing each salient area.

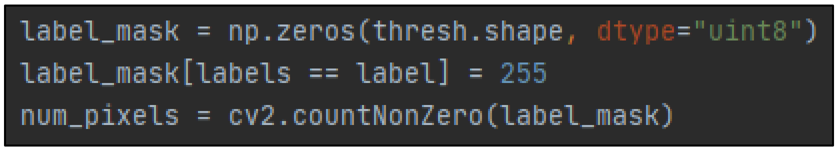


```
# Blurring to reduce noise
blurred = cv2.GaussianBlur(src=gray, ksize=(41, 41), sigmaX=0)
thresh = cv2.threshold(blurred, 150, 255, cv2.THRESH_BINARY)[1]
thresh = cv2.erode(thresh, None, iterations=2)
thresh = cv2.dilate(thresh, None, iterations=4)

# Perform connected component analysis on the image
labels = measure.label(thresh)
```

Figure 32: Thresholding Operation

The list which has previously been created is then iterated over with the following steps being performed on each iteration:

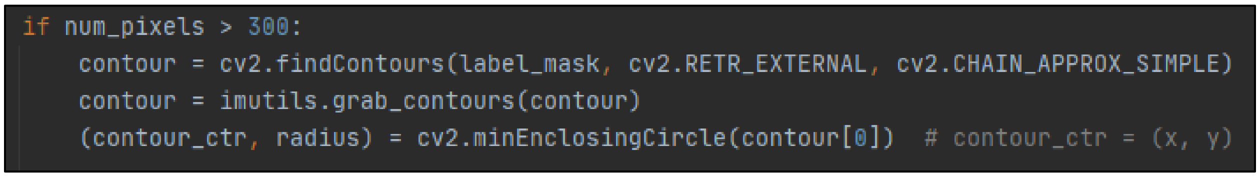
1. A mask is created over everything in the image except the current label. When repeated this will cause every label (and therefore salient area) to have a unique mask.



```
label_mask = np.zeros(thresh.shape, dtype="uint8")
label_mask[labels == label] = 255
num_pixels = cv2.countNonZero(label_mask)
```

Figure 33: Masking Everything but a Label

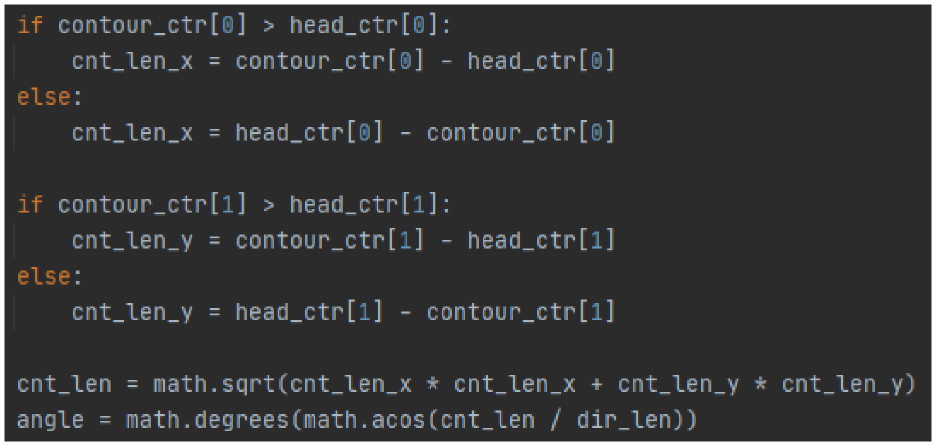
2. Treating a mask as a contour, grabbing that contour and acquiring the minimum size circle that encloses it will return the centre point of the mask, which can be used in calculating its distance from the previously calculated gaze direction.



```
if num_pixels > 300:
    contour = cv2.findContours(label_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contour = imutils.grab_contours(contour)
    (contour_ctr, radius) = cv2.minEnclosingCircle(contour[0])  # contour_ctr = (x, y)
```

Figure 34: Getting the Centre Point of a Label

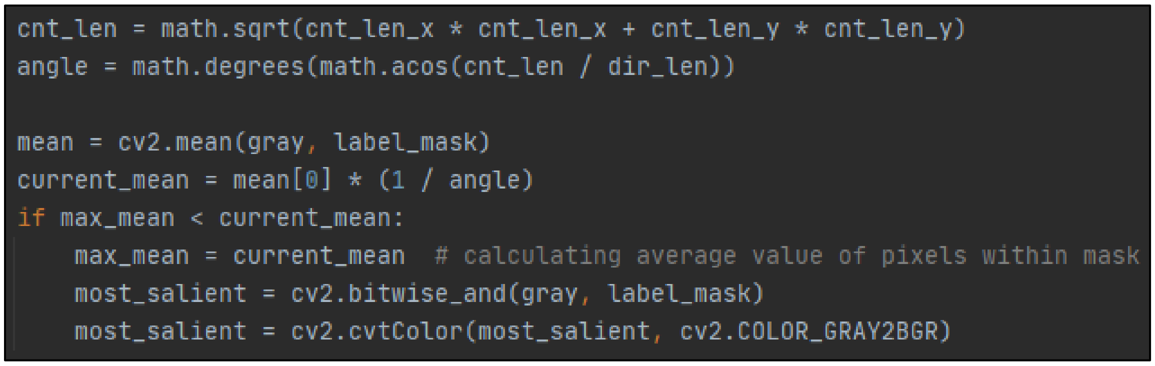
3. The center of the mask which has previously been calculated is then used in combination with the head center point and Pythagoras' theorem to calculate the distance that mask is from a person's head. This is then be used to calculate the angle between the predicted gaze direction and the mask.



```
if contour_ctr[0] > head_ctr[0]:
    cnt_len_x = contour_ctr[0] - head_ctr[0]
else:
    cnt_len_x = head_ctr[0] - contour_ctr[0]

if contour_ctr[1] > head_ctr[1]:
    cnt_len_y = contour_ctr[1] - head_ctr[1]
else:
    cnt_len_y = head_ctr[1] - contour_ctr[1]

cnt_len = math.sqrt(cnt_len_x * cnt_len_x + cnt_len_y * cnt_len_y)
angle = math.degrees(math.acos(cnt_len / dir_len))
```

Figure 35: Calculating Angle Between Mask and Gaze Direction

4. Finally, the mask is applied to the saliency map, meaning it is now possible to calculate the average intensity of pixels in a single salient area. The result is then multiplied by the inverse of the angle which has previously been calculated, giving a smaller total value the larger the angle gets. This is then compared among successive iterations of the label list, with only the most salient area being output and applied to the original image.



```
cnt_len = math.sqrt(cnt_len_x * cnt_len_x + cnt_len_y * cnt_len_y)
angle = math.degrees(math.acos(cnt_len / dir_len))

mean = cv2.mean(gray, label_mask)
current_mean = mean[0] * (1 / angle)
if max_mean < current_mean:
    max_mean = current_mean  # calculating average value of pixels within mask
    most_salient = cv2.bitwise_and(gray, label_mask)
    most_salient = cv2.cvtColor(most_salient, cv2.COLOR_GRAY2BGR)
```

Figure 36: Determining Most Salient Area

14 Ethics

There are several ethical considerations that this project must take into account.

For example, the hypothetical use of a camera system to record where exactly an individual Is looking may cause concerns to arise with how this data is being stored and used. Especially when considering the fact that at an exposition display, anyone who has created a display may have access to a system such as this. Issues may arise where a person wishes to have this information deleted, as a momentary glance at a exposition display may potentially record that person on the systems database.

Additionally, the fact that the area of deep learning is so computationally expensive gives rise to the question of how the carbon emissions output while carrying out deep learning tasks may affect the environment [12].

15 Conclusion

The undertaking of this project has opened an area of software engineering that is both very expansive and extremely interesting.

The goal of creating a gaze tracking system was met, with a program having been developed that is capable of following the stages detailed previously to estimate the most likely area in an image that a person is looking. Several results of this software are detailed below:

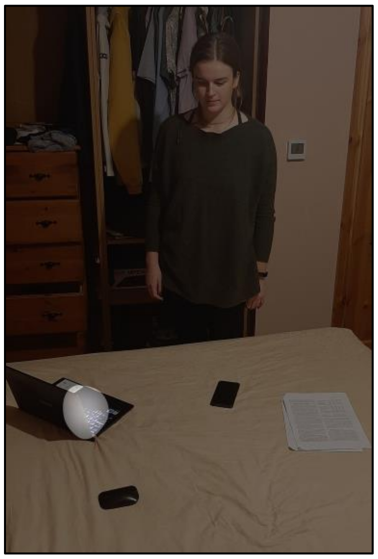


Figure 37: Accurate Result

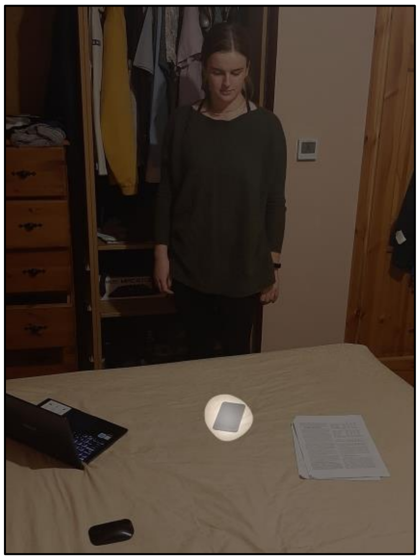


Figure 38: Accurate Result

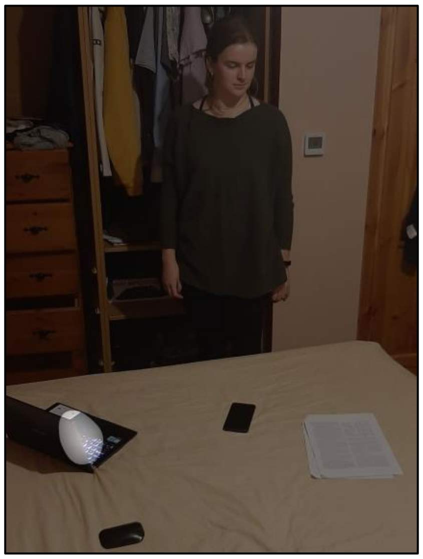


Figure 39: Inaccurate Result

As is seen in the above results, there is an inaccuracy that is present in some predictions. This is caused by the unreliability of the head pose estimation model.

Over the duration of the project there were many attempts to re-train a more effective head pose estimator. Although ultimately the training did not result in a more effective model, the time spent working on fixing the issues that arose has led to a greater understanding of areas such as deep learning and convolutional neural networks. Additionally, the projects reliance on areas of mathematics such as trigonometry and linear algebra has led to a greater understanding of those fields.

The resulting program has much opportunity for further development, and this will undoubtably be pursued in the future.

16 References

[1] D. Lian, Z. Yu and S. Gao, "Believe It or Not, We Know What You Are Looking at!," ACCV, Shanghai, 2018.

[2] F. Chollet, Deep Learning With Python, New York: Manning Publications, 2017.

[3] M. Lynch, *Digital Signal Processing,* Topic: “Image Processing”*,* Galway Mayo Institute of Technology, Galway, 04 2021.

[4] T. Bluche, "Deep Neural Networks Applications in Handwriting Recognition," DOCBOX, 2017 03 09. [Online]. Available: http://technodocbox.com/3D_Graphics/70716176-Deep-neural-networks-applications-in-handwriting-recognition.html. [Accessed 10 05 2021].

[5] J. Portilla, "Python for Computer Vision with OpenCV and Deep Learning," Udemy, 10 05 2020. [Online]. Available: https://www.udemy.com/course/python-for-computer-vision-with-opencv-and-deep-learning/. [Accessed 11 2020].

[6] F.-F. Li, R. Krishna and D. Xu, "CS231n Convolutional Neural Networks for Visual Recognition," 2020. [Online]. Available: http://cs231n.stanford.edu/. [Accessed 12 2020].

[7] Amazon Web Services, "What is data labeling for machine learning?," Amazon Web Services, 2020. [Online]. Available: https://aws.amazon.com/sagemaker/groundtruth/what-is-data-labeling/. [Accessed 05 2021].

[8] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple," IEEE, Cambridge, MA, 2001.

[9] N. Ruiz, E. Chong and J. Rehg, “Fine-Grained Head Pose Estimation Without Keypoints,” Georgia Institute of Technology, Atlanta, USA, 2018.

[10] Oreobird, "tf-keras-deep-head-pose," 2019. [Online]. Available: https://github.com/Oreobird/tf-keras-deep-head-pose. [Accessed 12 2020].

[11] A. Kroner, M. Senden, K. Driessens and R. Goebel, "Contextual encoder–decoder network for visual saliency prediction," Elsevier Ltd., Maastricht, 2020.

[12] R. Toews, "Deep Learning’s Carbon Emissions Problem," Forbes, 17 06 2020. [Online]. Available: https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/?sh=741b998c6b43. [Accessed 05 2021].

[13] N. Ruiz, "Deep Learning Head Pose Estimation using PyTorch," 1 10 2019. [Online]. Available: https://github.com/natanielruiz/deep-head-pose. [Accessed 12 2020].

[14] A. Recasens, A. Kholsa, C. Vondrick and A. Torralba, "Where are they looking?," Advances in Neural Information Processing Systems (NIPS), Boston, 2015.

17 Appendix

aflw_alexnet_100_epoch: After changes to loss calculation

```
1.  100/100 [==============================] - 11s 110ms/step - loss: 1727.1194 - yaw_loss: 625.7621 -
    pitch_loss: 453.5065 - roll_loss: 647.8506
2.  100/100 [==============================] - 11s 111ms/step - loss: 1436.8896 - yaw_loss: 346.9436 -
    pitch_loss: 450.3243 - roll_loss: 639.6218
3.  100/100 [==============================] - 11s 110ms/step - loss: 1347.9008 - yaw_loss: 257.5226 -
    pitch_loss: 451.1996 - roll_loss: 639.1785
4.  100/100 [==============================] - 11s 109ms/step - loss: 1287.3862 - yaw_loss: 200.4888 -
    pitch_loss: 450.1970 - roll_loss: 636.7004
5.  100/100 [==============================] - 11s 111ms/step - loss: 1255.5729 - yaw_loss: 174.2832 -
    pitch_loss: 448.7191 - roll_loss: 632.5707
6.  100/100 [==============================] - 11s 111ms/step - loss: 1226.6729 - yaw_loss: 144.0410 -
    pitch_loss: 449.3367 - roll_loss: 633.2950
7.  100/100 [==============================] - 11s 109ms/step - loss: 1200.5737 - yaw_loss: 131.0278 -
    pitch_loss: 445.4257 - roll_loss: 624.1203
8.  100/100 [==============================] - 11s 108ms/step - loss: 1182.9875 - yaw_loss: 114.6815 -
    pitch_loss: 445.6732 - roll_loss: 622.6332
9.  100/100 [==============================] - 11s 110ms/step - loss: 1134.4795 - yaw_loss: 104.7966 -
    pitch_loss: 433.5980 - roll_loss: 596.0846
10. 100/100 [==============================] - 12s 115ms/step - loss: 1101.7581 - yaw_loss: 124.6765 -
    pitch_loss: 417.0591 - roll_loss: 560.0226
11. 100/100 [==============================] - 12s 123ms/step - loss: 1056.9280 - yaw_loss: 103.8313 -
    pitch_loss: 407.8245 - roll_loss: 545.2723
12. 100/100 [==============================] - 12s 120ms/step - loss: 934.6066 - yaw_loss: 100.8575 -
    pitch_loss: 352.7475 - roll_loss: 481.0014
13. 100/100 [==============================] - 12s 124ms/step - loss: 841.2612 - yaw_loss: 99.9011 -
    pitch_loss: 304.9194 - roll_loss: 436.4405
14. 100/100 [==============================] - 12s 121ms/step - loss: 846.9860 - yaw_loss: 90.5510 -
    pitch_loss: 316.7496 - roll_loss: 439.6852
15. 100/100 [==============================] - 12s 119ms/step - loss: 794.3776 - yaw_loss: 87.4147 -
    pitch_loss: 291.9098 - roll_loss: 415.0529
16. 100/100 [==============================] - 12s 119ms/step - loss: 723.1291 - yaw_loss: 79.5888 -
    pitch_loss: 260.6152 - roll_loss: 382.9252
17. 100/100 [==============================] - 12s 120ms/step - loss: 676.0667 - yaw_loss: 62.9520 -
    pitch_loss: 252.3151 - roll_loss: 360.7997
18. 100/100 [==============================] - 12s 125ms/step - loss: 656.1428 - yaw_loss: 65.7909 -
    pitch_loss: 241.7466 - roll_loss: 348.6053
19. 100/100 [==============================] - 12s 119ms/step - loss: 633.1729 - yaw_loss: 56.6826 -
    pitch_loss: 230.0188 - roll_loss: 346.4716
20. 100/100 [==============================] - 12s 118ms/step - loss: 623.0546 - yaw_loss: 56.8669 -
    pitch_loss: 235.8581 - roll_loss: 330.3295
21. 100/100 [==============================] - 12s 118ms/step - loss: 591.5939 - yaw_loss: 52.7298 -
    pitch_loss: 214.6178 - roll_loss: 324.2460
22. 100/100 [==============================] - 12s 120ms/step - loss: 582.9557 - yaw_loss: 53.0531 -
    pitch_loss: 211.2234 - roll_loss: 318.6792
23. 100/100 [==============================] - 11s 110ms/step - loss: 602.9487 - yaw_loss: 70.4532 -
    pitch_loss: 211.3692 - roll_loss: 321.1264
24. 100/100 [==============================] - 11s 109ms/step - loss: 584.6817 - yaw_loss: 48.9605 -
    pitch_loss: 209.7475 - roll_loss: 325.9736
```

```
25. 100/100 [==============================] - 11s 112ms/step - loss: 555.9922 - yaw_loss: 45.4623 -
    pitch_loss: 199.6550 - roll_loss: 310.8749
26. 100/100 [==============================] - 12s 117ms/step - loss: 555.2040 - yaw_loss: 45.4236 -
    pitch_loss: 197.7171 - roll_loss: 312.0634
27. 100/100 [==============================] - 12s 116ms/step - loss: 541.2362 - yaw_loss: 45.0271 -
    pitch_loss: 191.0498 - roll_loss: 305.1595
28. 100/100 [==============================] - 12s 118ms/step - loss: 540.0010 - yaw_loss: 42.3964 -
    pitch_loss: 192.3823 - roll_loss: 305.2224
29. 100/100 [==============================] - 12s 119ms/step - loss: 536.0859 - yaw_loss: 42.7875 -
    pitch_loss: 187.7192 - roll_loss: 305.5790
30. 100/100 [==============================] - 12s 120ms/step - loss: 532.7598 - yaw_loss: 43.7291 -
    pitch_loss: 184.5083 - roll_loss: 304.5224
31. 100/100 [==============================] - 12s 120ms/step - loss: 529.1252 - yaw_loss: 42.0585 -
    pitch_loss: 182.9059 - roll_loss: 304.1608
32. 100/100 [==============================] - 12s 119ms/step - loss: 521.1782 - yaw_loss: 41.2101 -
    pitch_loss: 178.9192 - roll_loss: 301.0488
33. 100/100 [==============================] - 12s 119ms/step - loss: 517.6623 - yaw_loss: 37.1448 -
    pitch_loss: 179.0101 - roll_loss: 301.5073
34. 100/100 [==============================] - 12s 119ms/step - loss: 510.3538 - yaw_loss: 37.9834 -
    pitch_loss: 174.4417 - roll_loss: 297.9288
35. 100/100 [==============================] - 12s 119ms/step - loss: 514.2227 - yaw_loss: 40.8184 -
    pitch_loss: 174.3985 - roll_loss: 299.0058
36. 100/100 [==============================] - 12s 119ms/step - loss: 513.2996 - yaw_loss: 39.0222 -
    pitch_loss: 175.1626 - roll_loss: 299.1148
37. 100/100 [==============================] - 12s 119ms/step - loss: 512.6870 - yaw_loss: 39.2249 -
    pitch_loss: 174.6852 - roll_loss: 298.7767
38. 100/100 [==============================] - 12s 119ms/step - loss: 509.3445 - yaw_loss: 38.6335 -
    pitch_loss: 172.6185 - roll_loss: 298.0926
39. 100/100 [==============================] - 12s 120ms/step - loss: 504.4936 - yaw_loss: 37.3400 -
    pitch_loss: 171.6862 - roll_loss: 295.4674
40. 100/100 [==============================] - 11s 114ms/step - loss: 503.9762 - yaw_loss: 36.1465 -
    pitch_loss: 171.8439 - roll_loss: 295.9857
41. 100/100 [==============================] - 11s 108ms/step - loss: 502.2696 - yaw_loss: 37.5566 -
    pitch_loss: 168.8316 - roll_loss: 295.8814
42. 100/100 [==============================] - 11s 112ms/step - loss: 499.4351 - yaw_loss: 35.9776 -
    pitch_loss: 167.8213 - roll_loss: 295.6362
43. 100/100 [==============================] - 11s 113ms/step - loss: 500.0975 - yaw_loss: 35.9255 -
    pitch_loss: 168.5747 - roll_loss: 295.5973
44. 100/100 [==============================] - 11s 114ms/step - loss: 503.2715 - yaw_loss: 38.4994 -
    pitch_loss: 168.2152 - roll_loss: 296.5570
45. 100/100 [==============================] - 12s 117ms/step - loss: 501.3778 - yaw_loss: 36.2641 -
    pitch_loss: 169.1450 - roll_loss: 295.9686
46. 100/100 [==============================] - 12s 119ms/step - loss: 498.0812 - yaw_loss: 36.3522 -
    pitch_loss: 167.5214 - roll_loss: 294.2075
47. 100/100 [==============================] - 12s 118ms/step - loss: 494.3483 - yaw_loss: 35.1820 -
    pitch_loss: 165.7119 - roll_loss: 293.4544
48. 100/100 [==============================] - 12s 118ms/step - loss: 496.2869 - yaw_loss: 35.8822 -
    pitch_loss: 166.3972 - roll_loss: 294.0074
49. 100/100 [==============================] - 11s 107ms/step - loss: 493.6137 - yaw_loss: 34.8371 -
    pitch_loss: 165.8742 - roll_loss: 292.9023
50. 100/100 [==============================] - 11s 111ms/step - loss: 489.7904 - yaw_loss: 35.4563 -
    pitch_loss: 164.1978 - roll_loss: 290.1364
```

```
51. 100/100 [==============================] - 11s 112ms/step - loss: 489.5061 - yaw_loss: 35.6369 -
    pitch_loss: 163.7708 - roll_loss: 290.0982
52. 100/100 [==============================] - 12s 116ms/step - loss: 490.5626 - yaw_loss: 35.7111 -
    pitch_loss: 164.6329 - roll_loss: 290.2186
53. 100/100 [==============================] - 12s 116ms/step - loss: 492.0330 - yaw_loss: 35.3725 -
    pitch_loss: 165.1622 - roll_loss: 291.4983
54. 100/100 [==============================] - 12s 118ms/step - loss: 487.3651 - yaw_loss: 34.3957 -
    pitch_loss: 163.9512 - roll_loss: 289.0180
55. 100/100 [==============================] - 12s 119ms/step - loss: 490.8196 - yaw_loss: 34.0048 -
    pitch_loss: 165.1690 - roll_loss: 291.6458
56. 100/100 [==============================] - 12s 118ms/step - loss: 490.5773 - yaw_loss: 35.5279 -
    pitch_loss: 165.2547 - roll_loss: 289.7946
57. 100/100 [==============================] - 11s 112ms/step - loss: 488.0269 - yaw_loss: 34.2344 -
    pitch_loss: 164.2714 - roll_loss: 289.5211
58. 100/100 [==============================] - 11s 108ms/step - loss: 485.5295 - yaw_loss: 33.9529 -
    pitch_loss: 163.3665 - roll_loss: 288.2103
59. 100/100 [==============================] - 11s 110ms/step - loss: 486.5560 - yaw_loss: 34.4307 -
    pitch_loss: 162.9999 - roll_loss: 289.1252
60. 100/100 [==============================] - 11s 111ms/step - loss: 488.8658 - yaw_loss: 34.5828 -
    pitch_loss: 164.5763 - roll_loss: 289.7066
61. 100/100 [==============================] - 13s 126ms/step - loss: 486.2745 - yaw_loss: 34.7226 -
    pitch_loss: 162.9590 - roll_loss: 288.5929
62. 100/100 [==============================] - 13s 127ms/step - loss: 482.8137 - yaw_loss: 33.4526 -
    pitch_loss: 162.4112 - roll_loss: 286.9498
63. 100/100 [==============================] - 13s 128ms/step - loss: 483.5584 - yaw_loss: 33.3222 -
    pitch_loss: 162.4858 - roll_loss: 287.7504
64. 100/100 [==============================] - 12s 119ms/step - loss: 482.9295 - yaw_loss: 33.5262 -
    pitch_loss: 162.4553 - roll_loss: 286.9480
65. 100/100 [==============================] - 12s 122ms/step - loss: 482.4942 - yaw_loss: 34.1732 -
    pitch_loss: 161.5473 - roll_loss: 286.7738
66. 100/100 [==============================] - 12s 120ms/step - loss: 485.9164 - yaw_loss: 33.7356 -
    pitch_loss: 163.5018 - roll_loss: 288.6792
67. 100/100 [==============================] - 12s 119ms/step - loss: 485.5847 - yaw_loss: 33.9612 -
    pitch_loss: 162.5696 - roll_loss: 289.0539
68. 100/100 [==============================] - 12s 118ms/step - loss: 489.8030 - yaw_loss: 35.1300 -
    pitch_loss: 164.5726 - roll_loss: 290.1004
69. 100/100 [==============================] - 12s 119ms/step - loss: 487.8918 - yaw_loss: 35.2242 -
    pitch_loss: 163.5050 - roll_loss: 289.1625
70. 100/100 [==============================] - 12s 119ms/step - loss: 490.7849 - yaw_loss: 35.1940 -
    pitch_loss: 165.5822 - roll_loss: 290.0086
71. 100/100 [==============================] - 12s 118ms/step - loss: 495.9231 - yaw_loss: 36.6857 -
    pitch_loss: 168.0815 - roll_loss: 291.1559
72. 100/100 [==============================] - 13s 125ms/step - loss: 487.7472 - yaw_loss: 34.1690 -
    pitch_loss: 164.5120 - roll_loss: 289.0662
73. 100/100 [==============================] - 12s 116ms/step - loss: 481.8697 - yaw_loss: 32.5612 -
    pitch_loss: 161.3362 - roll_loss: 287.9723
74. 100/100 [==============================] - 11s 114ms/step - loss: 481.1468 - yaw_loss: 33.3166 -
    pitch_loss: 160.8670 - roll_loss: 286.9630
75. 100/100 [==============================] - 11s 113ms/step - loss: 482.3919 - yaw_loss: 33.2653 -
    pitch_loss: 161.3510 - roll_loss: 287.7756
76. 100/100 [==============================] - 12s 123ms/step - loss: 489.0421 - yaw_loss: 39.2696 -
    pitch_loss: 161.3201 - roll_loss: 288.4525
```

```
77.  100/100 [==============================] - 12s 122ms/step - loss: 488.2017 - yaw_loss: 36.4854 -
     pitch_loss: 162.8503 - roll_loss: 288.8661
78.  100/100 [==============================] - 12s 121ms/step - loss: 483.2513 - yaw_loss: 34.1958 -
     pitch_loss: 161.3473 - roll_loss: 287.7084
79.  100/100 [==============================] - 12s 125ms/step - loss: 481.5309 - yaw_loss: 32.9271 -
     pitch_loss: 161.0504 - roll_loss: 287.5533
80.  100/100 [==============================] - 12s 123ms/step - loss: 483.0082 - yaw_loss: 33.4360 -
     pitch_loss: 161.5654 - roll_loss: 288.0068
81.  100/100 [==============================] - 12s 118ms/step - loss: 480.0634 - yaw_loss: 31.8695 -
     pitch_loss: 161.5460 - roll_loss: 286.6479
82.  100/100 [==============================] - 12s 118ms/step - loss: 477.4349 - yaw_loss: 31.5137 -
     pitch_loss: 159.9353 - roll_loss: 285.9859
83.  100/100 [==============================] - 12s 118ms/step - loss: 475.8527 - yaw_loss: 31.2968 -
     pitch_loss: 159.1708 - roll_loss: 285.3851
84.  100/100 [==============================] - 12s 120ms/step - loss: 476.7733 - yaw_loss: 31.5613 -
     pitch_loss: 159.6099 - roll_loss: 285.6020
85.  100/100 [==============================] - 12s 118ms/step - loss: 477.8672 - yaw_loss: 32.2783 -
     pitch_loss: 159.9238 - roll_loss: 285.6651
86.  100/100 [==============================] - 12s 118ms/step - loss: 480.0558 - yaw_loss: 32.8348 -
     pitch_loss: 161.2174 - roll_loss: 286.0034
87.  100/100 [==============================] - 12s 118ms/step - loss: 477.7661 - yaw_loss: 31.2076 -
     pitch_loss: 160.0353 - roll_loss: 286.5232
88.  100/100 [==============================] - 12s 119ms/step - loss: 476.0893 - yaw_loss: 31.3422 -
     pitch_loss: 159.8776 - roll_loss: 284.8695
89.  100/100 [==============================] - 12s 123ms/step - loss: 476.4526 - yaw_loss: 32.2915 -
     pitch_loss: 159.0617 - roll_loss: 285.0996
90.  100/100 [==============================] - 12s 120ms/step - loss: 477.0620 - yaw_loss: 31.1511 -
     pitch_loss: 159.7575 - roll_loss: 286.1534
91.  100/100 [==============================] - 12s 119ms/step - loss: 476.7468 - yaw_loss: 31.2544 -
     pitch_loss: 160.0377 - roll_loss: 285.4549
92.  100/100 [==============================] - 12s 119ms/step - loss: 595.6812 - yaw_loss: 32.0601 -
     pitch_loss: 158.7689 - roll_loss: 404.8525
93.  100/100 [==============================] - 11s 114ms/step - loss: 480.0592 - yaw_loss: 32.8978 -
     pitch_loss: 158.8327 - roll_loss: 288.3288
94.  100/100 [==============================] - 11s 108ms/step - loss: 476.9391 - yaw_loss: 32.4410 -
     pitch_loss: 158.4604 - roll_loss: 286.0376
95.  100/100 [==============================] - 11s 111ms/step - loss: 475.8906 - yaw_loss: 32.0301 -
     pitch_loss: 157.3069 - roll_loss: 286.5536
96.  100/100 [==============================] - 11s 112ms/step - loss: 474.7669 - yaw_loss: 30.9523 -
     pitch_loss: 158.0263 - roll_loss: 285.7884
97.  100/100 [==============================] - 12s 117ms/step - loss: 471.6575 - yaw_loss: 30.5067 -
     pitch_loss: 156.5418 - roll_loss: 284.6092
98.  100/100 [==============================] - 12s 118ms/step - loss: 472.8683 - yaw_loss: 31.0228 -
     pitch_loss: 156.9403 - roll_loss: 284.9049
99.  100/100 [==============================] - 12s 121ms/step - loss: 471.3425 - yaw_loss: 30.3887 -
     pitch_loss: 156.0777 - roll_loss: 284.8761
100. 100/100 [==============================] - 12s 119ms/step - loss: 469.7290 - yaw_loss: 29.6956 -
     pitch_loss: 155.8293 - roll_loss: 284.2040
```

aflw_alexnet_50_epoch: After changing pitch and roll positions in input vector

```
1.  100/100 [==============================] - 22s 224ms/step - loss: 1424.6239 - yaw_loss: 634.4656 -
    pitch_loss: 427.8045 - roll_loss: 362.3536
2.  100/100 [==============================] - 11s 108ms/step - loss: 1122.0498 - yaw_loss: 338.2392 -
    pitch_loss: 423.1308 - roll_loss: 360.6798
3.  100/100 [==============================] - 10s 105ms/step - loss: 1001.8621 - yaw_loss: 225.4150 -
    pitch_loss: 418.7495 - roll_loss: 357.6974
4.  100/100 [==============================] - 12s 118ms/step - loss: 931.9810 - yaw_loss: 161.2678 -
    pitch_loss: 414.1158 - roll_loss: 356.5970
5.  100/100 [==============================] - 12s 116ms/step - loss: 889.6069 - yaw_loss: 126.2058 -
    pitch_loss: 409.2032 - roll_loss: 354.1978
6.  100/100 [==============================] - 11s 113ms/step - loss: 875.3649 - yaw_loss: 117.5091 -
    pitch_loss: 405.7725 - roll_loss: 352.0833
7.  100/100 [==============================] - 11s 108ms/step - loss: 846.6315 - yaw_loss: 100.4874 -
    pitch_loss: 398.3637 - roll_loss: 347.7805
8.  100/100 [==============================] - 11s 107ms/step - loss: 814.5033 - yaw_loss: 97.4398 -
    pitch_loss: 381.1705 - roll_loss: 335.8931
9.  100/100 [==============================] - 11s 107ms/step - loss: 791.0829 - yaw_loss: 107.1149 -
    pitch_loss: 362.1246 - roll_loss: 321.8434
10. 100/100 [==============================] - 11s 114ms/step - loss: 710.4825 - yaw_loss: 111.5597 -
    pitch_loss: 322.5679 - roll_loss: 276.3548
11. 100/100 [==============================] - 11s 112ms/step - loss: 750.6697 - yaw_loss: 121.2999 -
    pitch_loss: 332.3142 - roll_loss: 297.0556
12. 100/100 [==============================] - 11s 113ms/step - loss: 616.5737 - yaw_loss: 77.8766 -
    pitch_loss: 281.8337 - roll_loss: 256.8633
13. 100/100 [==============================] - 11s 115ms/step - loss: 509.0965 - yaw_loss: 69.2064 -
    pitch_loss: 232.7939 - roll_loss: 207.0961
14. 100/100 [==============================] - 12s 115ms/step - loss: 453.0780 - yaw_loss: 57.9506 -
    pitch_loss: 208.1574 - roll_loss: 186.9700
15. 100/100 [==============================] - 12s 116ms/step - loss: 419.2281 - yaw_loss: 51.9415 -
    pitch_loss: 195.1638 - roll_loss: 172.1227
16. 100/100 [==============================] - 11s 115ms/step - loss: 402.5083 - yaw_loss: 51.6329 -
    pitch_loss: 181.2604 - roll_loss: 169.6148
17. 100/100 [==============================] - 11s 114ms/step - loss: 360.6525 - yaw_loss: 44.4271 -
    pitch_loss: 167.1707 - roll_loss: 149.0546
18. 100/100 [==============================] - 11s 114ms/step - loss: 345.8927 - yaw_loss: 44.7546 -
    pitch_loss: 155.6954 - roll_loss: 145.4427
19. 100/100 [==============================] - 11s 115ms/step - loss: 320.5545 - yaw_loss: 41.9848 -
    pitch_loss: 137.4221 - roll_loss: 141.1478
20. 100/100 [==============================] - 12s 121ms/step - loss: 304.3399 - yaw_loss: 36.2982 -
    pitch_loss: 132.5190 - roll_loss: 135.5227
21. 100/100 [==============================] - 12s 117ms/step - loss: 280.0306 - yaw_loss: 33.9416 -
    pitch_loss: 122.4467 - roll_loss: 123.6424
22. 100/100 [==============================] - 11s 114ms/step - loss: 274.0674 - yaw_loss: 32.3405 -
    pitch_loss: 118.1923 - roll_loss: 123.5345
23. 100/100 [==============================] - 11s 114ms/step - loss: 265.0361 - yaw_loss: 31.5453 -
    pitch_loss: 112.7761 - roll_loss: 120.7147
24. 100/100 [==============================] - 12s 116ms/step - loss: 259.1096 - yaw_loss: 31.7671 -
    pitch_loss: 107.8214 - roll_loss: 119.5210
25. 100/100 [==============================] - 11s 114ms/step - loss: 245.6238 - yaw_loss: 29.4467 -
    pitch_loss: 102.9820 - roll_loss: 113.1951
```

```
26. 100/100 [==============================] - 11s 114ms/step - loss: 236.7916 - yaw_loss: 27.8804 -
    pitch_loss: 97.7954 - roll_loss: 111.1159
27. 100/100 [==============================] - 11s 114ms/step - loss: 236.8829 - yaw_loss: 28.8795 -
    pitch_loss: 99.4620 - roll_loss: 108.5414
28. 100/100 [==============================] - 12s 120ms/step - loss: 223.3411 - yaw_loss: 26.7964 -
    pitch_loss: 94.1539 - roll_loss: 102.3908
29. 100/100 [==============================] - 12s 119ms/step - loss: 217.9793 - yaw_loss: 24.4651 -
    pitch_loss: 94.5910 - roll_loss: 98.9232
30. 100/100 [==============================] - 12s 122ms/step - loss: 215.9996 - yaw_loss: 25.0401 -
    pitch_loss: 93.5106 - roll_loss: 97.4489
31. 100/100 [==============================] - 12s 115ms/step - loss: 210.6266 - yaw_loss: 25.1143 -
    pitch_loss: 91.5331 - roll_loss: 93.9791
32. 100/100 [==============================] - 11s 114ms/step - loss: 204.7803 - yaw_loss: 23.4003 -
    pitch_loss: 88.6986 - roll_loss: 92.6813
33. 100/100 [==============================] - 11s 115ms/step - loss: 198.0882 - yaw_loss: 22.1045 -
    pitch_loss: 88.4075 - roll_loss: 87.5762
34. 100/100 [==============================] - 12s 117ms/step - loss: 198.6122 - yaw_loss: 20.5873 -
    pitch_loss: 87.8568 - roll_loss: 90.1682
35. 100/100 [==============================] - 12s 115ms/step - loss: 192.1350 - yaw_loss: 20.7177 -
    pitch_loss: 86.2110 - roll_loss: 85.2063
36. 100/100 [==============================] - 11s 115ms/step - loss: 186.0444 - yaw_loss: 18.3982 -
    pitch_loss: 83.5436 - roll_loss: 84.1027
37. 100/100 [==============================] - 12s 118ms/step - loss: 186.3601 - yaw_loss: 19.7274 -
    pitch_loss: 84.8218 - roll_loss: 81.8109
38. 100/100 [==============================] - 12s 118ms/step - loss: 187.6512 - yaw_loss: 18.8406 -
    pitch_loss: 86.0443 - roll_loss: 82.7663
39. 100/100 [==============================] - 12s 117ms/step - loss: 186.4963 - yaw_loss: 19.1994 -
    pitch_loss: 85.0294 - roll_loss: 82.2674
40. 100/100 [==============================] - 12s 115ms/step - loss: 180.7737 - yaw_loss: 17.8584 -
    pitch_loss: 83.1850 - roll_loss: 79.7303
41. 100/100 [==============================] - 11s 115ms/step - loss: 185.8297 - yaw_loss: 19.0914 -
    pitch_loss: 84.7261 - roll_loss: 82.0122
42. 100/100 [==============================] - 12s 118ms/step - loss: 188.4058 - yaw_loss: 19.9834 -
    pitch_loss: 86.1797 - roll_loss: 82.2426
43. 100/100 [==============================] - 12s 119ms/step - loss: 187.3084 - yaw_loss: 19.0857 -
    pitch_loss: 87.3821 - roll_loss: 80.8406
44. 100/100 [==============================] - 12s 116ms/step - loss: 186.6157 - yaw_loss: 19.0033 -
    pitch_loss: 83.8750 - roll_loss: 83.7375
45. 100/100 [==============================] - 12s 116ms/step - loss: 179.1542 - yaw_loss: 17.7438 -
    pitch_loss: 82.5970 - roll_loss: 78.8133
46. 100/100 [==============================] - 12s 117ms/step - loss: 174.9344 - yaw_loss: 18.2622 -
    pitch_loss: 80.3327 - roll_loss: 76.3395
47. 100/100 [==============================] - 11s 114ms/step - loss: 172.0413 - yaw_loss: 17.3555 -
    pitch_loss: 79.2461 - roll_loss: 75.4397
48. 100/100 [==============================] - 11s 114ms/step - loss: 169.9687 - yaw_loss: 15.3461 -
    pitch_loss: 80.1898 - roll_loss: 74.4329
49. 100/100 [==============================] - 11s 114ms/step - loss: 176.2995 - yaw_loss: 17.1999 -
    pitch_loss: 81.3897 - roll_loss: 77.7099
50. 100/100 [==============================] - 11s 114ms/step - loss: 172.9588 - yaw_loss: 17.5715 -
    pitch_loss: 80.5914 - roll_loss: 74.7958
```