Fish Detection and Key Point Extraction using Deep Learning

**Introduction**

This document provides a comprehensive guide for the implementation and usage of the fish detection and key point extraction algorithm. The algorithm is designed to identify fully or partially visible fish in images and extract key points such as the nose, top fin, and tail. The source code is written in Python and leverages PyTorch for deep learning model inference and OpenCV for image processing tasks.

**Algorithm Design**

- \*\*Model Architecture:\*\* The model used is a convolutional neural network (CNN) with four convolutional layers followed by two fully connected layers. The CNN is trained to classify images into two categories: fully visible and partially visible.  
- \*\*Key Point Detection:\*\* Key points such as the nose, top fin, and tail are detected using contour analysis and geometric transformations.

**Implementation Steps**

1) Determine if the input image is fully visible or partially visible.

1. \*\*Directory Setup:\*\* Define directories for input images and output results.

-----CODE HERE-----

# Define directories

base\_dir = 'mask'

output\_dir = 'output'  
2. \*\*Model Definition:\*\* Define the CNN architecture using PyTorch.

-----CODE HERE-----

# Define the image classifier model

class ImageClassifier(nn.Module):

  def \_\_init\_\_(self):

    super(ImageClassifier, self).\_\_init\_\_()

    self.conv1 = nn.Conv2d(3, 32, kernel\_size=3, padding=1)

    self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)

    self.conv3 = nn.Conv2d(64, 96, kernel\_size=3, padding=1)

    self.conv4 = nn.Conv2d(96, 128, kernel\_size=3, padding=1)

    self.pool = nn.MaxPool2d(kernel\_size=2, stride=2, padding=0)

    self.dropout = nn.Dropout(0.5)

    self.fc1 = nn.Linear(128 \* 16 \* 16, 512)

    self.fc2 = nn.Linear(512, 2) # 2 classes: fully visible or partially visible

  def forward(self, x):

    x = self.pool(F.relu(self.conv1(x)))

    x = self.pool(F.relu(self.conv2(x)))

    x = self.pool(F.relu(self.conv3(x)))

    x = self.pool(F.relu(self.conv4(x)))

    x = x.view(-1, 128 \* 16 \* 16) # Adjusted size after pooling

    x = self.dropout(F.relu(self.fc1(x)))

    x = self.fc2(x)

    return x  
3. \*\*Load Pre-trained Model:\*\* Load the pre-trained model parameters from a file.

-----CODE HERE-----

# Load the saved model parameters

model\_path = 'model.pth'

model = ImageClassifier().to(device)

model.load\_state\_dict(torch.load(model\_path))

model.eval()  # Set the model to evaluation mode  
4. \*\*Image Preprocessing:\*\* Preprocess input images using OpenCV and PyTorch transforms.

-----CODE HERE-----

# Define the transformation

transform = transforms.Compose([

    transforms.Resize((256, 256)),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),

])

# Function to preprocess the image

def preprocess\_image(image):

# If `image` is a PIL image, convert it to a NumPy array

if isinstance(image, Image.Image):

image = np.array(image)

# Convert to grayscale if the image is not already

if len(image.shape) == 3:

binary = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

padding = image.shape[0] // 100 \* 5

# Apply binary thresholding

\_, binary = cv2.threshold(binary, 127, 255, cv2.THRESH\_BINARY)

# Detect contours

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

largest\_contour = max(contours, key=cv2.contourArea)

x, y, w, h = cv2.boundingRect(largest\_contour)

if w >= h:

size = w

else:

size = h

roi = np.zeros((size + padding \* 2, size + padding \* 2, 3), dtype=np.uint8)

x\_offset = (roi.shape[1] - w) // 2

y\_offset = (roi.shape[0] - h) // 2

roi[y\_offset:y\_offset+h, x\_offset:x\_offset+w] = image[y:y+h, x:x+w]

else:

roi = image

# Convert NumPy array to PIL image

roi = Image.fromarray(roi)

# Convert grayscale to RGB

roi = roi.convert('RGB')

# Apply transformations

roi = transform(roi)

# Add batch dimension

roi = roi.unsqueeze(0)

# Move to the appropriate device

return roi.to(device\_gpu)  
5. \*\*Model Inference:\*\* Use the CNN to classify images.

-----CODE HERE-----

# Function to predict the class of an image

def predict\_image(image\_path, model, class\_names):

    image = preprocess\_image(image\_path)

    with torch.no\_grad():  # Disable gradient calculation

        outputs = model(image)

        \_, predicted = torch.max(outputs, 1)

    return class\_names[predicted.item()]  
2) When the input image is fully visible, extract key points.

1. \*\*Image Rotation:\*\* Rotate the binary images to a horizontal position based on the fitted ellipse.

-----CODE HERE-----

# Function to rotate the image to a horizontal position

def horizontal\_state\_tuning(image):

    contours = detect\_contours(image)

    contour = max(contours, key=cv2.contourArea)

    # Fit an ellipse to the contour

    ellipse = cv2.fitEllipse(contour)

    # Get the angle of the major axis of the ellipse

    angle = ellipse[2]

    # Get the center of the image

    image\_center = (image.shape[1] // 2, image.shape[0] // 2)

    # Calculate the rotation matrix

    rotation\_matrix = cv2.getRotationMatrix2D(image\_center, angle - 90, 1.0)

    # Perform the rotation

    rotated\_image = cv2.warpAffine(image, rotation\_matrix, (image.shape[1], image.shape[0]), flags=cv2.INTER\_LINEAR, borderMode=cv2.BORDER\_CONSTANT, borderValue=(0, 0, 0))

    return rotated\_image

2. \*\*Noise Removal:\*\* Apply morphological operations to remove noise from rotated images.

-----CODE HERE-----

# Function to remove noise from the binary image

def remove\_noise(binary\_image):

    # Define a kernel for morphological operations

    kernel = np.ones((3, 3), np.uint8)

    # Apply morphological operations

    # Opening operation (erosion followed by dilation) to remove small noise

    cleaned\_image = cv2.morphologyEx(binary\_image, cv2.MORPH\_OPEN, kernel, iterations=2)

    # Closing operation (dilation followed by erosion) to close small holes inside objects

    cleaned\_image = cv2.morphologyEx(cleaned\_image, cv2.MORPH\_CLOSE, kernel, iterations=2)

    return cleaned\_image  
3. \*\*Key Point Extraction:\*\* Detect key points such as the nose, top fin, and tail.

-----CODE HERE-----

# Function to detect key points (nose, tail, top fin) on the fish

def detect\_key\_point(image, path):

    contours = detect\_contours(image)

    contour = max(contours, key=cv2.contourArea)

    # Find extreme points

    leftmost = tuple(contour[contour[:, :, 0].argmin()][0])

    rightmost = tuple(contour[contour[:, :, 0].argmax()][0])

    # Divide the fish into segments

    segment\_width = math.ceil((rightmost[0] - leftmost[0]) / 10 \* 4)  # Divide into 10 segments

    segment\_image = image.copy()

    segment\_image[0:segment\_image.shape[0], leftmost[0] + segment\_width:segment\_image.shape[1]] = 0

    segment\_width1 = segment\_width // 10 # Divide into 10 segments

    tail\_position = 'right'

    segment\_area = []

    segment\_id = 0

    for i in range(10):

        x1 = leftmost[0] + segment\_width - (i + 1) \* segment\_width1

        x2 = x1 + segment\_width1

        segment = segment\_image[:, min(x1, x2):max(x1, x2)]

        segment\_contours, \_ = cv2.findContours(segment, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

        if segment\_contours:

            segment\_contour = max(segment\_contours, key=cv2.contourArea)

            segment\_area.append(cv2.contourArea(segment\_contour))

        else:

            segment\_area.append(0)

    segment\_area.reverse()

    index = find\_inflection\_point(segment\_area)

    if index != -1:

        segment\_id = index

    body\_image = image.copy()

    tail\_image = image.copy()

    width = math.ceil((rightmost[0] - leftmost[0]) / 10)

    if segment\_id != 0:

        tail\_position = 'left'

        body\_image[0:body\_image.shape[0], 0:leftmost[0] + width \* 3] = 0

        tail\_image[0:tail\_image.shape[0], leftmost[0] + width \* 4:tail\_image.shape[1]] = 0

    else:

        body\_image[0:body\_image.shape[0], leftmost[0] + width \* 7:tail\_image.shape[1]] = 0

        tail\_image[0:tail\_image.shape[0], 0:leftmost[0] + width \* 6] = 0

    nose\_point, top\_fin\_point = detect\_nose\_and\_top\_fin(body\_image, tail\_position)

    tail\_point = detect\_tail\_middle(tail\_image, tail\_position)

    return nose\_point, top\_fin\_point, tail\_point

4. \*\*Mapping a point\*\* Mapping a point in the rotated image back to the original image.

-----CODE HERE-----

# Function to map a point in the rotated image back to the original image

def get\_original\_point(image, point):

    contours = detect\_contours(image)

    # Assuming the largest contour is the one we are interested in

    contour = max(contours, key=cv2.contourArea)

    # Fit an ellipse to the contour

    ellipse = cv2.fitEllipse(contour)

    # Get the angle of the major axis of the ellipse

    angle = ellipse[2]

    # Get the center of the image

    image\_center = (image.shape[1] // 2, image.shape[0] // 2)

    # Calculate the inverse rotation matrix

    inverse\_rotation\_matrix = cv2.getRotationMatrix2D(image\_center, 90 - angle, 1.0)

    # Add a third row to the rotation matrix to make it 3x3 for affine transformation

    inverse\_rotation\_matrix = np.vstack([inverse\_rotation\_matrix, [0, 0, 1]])

    # Transform the tail point back to the original image

    point\_rotated\_homogeneous = np.array([point[0], point[1], 1]).T

    point\_original = np.dot(inverse\_rotation\_matrix, point\_rotated\_homogeneous)

    point\_original = (int(point\_original[0]), int(point\_original[1]))

    return point\_original

5. \*\*Contour Detection:\*\* Detect contours in the binary images.  
-----CODE HERE-----

# Function to detect contours in the binary image

def detect\_contours(image):

  contours, \_ = cv2.findContours(image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

  return contours  
6. \*\*Intersection Calculation:\*\* Calculate the intersection points of lines with contours.

-----CODE HERE-----

# Get the line equation connecting the nose and tail

line\_eq = get\_line\_equation(original\_nose\_point, original\_tail\_point)

# Get the equation of the perpendicular line from the dorsal start point

perp\_line\_eq = get\_perpendicular\_line(original\_top\_fin\_point, line\_eq)

# Find the intersection of the perpendicular line with the contour

intersections = find\_intersections(contour, perp\_line\_eq)  
7. \*\*Visualization and Saving Results:\*\* Visualize the results and save the processed images.

-----CODE HERE-----

binary\_image = cv2.cvtColor(binary\_image, cv2.COLOR\_GRAY2BGR)

cv2.circle(binary\_image, original\_nose\_point, 3, (0, 0, 255), -1)

cv2.circle(binary\_image, (original\_tail\_point[0], original\_tail\_point[1]), 3, (255,0,0), -1)

cv2.line(binary\_image, original\_nose\_point, original\_tail\_point, (255, 0, 255), 2)

# Draw the vertical line from the known point to the intersection point on the second line

if intersections:

   if intersections.\_\_len\_\_() > 1:

    cv2.circle(binary\_image, (intersections[0][0], intersections[0][1]), 3, (0, 255, 0), -1)

     cv2.circle(binary\_image, (intersections[1][0], intersections[1][1]), 3, (0, 255, 0), -1)

     cv2.line(binary\_image, intersections[0], intersections[1], (0, 255, 255), 2)

   else:

      cv2.circle(binary\_image, original\_top\_fin\_point, 3, (0, 255, 0), -1)

      cv2.circle(binary\_image, intersections[0], 3, (0, 255, 0), -1)

      cv2.line(binary\_image, intersections[0], original\_top\_fin\_point, (0, 255, 255), 2)

cv2.imwrite(os.path.join(output\_dir, filename), binary\_image)

**Usage Instructions**

1. \*\*Install Dependencies:\*\* Ensure you have Python, PyTorch, OpenCV libraries installed.

-----CODE HERE-----

pip install torch torchvision

pip install opencv  
2. \*\*Set Up Directories:\*\* Place input images in the specified directory and create an output directory for results.  
3. \*\*Run the Script:\*\* Execute the Python script to process the images and save the results.

-----CODE HERE-----

python3 test.py

**Optimization Strategies for Jetson Xavier NX**

- \*\*Leverage GPU:\*\* Ensure CUDA and cuDNN are properly installed and configured on the Jetson Xavier NX.  
- \*\*Model Optimization:\*\* Use TensorRT to optimize the model for inference on Jetson Xavier NX.  
- \*\*Efficient Data Handling:\*\* Use efficient data loading and preprocessing techniques to maximize the performance on Jetson Xavier NX.

**Support and Troubleshooting**

- \*\*Deployment Support:\*\* Assistance will be provided for deploying the code on Jetson Xavier NX.  
- \*\*Bug Fixes:\*\* Support for troubleshooting and fixing any issues encountered during deployment.  
- \*\*Performance Enhancements:\*\* Continuous support for improving the performance of the algorithm post-deployment.

**Detailed Algorithm with Mathematical Explanation**

**Tail Detection**

\*\*Reasoning:\*\*  
The tail is typically the segment of the fish with the smallest area. This is because the tail usually tapers to a point and has less mass compared to the body.  
  
\*\*Mathematics:\*\*  
Divide the fish into segments and calculate the area of each segment. The segment with the minimum area (after some initial segments) is identified as the tail.  
  
\*\*Implementation:\*\*  
```CODE HERE```

# Function to detect the middle of the tail

def detect\_tail\_middle(image, position):

    contours = detect\_contours(image)

    contour = max(contours, key=cv2.contourArea)

    # Find extreme points

    leftmost = tuple(contour[contour[:, :, 0].argmin()][0])

    rightmost = tuple(contour[contour[:, :, 0].argmax()][0])

    # Divide the fish into segments

    segment\_width = (rightmost[0] - leftmost[0]) // 10  # Divide into 10 segments

    tail\_segment = None

    segment\_area\_min = float('inf')

    for i in range(8):  # We will have 10 segments to analyze

        if position == 'left':

            x1 = rightmost[0] - (i + 1) \* segment\_width

            x2 = x1 + segment\_width

        else:

            x1 = leftmost[0] + i \* segment\_width

            x2 = x1 + segment\_width

        segment = image[:, min(x1, x2):max(x1, x2)]

  segment\_contours, \_ = cv2.findContours(segment, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

        if segment\_contours:

            segment\_contour = max(segment\_contours, key=cv2.contourArea)

            segment\_area = cv2.contourArea(segment\_contour)

            if segment\_area < segment\_area\_min:

                segment\_area\_min = segment\_area

                tail\_segment = (min(x1, x2), max(x1, x2))

    if tail\_segment is not None:

        tail\_x\_coord = (tail\_segment[0] + tail\_segment[1]) // 2

    else:

        tail\_x\_coord = (leftmost[0] + rightmost[0]) // 2

    line\_eq = (float('inf'), tail\_x\_coord)

    tail\_coords = find\_intersections(contour, line\_eq)

    tail\_y\_coord = int((tail\_coords[0][1] + tail\_coords[1][1]) / len(tail\_coords))

    return (tail\_x\_coord, tail\_y\_coord)

**Nose Detection**

\*\*Reasoning:\*\*  
The nose of the fish is typically the foremost point of the fish along its main axis (leftmost or rightmost point depending on orientation).  
  
\*\*Mathematics:\*\*  
Find the extreme points (leftmost and rightmost) of the contour. The extreme point on the appropriate side (depending on the fish orientation) is identified as the nose.  
  
\*\*Implementation:\*\*  
```CODE HERE```

contours = detect\_contours(image)

contour = max(contours, key=cv2.contourArea)

 # Find extreme points

 leftmost = tuple(contour[contour[:, :, 0].argmin()][0])

 nose\_x\_coord = leftmost[0]

 nose\_point = find\_y\_coordinates(contour, nose\_x\_coord)[0]

**Top Fin Detection**

\*\*Reasoning:\*\*  
The top fin is typically located along the upper contour of the fish.  
  
\*\*Mathematics:\*\*  
Find the extreme points (topmost) of the contour.  
  
\*\*Implementation:\*\*  
```CODE HERE```

# Function to detect the start point of the top fin

def detect\_nose\_and\_top\_fin(image, position):

    if position ==  'left':

        image = cv2.flip(image, 1)

    contours = detect\_contours(image)

    contour = max(contours, key=cv2.contourArea)

    # Get the start point of the top fin

start\_point\_of\_top\_fin = tuple(contour[contour[:, :, 1].argmin()][0])

    if position == 'left':

        nose\_point = (image.shape[1] - nose\_point[0], nose\_point[1])

  start\_point\_of\_top\_fin = (image.shape[1] - start\_point\_of\_top\_fin[0], start\_point\_of\_top\_fin[1])

    return nose\_point, start\_point\_of\_top\_fin

**Perpendicular Point Calculation**

\*\*Reasoning:\*\*  
Find a point on the contour that is perpendicular to the line connecting the nose and the tail.  
  
\*\*Mathematics:\*\*  
Calculate the line equation for the line connecting the nose and the tail. Then, compute the perpendicular line passing through the start point of the top fin. Find the intersection of this perpendicular line with the contour.  
  
\*\*Implementation:\*\*  
```CODE HERE```

# Function to calculate the equation of the line passing through two points

def get\_line\_equation(p1, p2):

    """Calculate the slope (m) and y-intercept (b) of the line passing through points p1 and p2."""

    x1, y1 = p1

    x2, y2 = p2

    if x1 != x2:

        m = (y2 - y1) / (x2 - x1)

        b = y1 - m \* x1

    else:

        m = float('inf')

        b = y1

    return m, b

# Function to calculate the equation of the line perpendicular to a given line passing through a point

def get\_perpendicular\_line(p, line\_eq):

    """Calculate the equation of the line perpendicular to the given line passing through point p."""

    m1, \_ = line\_eq

    if m1 != 0:

        m2 = -1 / m1

        b2 = p[1] - m2 \* p[0]

        return m2, b2

    elif m1 == 0:

        # Vertical line through point p

        return float('inf'), p[0]

    else:

        return float('inf'), 0

# Function to find the intersections of a line with the contour

def find\_intersections(contour, line\_parms):

    intersections = []

    n = len(contour)

    if line\_parms[0] == float('inf'):

        # Vertical line case

        x\_intersect = line\_parms[1]

        for i in range(n - 1):

            x1, y1 = contour[i][0]

            x2, y2 = contour[i + 1][0]

            # Check if the segment crosses the vertical line

            if (x1 <= x\_intersect <= x2) or (x2 <= x\_intersect <= x1):

                # Calculate the intersection

                if x2 != x1:

                    m = (y2 - y1) / (x2 - x1)

                    c = y1 - m \* x1

                    y = m \* x\_intersect + c

                    intersections.append((int(x\_intersect), int(y)))

                else:

                    # Handle vertical segment case

                    intersections.append((int(x1), int(y1)))

    else:

        for i in range(n - 1):

            x1, y1 = contour[i][0]

            x2, y2 = contour[i + 1][0]

            # Line equation at endpoints

            y\_line1 = line\_parms[0] \* x1 + line\_parms[1]

            y\_line2 = line\_parms[0] \* x2 + line\_parms[1]

            # Check if segment crosses the line

            if (y1 >= y\_line1 and y2 <= y\_line2) or (y1 <= y\_line1 and y2 >= y\_line2):

                # Calculate the intersection

                if x2 != x1:

                    m = (y2 - y1) / (x2 - x1)

                    c = y1 - m \* x1

                    x = (line\_parms[1] - c) / (m - line\_parms[0])

                    y = line\_parms[0] \* x + line\_parms[1]

                    intersections.append((int(x), int(y)))

                else:

                    intersections.append((x1, y1))

    if not intersections:

        return []

    y\_values = [point[1] for point in intersections]

    min\_y = min(y\_values)

    max\_y = max(y\_values)

    min\_y\_point = [point for point in intersections if point[1] == min\_y][0]

    max\_y\_point = [point for point in intersections if point[1] == max\_y][0]

    return min\_y\_point, max\_y\_point

# Get the line equation connecting the nose and tail

line\_eq = get\_line\_equation(original\_nose\_point, original\_tail\_point)

# Get the equation of the perpendicular line from the dorsal start point

perp\_line\_eq = get\_perpendicular\_line(original\_top\_fin\_point, line\_eq)

# Find the intersection of the perpendicular line with the contour

intersections = find\_intersections(contour, perp\_line\_eq)