Towards One-Shot Iris Recognition

Sahil Jain[a] and Theus H. Aspiras[a]

[a]University of Dayton, 300 College Park, Dayton, USA

ABSTRACT

Iris recognition is a highly sought-after technology in security-based environments. Especially in locations requiring persistent surveillance, recognizing unauthorized personnel is necessary, and iris recognition is considered a reliable technology. Though this technology is considered non-invasive, all personnel are required to be very close to the system to work. We propose a more non-invasive iris recognition system that utilizes facial land-marking and one-shot learning to provide longer-range recognition of personnel.

1. INTRODUCTION

Background of Iris Recognition - Many biometric systems exist today such as facial recognition, fingerprint scanning, and Iris recognition. The Iris is the annular region within the human eye bounded by the pupil and sclera (white part). Iris recognition has become popular among researchers due to its distinct nature offering better stability and recognition of a person.[1]

Most of the work done in identifying and verifying iris patterns began in the 1990s. The first iris recognition system was developed by John Daugman in 1993. Daugman's use of integro-differential operator laid the foundation for future IR systems as an application in pattern recognition. Today, we have come a long way in the detail in which an iris can be recognized. Modern-day Biometric iris recognition scanners illuminate the iris with infrared light to pick up unique patterns and details not visible to the naked eye.[1]

Contribution - The problem though with some of today's Iris recognition software requires users to place their eye very close to a camera to properly capture and read the iris. This method works well, but it requires the user to be very still keeping their head straight and eyes facing directly towards the camera, making it not very user-friendly. This paper introduces a novel approach to mapping a portrait using the facial landmarking features of Mediapipe to detect the iris region of the eye and a Keras neural network to recognize this region of interest and assign values to the features that can be used to identify a person's iris.

2. RELATED WORK

2.1 Iris Detection

Current techniques that exist in the detection of the iris involve taking a close-up image of the eye and locating the iris by Circular Hough transform. Circular Hough Transform (CHT), are fundamental tools in computer vision for detecting shapes like lines and circles within an image.

The basic idea behind the Hough Transform is to identify points in an image that might lie on a particular shape by mapping these points to parameter space. For instance, in the case of lines, every point in an image corresponds to a line in parameter space (typically represented as slope-intercept or Hesse normal form). When enough points align along a line, there's a peak in the parameter space, indicating the potential existence of that line in the image.

The Circular Hough Transform works similarly, but instead of lines, it's focused on identifying circles characterized by the center (α, β) and radius (r) which aligns with the circle equation:

$$(x - \alpha)^2 + (y - \beta)^2 = r^2$$

Further author information: (Send correspondence to S.J.)
S.J.: E-mail: sahil5@illinois.edu
T.H.A.: E-mail: aspirast1@udayton.edu, Telephone: +1(937)751-9000

Every edge pixel in the image is considered a potential circle center, and for each center, the algorithm votes for possible radii that could form a circle. The accumulation of votes in the parameter space signifies the presence of circles.

Both the Hough Transform and the Circular Hough Transform are robust against noise and partial occlusion to some extent, making them valuable in scenarios where traditional edge detection might struggle due to these factors.

Understanding the parametric equations of the shapes you're trying to detect is indeed crucial for these algorithms to work effectively. This knowledge helps in defining the parameter space and in setting appropriate thresholds for detection.

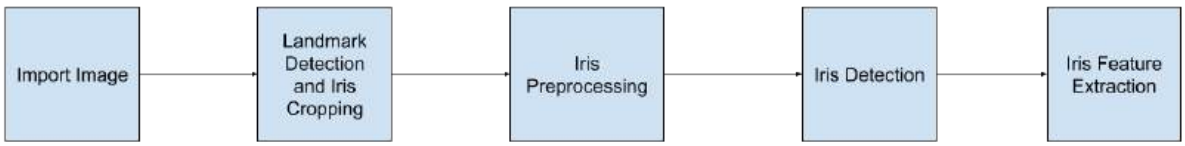
These transforms are foundational in computer vision and find applications in various domains like object detection, medical imaging, robotics, and more. Their versatility lies in their ability to detect and identify simple geometric shapes amidst complex visual data.

2.2 Iris Recognition

Iris recognition traditionally operates in the visible spectrum. However, using multispectral imaging involves capturing iris images using various wavelengths beyond the visible range, like near-infrared (NIR) or even ultraviolet (UV). The process behind this technique begins with determining the specific wavelengths or spectral bands to be captured based on the intended application. For iris imaging, near-infrared (NIR) wavelengths are often utilized due to their ability to penetrate ocular tissues and reduced sensitivity to ambient lighting. They use imaging devices equipped with sensors capable of capturing images in the chosen spectral bands. Cameras or sensors sensitive to NIR or other desired wavelengths are used to acquire images. The feature extraction and analysis comes from a deep analysis of the unique features, patterns, or characteristics specific to each spectral band. Different spectral bands might reveal distinct details or properties of the object or scene. In iris imaging, multispectral analysis can highlight specific iris textures or characteristics that are less visible in the visible spectrum. In the context of iris recognition, the multispectral image may undergo further processing specific to iris biometrics. This involves segmentation of the iris region, feature extraction from different spectral bands, and information integration for more robust and accurate recognition.

A more commonly used technique in Iris recognition is CNN Architectures. Convolutional Neural Networks (CNNs) are a class of deep learning models particularly effective in image-related tasks, including feature extraction. The core operation involves sliding small square-shaped filters (kernels) across the input image. This performs element-wise multiplication and summation, creating feature maps highlighting different input aspects. The feature extraction and analysis of this technique come from the data within each of these filters as they specialize in recognizing specific patterns or features (like edges, and textures) within the input data. Researchers use various combinations of these filters to create their training models which lead to different techniques to accomplish the same goal but more simply and/or efficiently.

3. METHODOLOGY



3.1 Iris Detection

To detect the Iris of the processed images I used an Auto Encoder Keras Model. This Auto Encoder has a U-Net architecture, designed mainly for image segmentation tasks. The Auto Encoder shown in Figure 3 consists

of an input layer and various, Convolutional Layers (Conv2D), Pooling Layers, Dropout Layers, and Batch Normalization Layers.

Convolutional Layers are used primarily in image processing, they apply a convolution operation to the input, passing the result to the next layer. It's efficient in capturing spatial hierarchies. This function takes various input parameters such as the number of filters, kernel size, activation function, padding, and strides. Keeping the kernel size, activation function, padding, and strides to a common setting of he_uniform, Rectified Linear Activation (relu), same, and (3,3) respectively. This allows me to analyze the image in great detail while retaining the necessary features and spatial information as I increase the number of filters from 32, 64, 128, 256, to 512 for the contraction path [labeled Conv2d] and 256, 128, 64, to 32 for the expansive path [labeled Conv2dTranspose].

These layers are followed by some Dropout, Pooling, and Batch Normalization layers to avoid overfitting within the model. The Dropout layers are a regularization technique where a random fraction of neurons is turned off during training to prevent overfitting. The Pooling layers, Commonly used after convolutional layers, help to reduce the spatial dimensions of the input, reducing computation and controlling overfitting. Finally, the Batch Normalization layers help in normalizing the input of every layer, making the network faster and reducing the chance of overfitting.

Shown here is the result of the auto-encoder for picture 6 [bottom left image] of the train data in Figure 15 and picture 5 of the test data [rightmost image] in Figure 16.

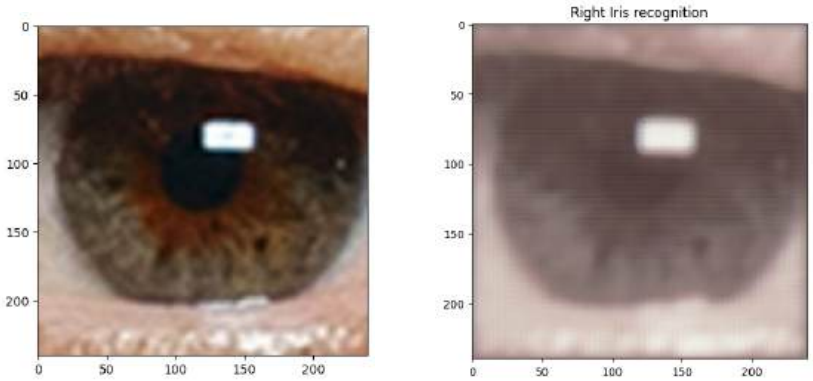


Figure 1: Model Results for training picture 6

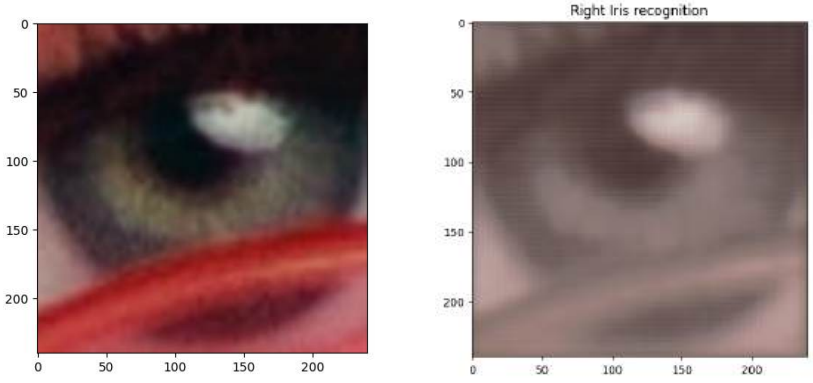
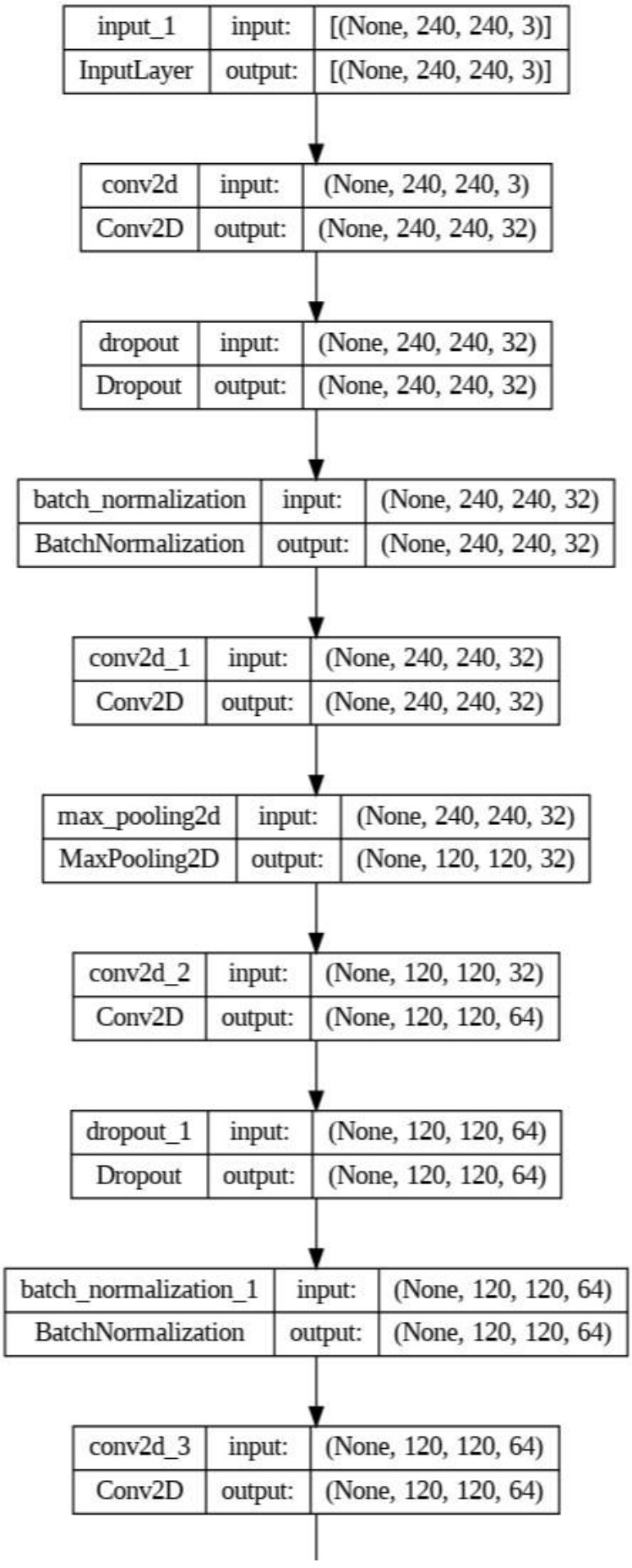
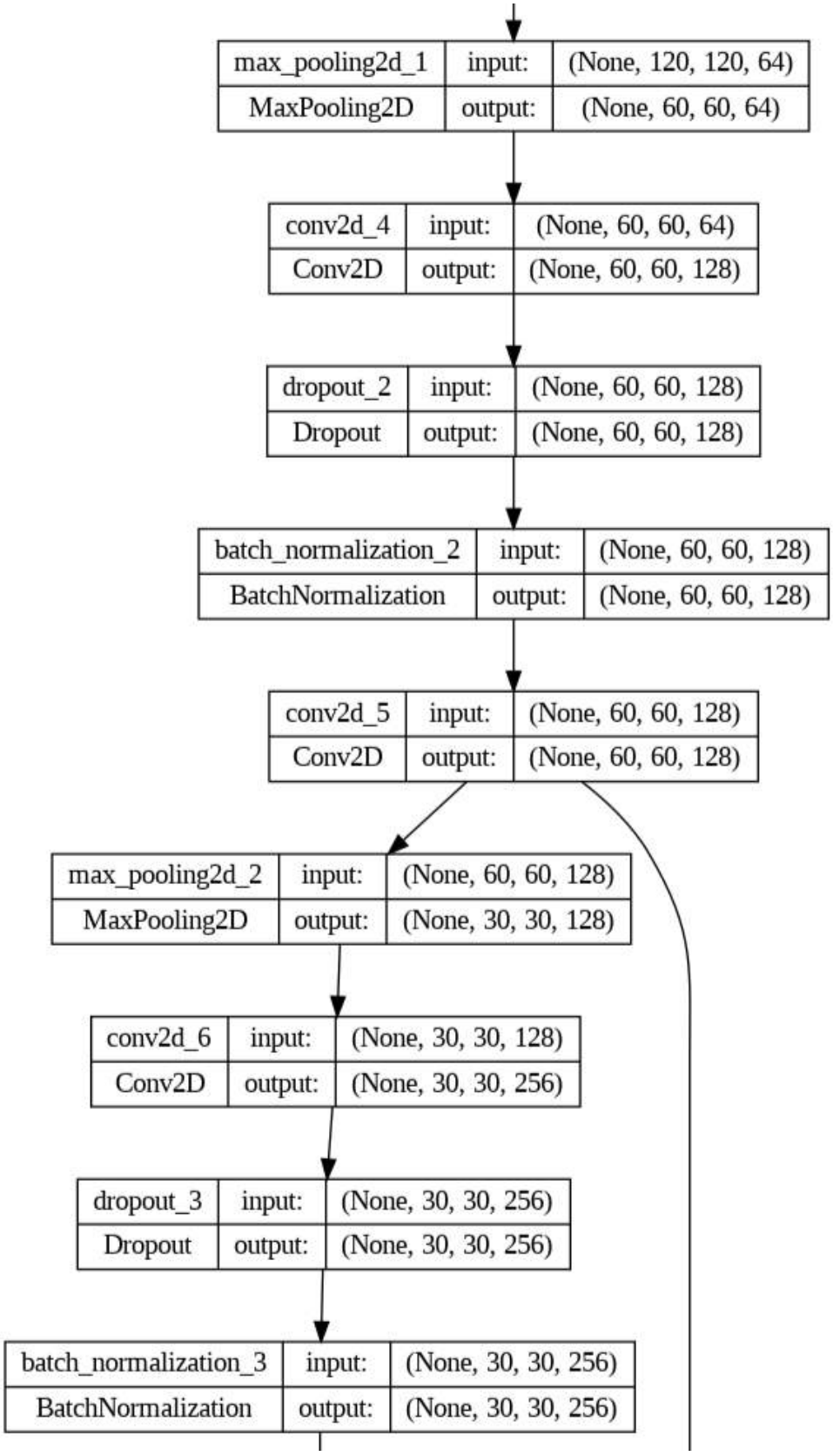
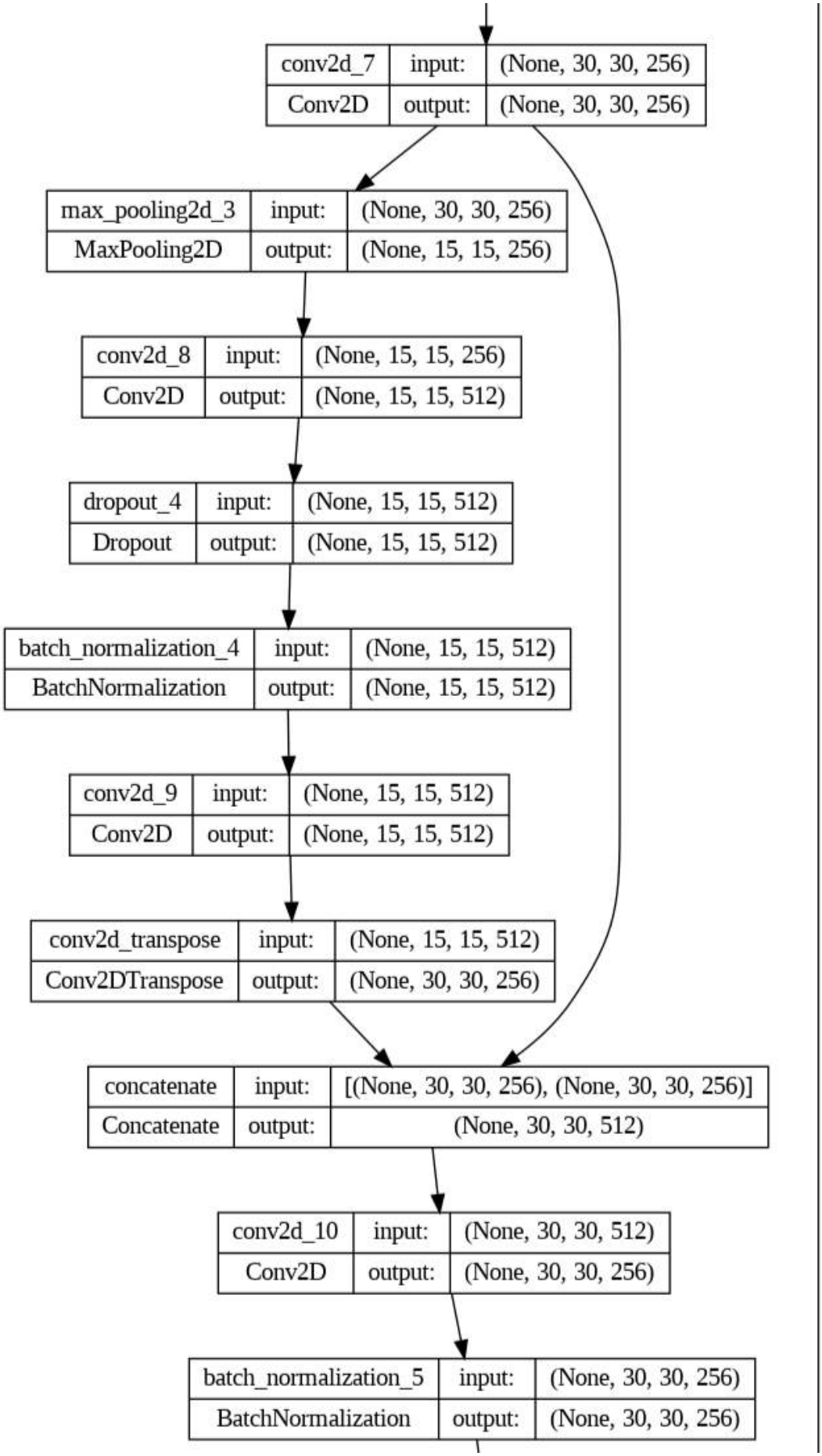
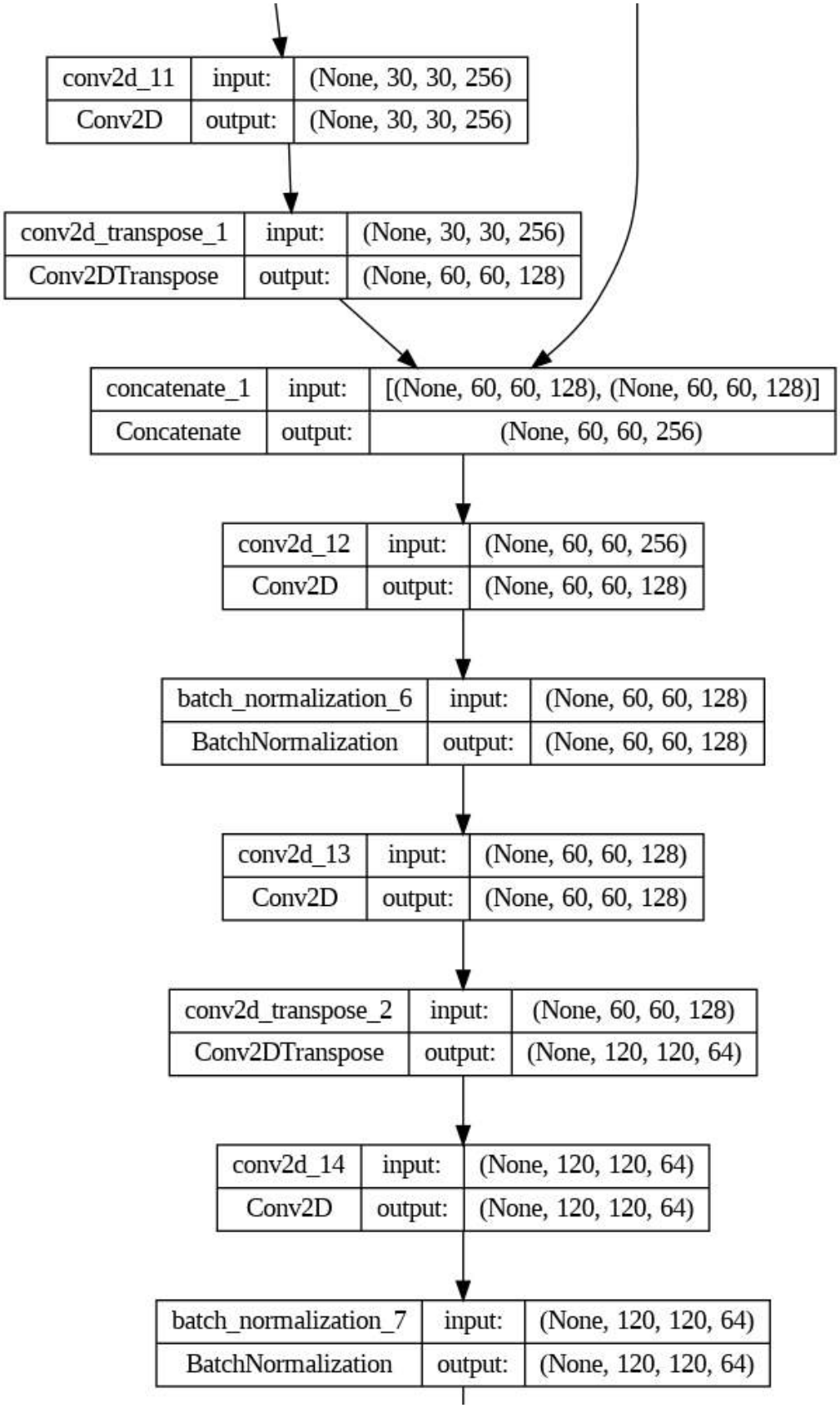


Figure 2: Model Results for testing picture 5


input_1 | input: | [(None, 240, 240, 3)]
InputLayer | output: | [(None, 240, 240, 3)]
conv2d | input: | (None, 240, 240, 3)
Conv2D | output: | (None, 240, 240, 32)
dropout | input: | (None, 240, 240, 32)
Dropout | output: | (None, 240, 240, 32)
batch_normalization | input: | (None, 240, 240, 32)
BatchNormalization | output: | (None, 240, 240, 32)
conv2d_1 | input: | (None, 240, 240, 32)
Conv2D | output: | (None, 240, 240, 32)
max_pooling2d | input: | (None, 240, 240, 32)
MaxPooling2D | output: | (None, 120, 120, 32)
conv2d_2 | input: | (None, 120, 120, 32)
Conv2D | output: | (None, 120, 120, 64)
dropout_1 | input: | (None, 120, 120, 64)
Dropout | output: | (None, 120, 120, 64)
batch_normalization_1 | input: | (None, 120, 120, 64)
BatchNormalization | output: | (None, 120, 120, 64)
conv2d_3 | input: | (None, 120, 120, 64)
Conv2D | output: | (None, 120, 120, 64)


max_pooling2d_1 | input: | (None, 120, 120, 64)
MaxPooling2D | output: | (None, 60, 60, 64)
conv2d_4 | input: | (None, 60, 60, 64)
Conv2D | output: | (None, 60, 60, 128)
dropout_2 | input: | (None, 60, 60, 128)
Dropout | output: | (None, 60, 60, 128)
batch_normalization_2 | input: | (None, 60, 60, 128)
BatchNormalization | output: | (None, 60, 60, 128)
conv2d_5 | input: | (None, 60, 60, 128)
Conv2D | output: | (None, 60, 60, 128)
max_pooling2d_2 | input: | (None, 60, 60, 128)
MaxPooling2D | output: | (None, 30, 30, 128)
conv2d_6 | input: | (None, 30, 30, 128)
Conv2D | output: | (None, 30, 30, 256)
dropout_3 | input: | (None, 30, 30, 256)
Dropout | output: | (None, 30, 30, 256)
batch_normalization_3 | input: | (None, 30, 30, 256)
BatchNormalization | output: | (None, 30, 30, 256)


conv2d_7 | input: | (None, 30, 30, 256)
Conv2D | output: | (None, 30, 30, 256)
max_pooling2d_3 | input: | (None, 30, 30, 256)
MaxPooling2D | output: | (None, 15, 15, 256)
conv2d_8 | input: | (None, 15, 15, 256)
Conv2D | output: | (None, 15, 15, 512)
dropout_4 | input: | (None, 15, 15, 512)
Dropout | output: | (None, 15, 15, 512)
batch_normalization_4 | input: | (None, 15, 15, 512)
BatchNormalization | output: | (None, 15, 15, 512)
conv2d_9 | input: | (None, 15, 15, 512)
Conv2D | output: | (None, 15, 15, 512)
conv2d_transpose | input: | (None, 15, 15, 512)
Conv2DTranspose | output: | (None, 30, 30, 256)
concatenate | input: | [(None, 30, 30, 256), (None, 30, 30, 256)]
Concatenate | output: | (None, 30, 30, 512)
conv2d_10 | input: | (None, 30, 30, 512)
Conv2D | output: | (None, 30, 30, 256)
batch_normalization_5 | input: | (None, 30, 30, 256)
BatchNormalization | output: | (None, 30, 30, 256)


conv2d_11 | input: | (None, 30, 30, 256)
Conv2D | output: | (None, 30, 30, 256)
conv2d_transpose_1 | input: | (None, 30, 30, 256)
Conv2DTranspose | output: | (None, 60, 60, 128)
concatenate_1 | input: | [(None, 60, 60, 128), (None, 60, 60, 128)]
Concatenate | output: | (None, 60, 60, 256)
conv2d_12 | input: | (None, 60, 60, 256)
Conv2D | output: | (None, 60, 60, 128)
batch_normalization_6 | input: | (None, 60, 60, 128)
BatchNormalization | output: | (None, 60, 60, 128)
conv2d_13 | input: | (None, 60, 60, 128)
Conv2D | output: | (None, 60, 60, 128)
conv2d_transpose_2 | input: | (None, 60, 60, 128)
Conv2DTranspose | output: | (None, 120, 120, 64)
conv2d_14 | input: | (None, 120, 120, 64)
Conv2D | output: | (None, 120, 120, 64)
batch_normalization_7 | input: | (None, 120, 120, 64)
BatchNormalization | output: | (None, 120, 120, 64)

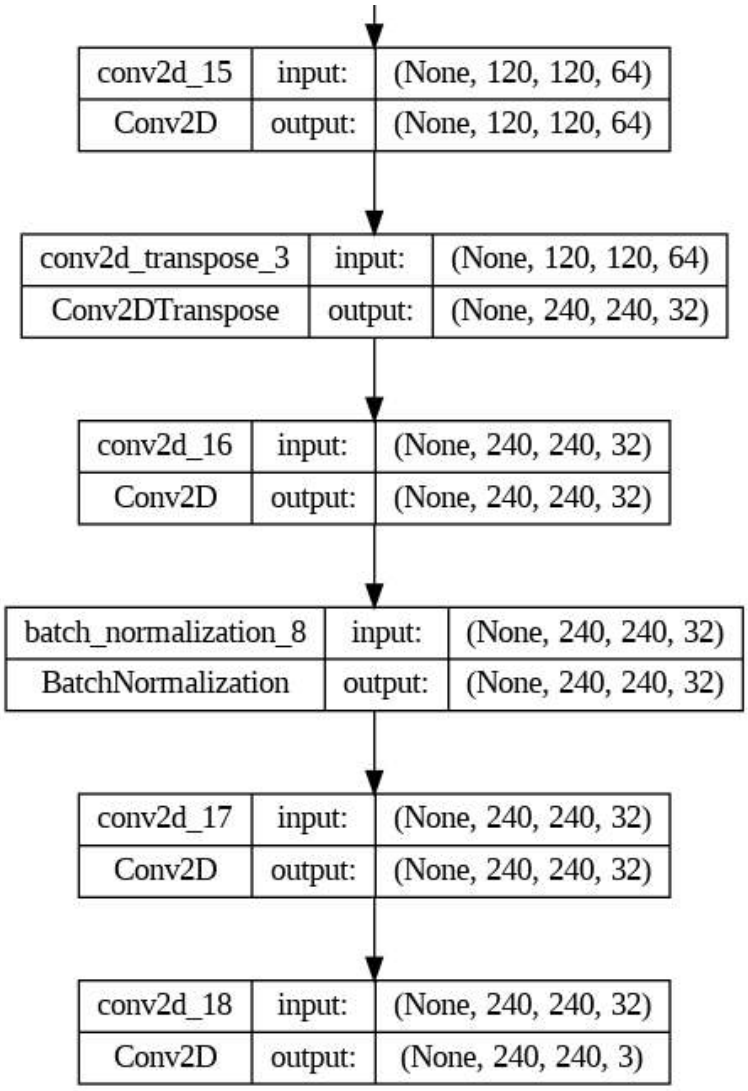


Figure 3: My Neural Network Model: Auto-Encoder

3.2 Landmark Detection and Iris Cropping

For landmark detection and cropping of the iris from the portrait image, I used extract_eye. I created two identical copies of this function, one for each eye. In this section I will describe the specifics of the extract_eye_right, the extraction function for the right eye. This function takes one input: portrait image which represents the file path to the image that is stored in my Google Drive.

3.2.1 Initilization

The function begins by setting up the required variables such as the indices of points related to the right eye and iris along with reading the image data of the portrait in the given path.

```python
def extract_eye_right(portrait_image):

  mp_face_mesh = mp.solutions.face_mesh
  # right eyes indices
  RIGHT_EYE =[ 362, 382, 381, 380, 374, 373, 390, 249, 263, 466, 388, 387, 386, 385,384, 398 ]
  # irises Indices list
  RIGHT_IRIS = [474, 475, 476, 477]

  img = cv2.imread(portrait_image)
```

Figure 4: Function Initilization

3.2.2 FaceMesh Processing

Once these variables have been loaded we can begin setting up our coordinate system on the imported image. We did this by utilizing MediaPipe's Face Mesh model to detect facial landmarks in the image, focusing on the right eye. Once we have detected the proper landmarks we can construct a mask and extract the coordinates for specific landmarks, especially those corresponding to the right eye's iris, and store them within an array.

```python
with mp_face_mesh.FaceMesh(
      max_num_faces = 1,
      refine_landmarks = True,
      min_detection_confidence = 0.5,
      min_tracking_confidence = 0.5
  ) as face_mesh:

  rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  img_h, img_w = img.shape[:2]
  results = face_mesh.process(rgb_img)
  mask = np.zeros((img_h, img_w), dtype=np.uint8)

  if results.multi_face_landmarks:
    mesh_points=np.array([np.multiply([p.x, p.y], [img_w, img_h]).astype(int)
    for p in results.multi_face_landmarks[0].landmark])

    # Mesh points is an array of all coordinates on the face
    (r_cx, r_cy), r_radius = cv2.minEnclosingCircle(mesh_points[RIGHT_IRIS])

    center_right = np.array([r_cx, r_cy], dtype=np.int32)
```

Figure 5: FaceMesh Processing

3.2.3 Drawing and Masking

Once we have established our coordinate system for the face we can use the corresponding indices of the array to draw circles and rectangles around the detected right eye and its iris for visualization purposes along with generating a mask with the area of the right eye marked.

```
# Draw right iris location on image
cv2.circle(img, center_right, int(r_radius), (0,255,0), 2, cv2.LINE_AA)

# Coordinates of top left and bottom right of cropping area
r_left = center_right[0] - int(r_radius)
r_right = center_right[0] + int(r_radius)
r_top = center_right[1] + int(r_radius)
r_bottom = center_right[1] - int(r_radius)

# Showing the region of intrest to crop
cv2.rectangle(img, (r_left, r_top), (r_right, r_bottom), (0,0,255), 3)

# drawing on the mask
cv2.circle(mask, center_right, int(r_radius), (255,255,255), -1, cv2.LINE_AA)
```

Figure 6: Drawing and Masking

3.2.4 Image Manipulation

Once we have got our traces for the region of interest (ROI) we can crop the region of interest, but before we can do that we need the modified image and mask back to RGB color space from BGR (used by OpenCV) for compatibility with other libraries.

```
# Change the colors back
RGB_edited_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
RGB_edited_mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)
```

Figure 7: Image Manipulation

3.2.5 Cropping

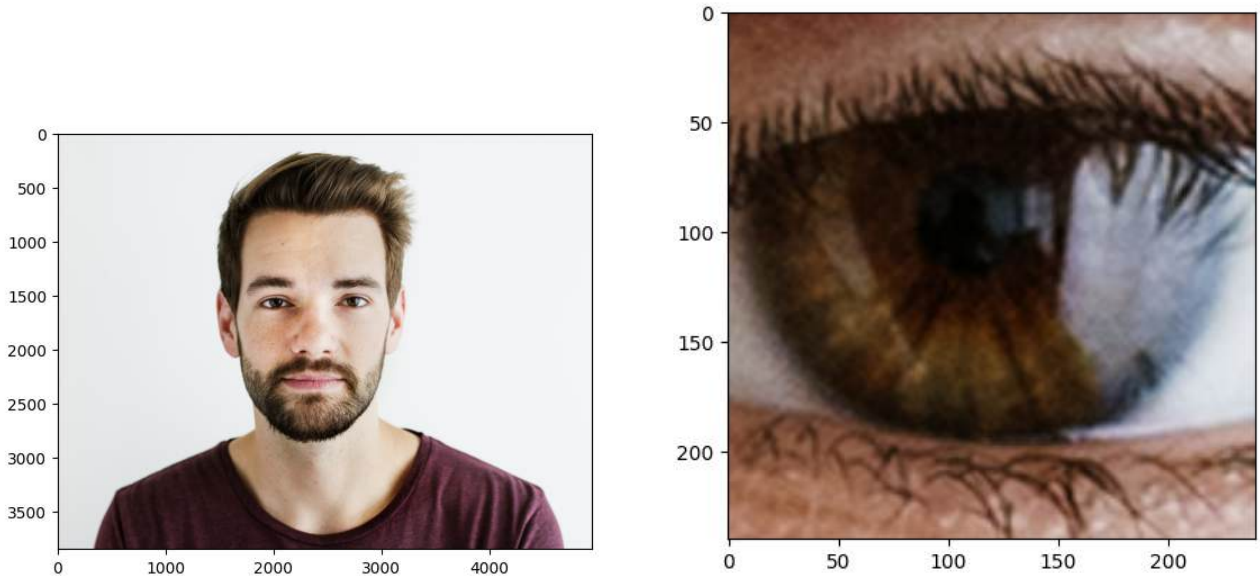
Now we can finally crop the region around the right eye from the original image using the calculated coordinates. Once we have our cropped eye we resized the result to a standard size of 240 X 240 pixels to allow compatibility with our neural network.

```
# Cropping the eyes
img_copy = Image.open(portrait_image)

# Getting each individual eye
cropped_right = img_copy.crop((r_left, r_bottom, r_right, r_top))

# resizing the cropped images
cropped_right = cropped_right.resize((240,240))
```

Figure 8: Cropping

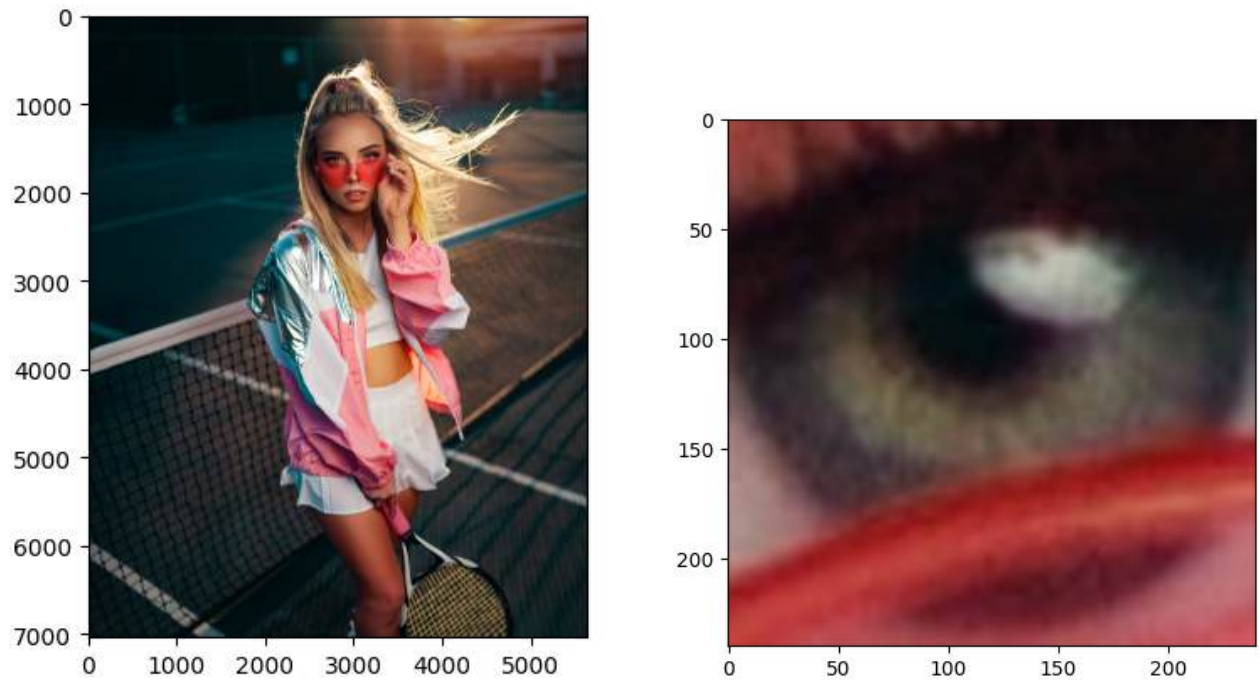
3.2.6 Return

Finally, we returned the result of the function: a cropped eye in the form of a tight rectangle around the eye region. The results of the function [extract_eye_right] are shown below.



(a) function input (b) function output

Figure 9: input and output of function



(a) function input (b) function output

Figure 10: input and output of function

3.3 Iris Preprocessing

After calling the extract_eye function for each eye, the cropped images for testing and training needed to be processed before being usable in the model. This involved converting the cropped image into a numpy array

and then appending that array to a list that can go into the model for training or testing. This array for each cropped image has three elements: height, width, and a digit (1 or 3) to represent RGB or Black and White in this order. This is what the print statement means. It serves as a check that the picture data that goes into the model is a 240 X 240 Image in Color or the array, (240, 240, 3).

```
# Storing the eyes for training
train_eyes = []
for path in portrait_path_list:

  right_eye = extract_eye_right(path)
  normalized_right_eye = np.asarray(right_eye)
  print('the size of this right eye is', normalized_right_eye.shape)
  train_eyes.append(normalized_right_eye)

  left_eye = extract_eye_left(path)
  normalized_left_eye = np.asarray(left_eye)
  print('the size of this left eye is', normalized_left_eye.shape)
  train_eyes.append(normalized_left_eye)
```

Figure 11: Iris Preprocessing

3.4 Iris Feature Extraction

To extract the features of the Iris I retrieved the output of the conv2d_9 layer of the model in Figure 3. I chose this because this is the point at which the Contraction Path ended before entering the Expansive Path. At this point, the image of the iris has been condensed so far that only the most important features have been identified and stored as a feature map of numbers from 0 to 1. This is shown by how the shape of the output of the training data at this point is (20, 15, 15, 512), representative of 20 images, each having 15 x 15 feature maps with 512 channels. This also matches the (10, 15, 15, 512) shape for the testing data as there were 5 portraits and we stored two eyes from each.

```
[[0.         0.         0.         ... 0.6455171  0.05434339 0.        ]
 [0.         0.         0.12802789 ... 0.19668034 0.15848193 0.        ]
 [0.         0.         0.         ... 0.         0.1615931  0.        ]
 ...
 [0.         0.         0.         ... 0.7406035  0.02952107 0.        ]
 [0.         0.         0.         ... 0.         0.14503658 0.        ]
 [0.         0.         0.         ... 0.         0.08138174 0.        ]]
```

Figure 12: Features for the training data

```
[[0.        0.        0.        ... 0.        0.13275217 0.       ]
 [0.        0.        0.        ... 0.12611724 0.15604202 0.       ]
 [0.        0.        0.        ... 0.23045583 0.08149906 0.       ]
 ...
 [0.        0.        0.        ... 0.7384189  0.03886301 0.       ]
 [0.        0.        0.        ... 0.5746252  0.02452056 0.       ]
 [0.        0.        0.        ... 0.5468282  0.01108078 0.       ]]
```

Figure 13: Features for the test data

3.5 Iris Feature Matching

As described before, I was able to assign each image a collection of numbers between one and zero representative of the feature map, but storing this much data for the image is very inefficient when it comes to storing the data of hundreds of images. This is why I fit these features to a PCA chart [more on this later] to assign them a simple x and y value representative of two components to simplify the data. By assigning each iris of each portrait a unique x and y I am easily able to identify an iris while maximizing the amount of eyes I can store.

```
[[ -62.176018  -107.22354  ]
 [ -19.139055   -97.48616  ]
 [ -38.9694     117.865135 ]
 [ -42.0112     118.3501   ]
 [ -54.17222   -114.49259  ]
 [ -70.08836    -59.895847 ]
 [ -60.24565      5.473821 ]
 [ -49.952396    -4.411939 ]
 [  -3.1249075   69.140594 ]
 [ -43.42824     61.32567  ]
 [ 149.70047      8.401794 ]
 [ 122.52612      6.305813 ]
 [ -65.57484    -42.544117 ]
 [ -46.43943    -49.03231  ]
 [ -39.289288    31.62102  ]
 [ -27.160482    17.559647 ]
 [  40.634647    25.382553 ]
 [ -29.479359    71.67493  ]
 [ 191.88567    -47.949417 ]
 [ 146.50395    -10.065212 ]]
```

(a) Training PCA Data

```
[[ 41.81895   124.11696  ]
 [ 16.706783   80.49579  ]
 [ 96.33339   -64.09605  ]
 [ 81.41267   -63.65536  ]
 [ 58.437862  -15.669622 ]
 [ 12.102487    9.234486 ]
 [-67.50743    -5.1700897]
 [-78.49732    -7.29804  ]
 [-78.05335   -19.690163 ]
 [-82.75402   -38.267994 ]]
```

(b) Testing PCA Data

Figure 14: PCA Data

4. RESULTS

4.1 Dataset

For my dataset, I decided to select my images from the internet. This is because I can tailor the dataset to suit my specific needs and objectives. This permits me to collect data that directly aligns with the problem I am trying to solve. Choosing my dataset also helps me have more control over its quality. I can ensure accuracy, relevance, and completeness, which is crucial for the success of many machine learning and data analysis tasks.

From an ethical standpoint, existing datasets might contain biases or sensitive information. By creating my dataset, I have more control over the ethical considerations, ensuring the data is collected ethically and responsibly.

Creating a new dataset can lead to innovative discoveries and insights. You might find patterns or correlations that were previously unknown simply because you've collected and analyzed data uniquely. For individuals learning data science or machine learning, creating a dataset provides a hands-on learning experience. It involves understanding data collection processes, data cleaning, and preprocessing, which are essential skills in the field.

The Citations for the images can be found in the following entries in the bibliography:[2–16]

Figure 15: Train Data

Figure 16: Test Data

4.2 Recognition Performance

Principle Component Analysis (PCA) is a statistical technique used to simplify data by reducing its dimensionality while retaining trends and patterns. A PCA chart, often referred to as a PCA plot or PCA graph, visually represents this reduced-dimensional data.

In a PCA chart, each data point is represented in a new coordinate system formed by the principal components (derived from the original variables). Principal components are linear combinations of the original variables that capture the maximum variance in the data.

The chart typically displays the data points in a scatter plot where the axes represent the principal components. It helps visualize clusters, patterns, or groupings that might exist in the data by showing how the data points relate to each other in this new reduced-dimensional space.

In addition to making identifying an iris easy, PCA anlysis was helpful because a neural network might produce output with a high number of dimensions, which can be difficult to interpret or visualize. PCA can reduce these dimensions while preserving the most significant variations in the output, making it easier to comprehend. Shown below is the PCA chart I constructed for my test data and train data.

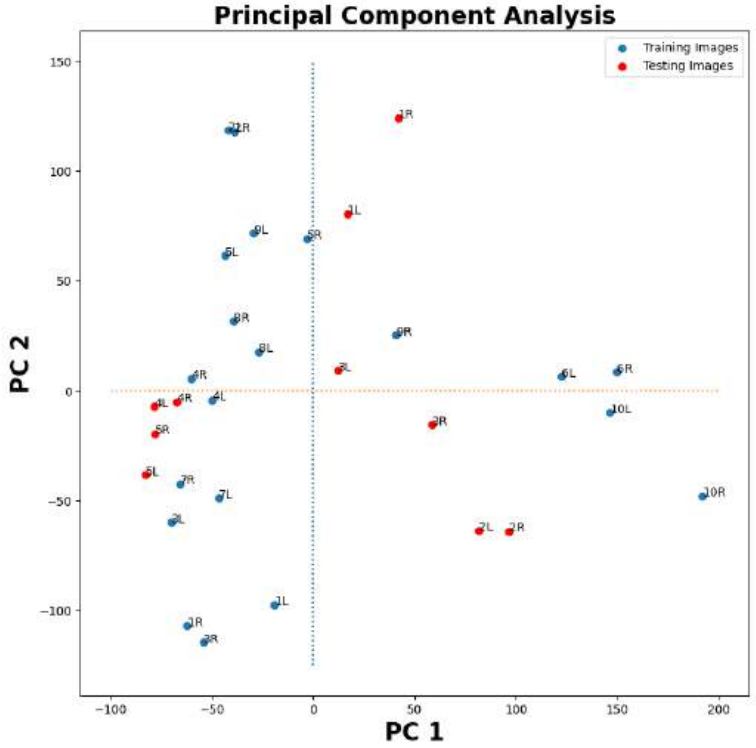


Figure 17: PCA chart for Auto Encoder

5. DISCUSSION

We expect that the iris features being plotted should be around the same for the left and right eye, which holds for points such as picture 6 in the training images [Figure 1] or picture 5 in the testing images [Figure 2], but for images such as picture 5 in the training images [Figure 9] they are more spread out.

The reason for this might be the idea of noise being present in the cropped eye since eyelashes or glare might lead to differing pixel data and hence different features being read.

It is also apparent that the data points are split into four regions, positive, negative, and zero. This shows how close the feature data matches that of the original picture. We can see that picture 4 in the testing [Figure 16] and training [Figure 15] data sets are very close to zero for Principle component 2 (PC 2), but are very negative in PC1. This variation of the points most likely comes from the issue that comes with the recreation of the image in the expansive path [this is the reason why it is there].

6. CONCLUSION

The U.S. military initially used iris scanning technology in Iraq and Afghanistan to identify detainees, utilizing devices like SEEK II for iris, fingerprint, and face scans. This technology, meant for foreign battlefields, has made its way to various U.S. police departments.[17]

New York City's Police Department was an early adopter, employing BI2 Technologies' MORIS system in 2010. Although voluntary, there have been reports of individuals facing prolonged detention for refusing iris photographs. Prisons, like Rhode Island's Department of Corrections, have also incorporated this technology. A 2015 EFF survey highlighted plans by sheriff's offices in Orange County and Los Angeles County to implement iris scanning.[17]

Currently, iris recognition devices are being set up in sheriff's departments along the U.S.-Mexico border. BI2 Technologies provided free trials of stationary iris capture devices, with plans for mobile versions. These devices generate iris templates for comparison within 20 seconds against a vast database housing nearly a million iris scans from over 180 law enforcement jurisdictions, managed by BI2 in undisclosed locations.[17]

The database, managed by a third-party vendor in San Antonio, Texas, and backup facilities, is touted as the largest of its kind in North America, as stated by a BI2 executive to The Intercept.[17]

The basic technique of using landmark detection iris preprocessing and feature extraction presented in this paper will be greatly useful in making these iris scans more efficient as you don't need to be in the uncomfortable position of putting your eye close to a camera. However, further developments need to be made to perfect the feature extraction before releasing it for use in law enforcement.[17]

ACKNOWLEDGMENTS

This unnumbered section is used to identify those who have aided the authors in understanding or accomplishing the work presented and to acknowledge sources of funding.

REFERENCES

[1] Randhawa, A. S., Radheya, A., and Manikantan, K., "Segmentation based background removal technique for enhanced iris recognition," in [*2015 IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*], 38–43 (2015).
[2] Bob McNeely, The White House, "Official White House photo of President Bill Clinton, President of the United States.," (1993). [Online; accessed January 3, 2024].
[3] Andy Gotts, "Andy Gotts Photographer," (2024). [Online; accessed January 3, 2024].
[4] Andrew Personal Training, Pexels, "Man Wearing Orange Nike Crew-neck T-shirt," (2017). [Online; accessed January 3, 2024].
[5] Andrea Piacquadio, Pexels, "Strong confident sportsman holding kettlebell in hand while training in gym and looking at camera," (2020). [Online; accessed January 17, 2024].
[6] KEREM KSLR, Pexels, "Portrait of a Man Standing Outdoors," (N/A). [Online; accessed January 17, 2024].
[7] Roy Reyna, Pexels, "Photo Of Woman Holding Tennis Racquet," (2019). [Online; accessed January 17, 2024].
[8] Đặng Nhật, Pexels, "A Woman in Red Long Sleeve Dress," (2022). [Online; accessed January 17, 2024].
[9] Mateus Souza, Pexels, "Woman Wearing White Shirt With White Flower on Her Ear," (2018). [Online; accessed January 17, 2024].
[10] Tuân Kiệt Jr., Pexels, "Woman Standing and Doing Pose Beside Lake," (2018). [Online; accessed January 17, 2024].
[11] NL, Pexels, "Man Wearing Black Framed Eyeglasses," (2017). [Online; accessed January 17, 2024].
[12] Pixabay, Pexels, "Woman Wearing Black Spaghetti Strap Top," (2016). [Online; accessed January 17, 2024].
[13] rawpixel.com, Freepik, "Portrait of white man isolated," (N/A). [Online; accessed January 17, 2024].
[14] rawpixel.com, Freepik, "African man, successful entrepreneur wearing glasses face portrait," (N/A). [Online; accessed January 19, 2024].

[15] cookie_studio, Freepik, "Portrait of young pretty positive girl smiling," (N/A). [Online; accessed January 19, 2024].

[16] cookie_studio, Freepik, "Close up shot pretty woman with perfect teeth dark clean skin having rest indoors smiling happily after received good positive news," (N/A). [Online; accessed January 19, 2024].

[17] Electronic Frontier Foundation, "Iris Recognition," (2019). [Online; accessed January 3, 2024].

[18] Alred, G. J., Brusaw, C. T., and Oliu, W. E., [*Handbook of Technical Writing*], St. Martin's, New York (2015 (eleventh edition)).

[19] Goossens, M., Mittelbach, F., and Rahtz, S., [*The LaTeX Companion*], Addison-Wesley, Reading, Mass. (1997).

[20] Mittelbach, F., Goossens, M., Braams, J., and Carlisle, D., [*The LaTeX Companion*], Addison-Wesley, Reading, Mass., second ed. (2004).

[21] Gull, S. F., "Developments in maximum-entropy data analysis," in [*Maximum Entropy and Bayesian Methods*], Skilling, J., ed., 53–71, Kluwer Academic, Dordrecht (1989).

[22] Hanson, K. M., "Introduction to Bayesian image analysis," in [*Medical Imaging: Image Processing*], Loew, M. H., ed., *Proc. SPIE* **1898**, 716–731 (1993).

[23] Lamport, L., [*LaTeX: A Document Preparation System*], Addison-Wesley, Reading, Mass. (1994).

[24] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E., "Equations of state calculations by fast computing machine," *J. Chem. Phys.* **21**, 1087–1091 (1953).

[25] Perelman, L. C., Paradis, J., and Barrett, E., [*Mayfield Handbook of Technical and Scientific Writing*], Mountain View, Mayfield (April 1997). http://mit.imoat.net/handbook/.

[26] Lees-Miller, J. D., "Free and Interactive Online Introduction to LaTeX." Overleaf, 26 February 2015 https://www.overleaf.com/latex/learn/free-online-introduction-to-latex-part-1. (Accessed: 15 September 2015).

[27] Sager, C., Zschech, P., and Kühl, N., "labelcloud: A lightweight domain-independent labeling tool for 3d object detection in point clouds," (2021).

[28] Ibrahim, M., Akhtar, N., Wise, M., and Mian, A., "Annotation tool and urban dataset for 3d point cloud semantic segmentation," *IEEE Access* **9**, 35984–35996 (2021).

[29] Meng, Q., Wang, W., Zhou, T., Shen, J., Jia, Y., and Van Gool, L., "Towards a weakly supervised framework for 3d point cloud object detection and annotation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[30] Wang, B., Wu, V., Wu, B., and Keutzer, K., "Latte: accelerating lidar point cloud annotation via sensor fusion, one-click annotation, and tracking," in [*2019 IEEE Intelligent Transportation Systems Conference (ITSC)*], 265–272, IEEE (2019).

[31] Chen, Z., Zeng, W., Yang, Z., Yu, L., Fu, C.-W., and Qu, H., "Lassonet: Deep lasso-selection of 3d point clouds," *IEEE transactions on visualization and computer graphics* **26**(1), 195–204 (2019).

[32] Qi, C. R., Su, H., Mo, K., and Guibas, L. J., "Pointnet: Deep learning on point sets for 3d classification and segmentation," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 652–660 (2017).

[33] Varney, N., Asari, V. K., and Graehling, Q., "Dales: A large-scale aerial lidar data set for semantic segmentation," in [*Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*], 186–187 (2020).

[34] Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J., "Kpconv: Flexible and deformable convolution for point clouds," in [*Proceedings of the IEEE/CVF international conference on computer vision*], 6411–6420 (2019).

[35] Varney, N., Asari, V. K., and Graehling, Q., "Pyramid point: A multi-level focusing network for revisiting feature layers," *arXiv preprint arXiv:2011.08692* (2020).