Curvetopia : Incremental Beautification using algorithms and neural networks

Suyash Ajit Chavan
whEN bhAI, IIITB

Subhajeet Lahiri
whEN bhAI, IIITB

***Abstract*—Our solution to the given problem statement is a framework designed for the incremental beautification of hand-drawn doodles using a combination of traditional algorithms and neural networks. It introduces a modular system where various filters can be applied one by one to refine and enhance the curves in a drawing. The framework includes several custom filters, such as the LineFilter for regularization and the ConvexFilter for reshaping convex curves into aesthetically pleasing forms like circles, ellipses, and polygons. Additionally, a neural network-based PixelDetector filter is employed to recognize regular patterns made up of multiple curves. Together, these components enable Curvetopia to progressively transform rough sketches into polished and visually harmonious designs.**

Link to the GitHub Repository
Link to the hosted main notebook
Link to the detector model snapshot
Link to the synthetic dataset

Design of the Incremental Algorithm

The incremental beautification process in Curvetopia is driven by a series of filters that progressively refine the curves in a doodle, with each filter designed to target specific types of irregularities or patterns. The flow of data between these filters is key to transforming rough sketches into polished, aesthetically pleasing designs. refer to Fig. 1.

Initial Regularization

The process starts with the `SmoothenFilter`, which refines all curves by smoothing irregularities using B-spline interpolation. Next, the `LineFilter` is applied to identify and straighten curves that are nearly linear through regression analysis. This sequential filtering ensures the curves are both aesthetically enhanced and regularized, preparing them for further processing.

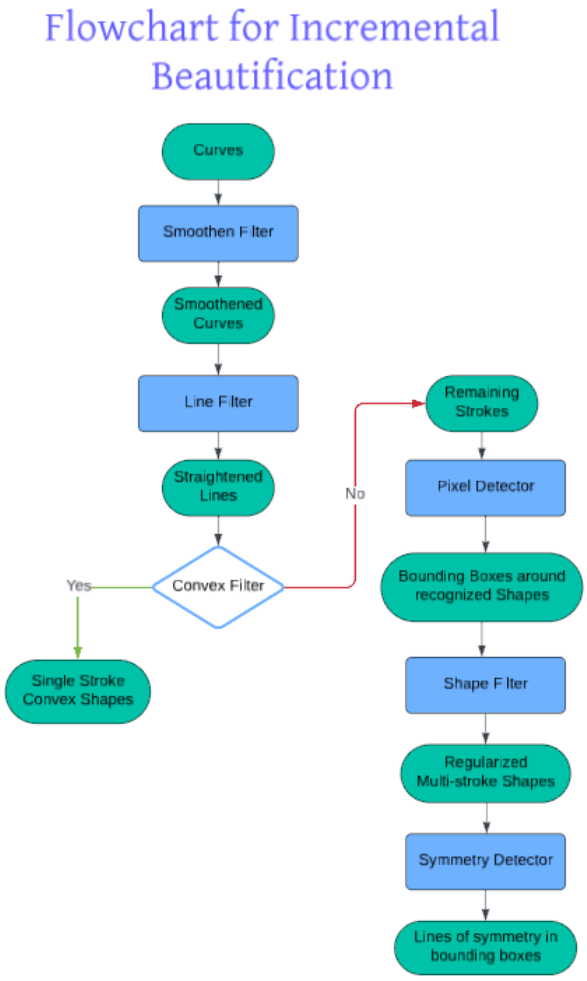

Fig. 1. flowchart for Incremental Beautification

Shape Identification and Regularization

The next step involves the `ConvexFilter`, which processes the non-linear curves. This filter aims to recognize and regularize basic geometric shapes such as circles, el-

lipses, and polygons that are formed by a single curve. The `ConvexFilter` evaluates the convexity of each curve and further determines whether it matches the properties of specific shapes. If a match is found, the filter replaces the irregular curve with a regularized version of the identified shape. The output at this stage is a combination of regularized basic shapes and complex, unrecognized curves.

Residual Curve Conversion

The curves that remain after the basic shapes have been regularized are then converted into an image format. This step prepares the remaining data for processing by a neural network, enabling the detection of more complex shapes that cannot be easily identified by the previous filters.

Object Detection and Refinement

The image generated from the leftover curves is passed through a `PixelDetector`, which is based on a Faster R-CNN model. This detector identifies complex shapes within the image, providing bounding boxes and labels for each detected object. The bounding boxes are used to isolate each shape, and the corresponding labels help in determining the type of regularization that should be applied. The detected shapes are then refined by filling them in with regularized versions, ensuring they conform to a more structured and visually appealing form.

Symmetry Exploration

Finally, the refined shapes are analyzed for lines of symmetry. Identifying and enhancing symmetry within the detected figures further enhances the visual appeal of the doodle, making the final output more balanced and aesthetically pleasing.

Throughout this incremental process, the data flows from one filter to the next, with each filter refining the curves based on its specific function. The combination of algorithmic regularization and neural network-based detection ensures that the doodles are progressively beautified in a structured and efficient manner.

Curve Regularization using B-Spline Interpolation and Linear Regression

The regularization process is a crucial initial step in the Curvetopia pipeline, aimed at refining and standardizing the input curves before more advanced processing. This step involves two primary filters: the `SmoothenFilter` and the `LineFilter`, applied sequentially to the curves.

SmoothenFilter

The first filter applied is the `SmoothenFilter`, which focuses on enhancing the visual quality of all input curves by smoothing out irregularities. The `SmoothenFilter` uses B-spline interpolation to achieve this smoothening effect. Specifically, the filter takes each curve and generates a smooth B-spline representation using the `splprep` function from the `scipy.interpolate` module. This function creates a smooth curve based on a specified smoothening factor, which determines the extent of the curve's refinement. After obtaining the B-spline, the 'splev' function evaluates the spline at densely spaced parameter values, producing a new set of points that form a refined, smooth curve.

LineFilter

Following the smoothening process, the `LineFilter` is applied to the refined curves. The purpose of the `LineFilter` is to identify curves that are nearly linear and straighten them to enhance regularity. This filter employs a linear regression model to evaluate the linearity of each curve. If a curve's linearity meets or exceeds a predefined threshold, the `LineFilter` straightens the curve by fitting a straight line through it.

Curve Detection using Poly lines

The **Convex Filter** class is designed to identify and classify shapes based on convexity and various geometric properties. For a detailed understanding of the algorithm's workflow, refer to Fig. 2. Below are the key identification methods:

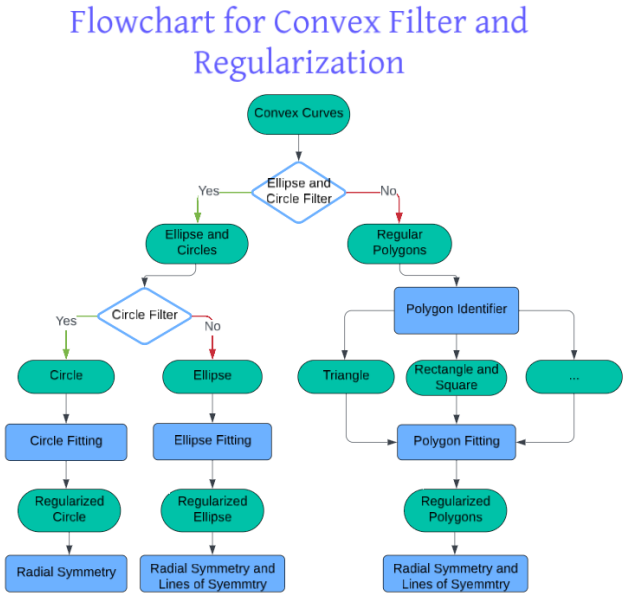

Fig. 2. Flowchart for Convex filter, Regularization and Symmetry Identification

1. Convex Shape Identification (`is_convex_shape`)

Concept: Determines if a shape is convex by comparing the area of the original shape to the area of its convex hull.

Formulas:

- **Original Area**: Calculated using the Shoelace formula:

$$\text{Area} = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})\right|$$

- **Convex Hull Area**: Same as above but applied to the convex hull vertices.
- **Convexity Check**: The difference between the convex hull area and the original area is compared to a threshold to determine convexity.

2. Ellipse or Circle Identification (`is_ellipse_or_circle`)

Concept: Determines if a shape is likely an ellipse or circle by calculating its circularity.

Formulas:

- **Area**: Same Shoelace formula as above.
- **Perimeter**: Summation of distances between consecutive points:

$$\text{Perimeter} = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

- **Circularity**:

$$\text{Circularity} = \frac{4\pi \times \text{Area}}{\text{Perimeter}^2}$$

- **Circularity Check**: Circularity is compared to a threshold to identify circles/ellipses.

3. Circle Identification (`is_circle`)

Concept: Differentiates between circles and ellipses by examining the aspect ratio of the bounding box around the shape.

Formulas:

- **Bounding Box**: Minimum and maximum x and y coordinates define the bounding box.
- **Aspect Ratio**: Ratio of the bounding box's height to its width.
- **Circle Check**: Aspect ratio is compared to a threshold to confirm if the shape is a circle (closer to 1).

4. Regular Polygon Identification (`identify_polygon`)

Concept: Identifies and classifies convex polygons by analyzing the number of edges and corners after filtering out circles and ellipses.

Steps:

- **Input Shapes**: The function processes shapes that are confirmed to be convex polygons after the circle and ellipse filters are applied.
- **Edge Detection**: Detects the number of edges by counting the vertices of the polygon.
- **Corner Analysis**: Analyzes the angles at each vertex to differentiate between various polygons (e.g., triangle, square, rectangle).
- **Polygon Classification**: Classifies the polygon based on the number of edges and the regularity of corner angles.

CURVE COMPLETION ON POLY LINES

CURVE FITTING FOR REGULAR SHAPES USING *Geometry*

Concept: This section focuses on the application of curve fitting techniques to identify and approximate regular shapes such as circles, ellipses, and polygons. By employing specific fitting functions, the geometric characteristics of these shapes are precisely determined. refer to Fig. 3 for results.

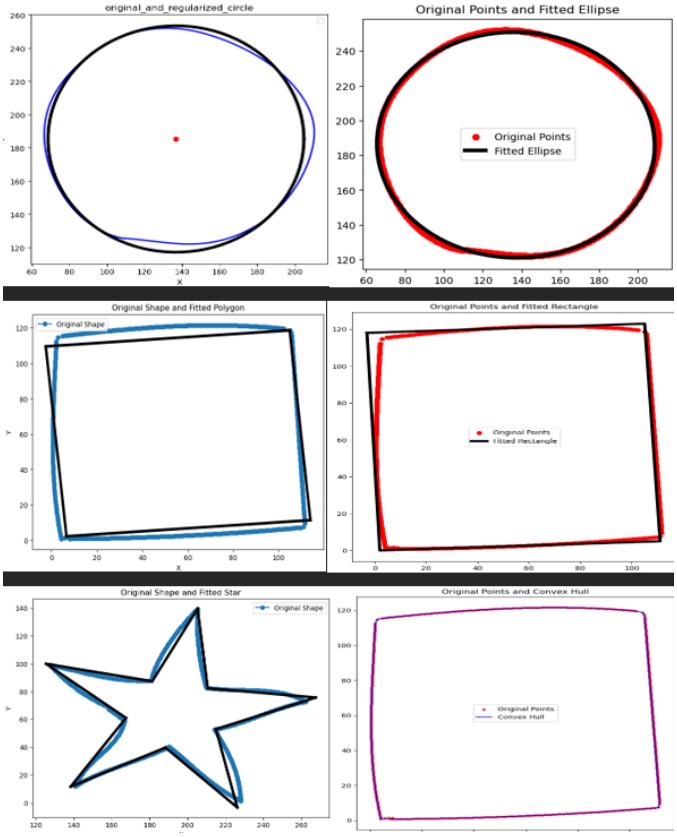

Fig. 3. Curve Fitting for Regular shapes

1. Circle Fitting (`fit_circle`)

Purpose: Accurately identifies and fits a circle to a set of points by minimizing the distance between the points and the circle's perimeter.

Method:

- **Initial Guess**: An initial estimate of the circle's center and radius is generated using the points' centroid and average distance.
- **Optimization**: The least squares method is applied to refine the circle's parameters, ensuring the best fit to the data points.
- **Symmetry**: Center of circle is radial center.
- **Result**: The output is the circle's center coordinates and radius, providing an optimal fit for circular shapes.

2. Ellipse Fitting (`fit_ellipse`)

Purpose: Determines the best-fitting ellipse for a given set of points, accommodating shapes that are elongated and asymmetrical.

Method:

- **General Form**: The fitting process starts by considering the general quadratic equation of an ellipse.
- **Parameter Estimation**: Key parameters such as the major and minor axes, orientation, and center are estimated through direct least squares fitting.
- **Refinement**: Further optimization ensures that the ellipse closely matches the shape of the input data.
- **Result**: The function returns the parameters that define the best-fitting ellipse.

3. Polygon Fitting (`fit_polygon`)

Purpose: Constructs and fits a regular polygon to a set of points by using the centroid and the highest y-value point to define the polygon's geometry, and visualizes symmetry lines.

Method:

- **Corner Extraction**: Identify corners from the provided curve and calculate their centroid.
- **Fixed Point and Radius**: Select the point with the highest y-value as the fixed point and calculate the radius from the centroid.
- **Angle Calculation**: Determine the initial angle of the fixed point relative to the centroid.
- **Polygon Generation**: Generate the vertices of a regular polygon based on the centroid, radius, and initial angle. Draw the regular polygon on the provided image.
- **Symmetry Lines**: Draw lines of symmetry based on vertex-to-opposite-vertex and side-midpoint-to-opposite-side-midpoint for even vertices, and vertex-to-opposite-side-midpoint for odd vertices. Extend lines slightly beyond the shape.
- **Radial Symmetry Lines**: Draw dotted radial lines from the centroid to beyond each vertex.

4. Rectangle Fitting (`fit_rectangle`)

Purpose: This method fits a rotated rectangle to a given set of points and identifies symmetry lines within the shape.

Method:

- **Bounding Box**: A rotated rectangle is fitted to the set of points using the minimum area rectangle method.
- **Drawing the Rectangle**: The fitted rectangle's top-left and bottom-right corners are used to draw the rectangle on the image.
- **Symmetry Lines**: Two symmetry lines are drawn across the rectangle, extending slightly beyond its boundaries to emphasize symmetry.
- **Result**: The function returns the original image and an image with highlighted symmetry lines.

5. Star Fitting (`fit_star`)

Purpose: Identifies and fits a star shape to a set of points, calculates centroid-based distances, and draws radial and mirror symmetry lines.

Method:

- **Vertex Detection**: The corners of the polygonal shape are identified, and their centroid is calculated.
- **Distance Calculation**: Distances from the centroid to each corner are computed to separate outer and inner points.
- **Outer Point Selection**: The fixed outer point with the highest y-value is identified, and radii for the outer and inner points are calculated.
- **Star Points Generation**: Star points are generated by alternating between outer and inner points, creating a star shape with vertices.
- **Symmetry Lines**: Symmetry lines are drawn both across the star (mirror symmetry) and radially from the centroid, with extensions for better visualization.
- **Result**: The function returns the plotted star shape with symmetry lines.

CURVE COMPLETION FOR INCOMPLETE CONVEX SHAPES USING *Convex Hull*

In geometric analysis and computer vision, incomplete convex shapes often require reconstruction to achieve accurate and complete representations. This section introduces a method that utilizes the ***convex hull*** to approximate and complete such shapes. By computing the convex hull and applying subsequent identification and regularization techniques, this approach ensures a precise and computationally efficient completion of convex shapes. Refer Fig. 4 and Fig. 5 for results.

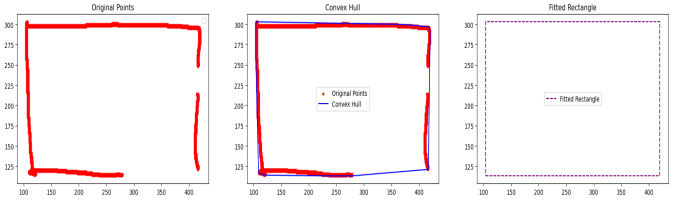

Fig. 4. Curve completion for Incomplete rectangle using Convex Hull

Method:

1) **Compute the Convex Hull**: For incomplete convex shapes, first calculate the ***convex hull*** of the given points. The convex hull serves as the smallest convex shape that can encompass all the given points, effectively approximating and completing the convex shape.
2) **Formulate the Convex Hull Points**: Convert the convex hull points into a list format. This list represents the boundary of the convex shape and will be used as a representation of the shape for subsequent analysis.
3) **Apply Convex Shape Identification and Regularization**: Pass the convex hull points to the Convex Shape Identification and Regularization process. This process involves fitting and regularizing the convex hull to ensure that the shape adheres to specific regularity criteria, similar to how complete convex shapes are processed.
4) **Ensure Efficiency and Accuracy**: This curve completion technique efficiently handles the completion of convex shapes with high accuracy while minimizing computational complexity. By focusing on the convex hull, the

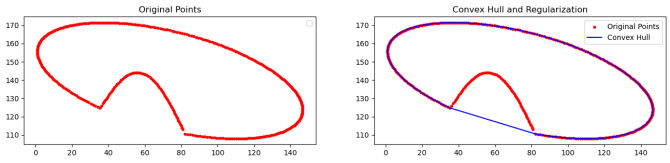

Fig. 5. Curve Completion for Incomplete Ellipse using Convex Hull

method simplifies the problem of shape completion and regularization.

Multi-Stroke Shape Detection Using FasterRCNN

The detection of multi-stroke shapes within doodles is a key component of the Curvetopia pipeline, leveraging FasterRCNN to identify and classify shapes. This process is divided into three stages: dataset creation, model training, and inference.

A. Dataset Creation

To train the FasterRCNN model, a synthetic dataset was created using the Quick, Draw! dataset by Google. The doodles were randomly scaled, transformed, and placed on a blank image canvas. Bounding boxes were generated around the drawn shapes, and the corresponding labels were recorded. This process was repeated to create a large, diverse dataset, which was uploaded as a Hugging Face dataset for ease of access and reproducibility.

B. Training

The FasterRCNN model, initialized with a ResNet-50 backbone pre-trained on COCO, was fine-tuned for the multi-stroke shape detection task. The model's box predictor was adapted to output predictions for eight classes, representing different shapes. The training process involved multiple epochs, where batches of images and targets were processed, and the model was optimized using gradient descent. Loss values were accumulated and monitored for each epoch, and the model was periodically saved to track progress.

C. Inference - PixelDetector

For inference, the trained FasterRCNN model is encapsulated within the `PixelDetector` filter. The filter converts the remaining curves into an image, which is then processed by the model to generate bounding boxes and corresponding labels. These outputs are refined by filtering out low-confidence predictions, ensuring that only the most accurate detections are used for subsequent processing. This allows for the effective identification of complex, multi-stroke shapes within the doodles, forming the basis for further regularization and beautification.

Curve Completion on Pixels Using Convex Hull

In the process of shape analysis and regularization, accurately identifying and completing shapes within bounding boxes is crucial, particularly when dealing with complex or incomplete structures. The use of a convex hull allows for the precise delineation of the main shape while excluding extraneous overlapping curves. This approach ensures that the identified shape is both accurate and ready for subsequent regularization and symmetry analysis.

Method

- **Bounding Box Identification:** Initially, bounding boxes are generated for the curves detected in the image, accompanied by curve labels and confidence scores. These bounding boxes encompass the curves of interest.
- **Convex Hull Construction:** For each bounding box, the convex hull of the contained curves is computed. The convex hull effectively removes extraneous curves that do not belong to the primary shape within the bounding box, isolating the true shape for further analysis.
- **Shape Verification and Regularization:** Once the convex hull is obtained, curve identification is performed to verify that the shape corresponds to the label assigned to the bounding box (e.g., circle, rectangle). If the label is accurate, the shape is then regularized using curve fitting techniques to ensure geometric precision.
- **Symmetry Analysis:** After confirming and regularizing the shape, appropriate lines of symmetry and radial symmetry are plotted. This step is essential for understanding the geometric properties of the shape and for applications where symmetry plays a critical role.

Exploring Symmetry

Algorithm for Finding Lines of Symmetry and Radial Symmetry in Regular Shapes

Symmetry plays a fundamental role in the geometric analysis of regular shapes, providing insights into their structural properties. In this section, we explore the algorithmic approach to identifying lines of symmetry and radial symmetry in various regular shapes. By understanding these symmetries, we can better appreciate the inherent balance and uniformity in geometric forms. Refer to Fig. 6 and Fig. 7 for results.

Lines of Symmetry

1) **Rectangle:** Horizontal and vertical lines of symmetry.
2) **Circle:** Infinite lines of symmetry passing through the center.
3) **Ellipse:** Horizontal and vertical lines of symmetry along the major and minor axes.
4) **Regular Polygons (e.g., triangles, squares):** Perpendicular lines from each vertex to the opposite side.

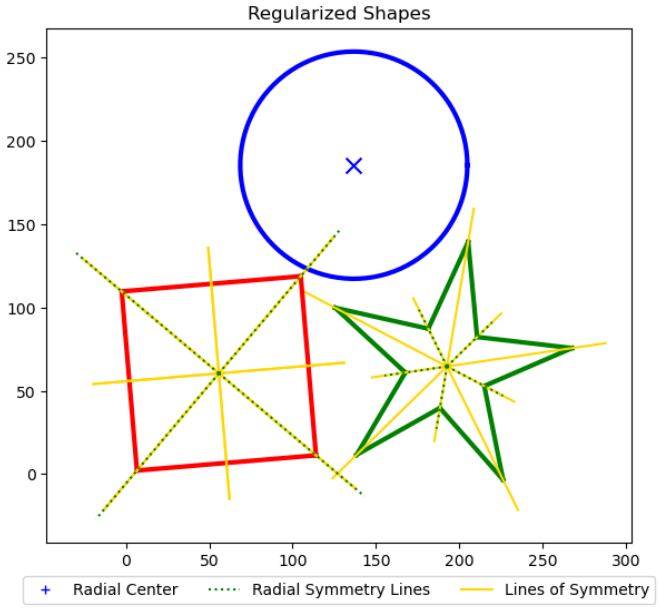

Fig. 6. Lines of Symmetry in Regular Shapes

Radial Symmetry

1) **Circle:** The center of the circle is the radial center.
2) **Ellipse:** The center of the ellipse is the radial center.
3) **Regular Polygons (e.g., triangles, squares):** The centroid of all points is the radial center, with radial symmetry lines extending from this centroid to all vertices.
4) **Stars:** The centroid of all points is the radial center, with radial symmetry lines extending from this centroid to all outer points.

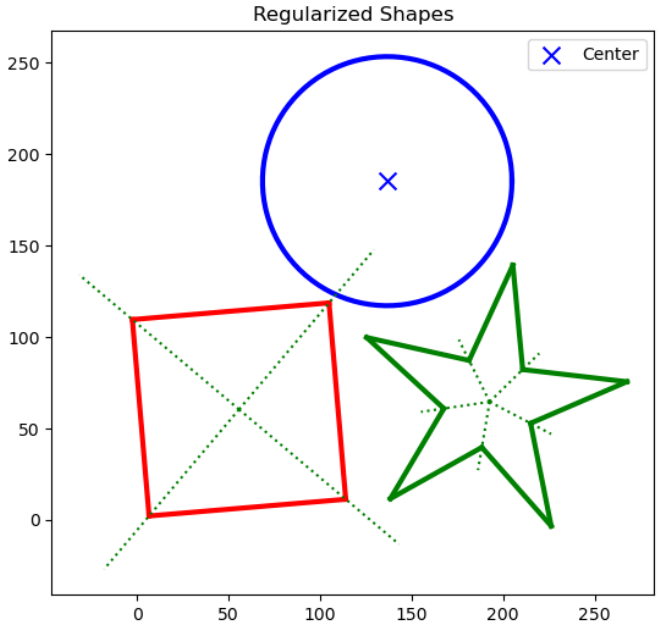

Fig. 7. Radial Symmetry in Regular shapes

Algorithm for Finding Lines of Symmetry in Irregular Shapes Using Principal Component Analysis (PCA)

Identifying lines of symmetry in irregular shapes can be challenging due to their complex geometries. However, *Principal Component Analysis (PCA)* provides a robust mathematical approach to tackle this problem. By analyzing the distribution of points in a shape, PCA helps in determining the best possible lines of symmetry, even in non-standard forms. This section outlines a step-by-step algorithm for detecting symmetry in irregular shapes using PCA, from contour extraction to symmetry evaluation and visualization. Refer to Fig. 8 and Fig. 9 for results.

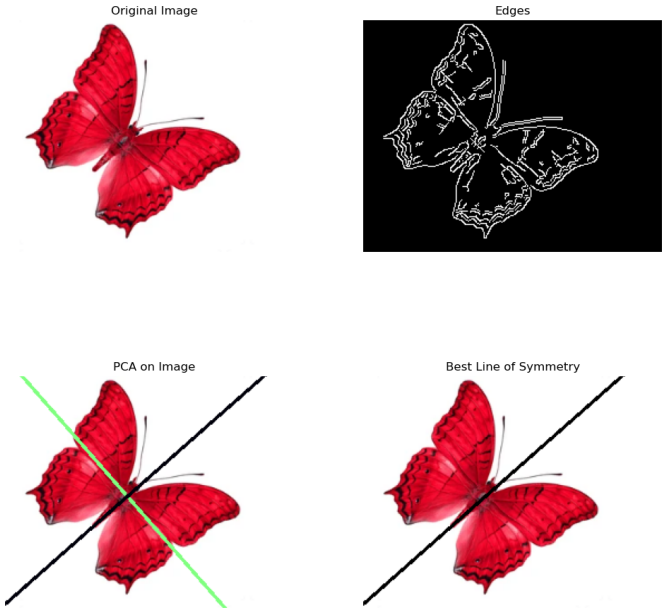

Fig. 8. Step-by-step Working of Algorithm on Butterfly Image

1. Extract Shape Contours

- Convert the image to grayscale if necessary.
- Apply edge detection (e.g., Canny) to identify contours.
- Extract polylines representing the shapes from the contours.

2. Principal Component Analysis (PCA)

- Combine points from all extracted polylines.
- Compute PCA to obtain the principal components and the mean point of the shape.

3. Mirror Points Across Components

- For each principal component (line), reflect every point of the shape across this line.
- Calculate the mirrored position by projecting each point onto the line and finding its reflection.

4. Evaluate Symmetry

- Measure Differences: Compute distances between each original point and its mirrored counterpart.
- Calculate Mean Squared Error (MSE): The symmetry score is determined by the MSE, which measures the average squared difference between the original and mirrored points.
- Lower Score Indicates Better Symmetry: A lower MSE suggests that mirrored points align closely with the original points, indicating a good symmetry candidate.
- Select Best Symmetry Line: Compare symmetry scores across all principal components and choose the one with the smallest MSE as the best symmetry line.

5. Draw Results

- Visualize the principal components on the original image.
- Highlight the best symmetry line on the image, extending it across the shape while leaving some space at the ends for better visual representation.

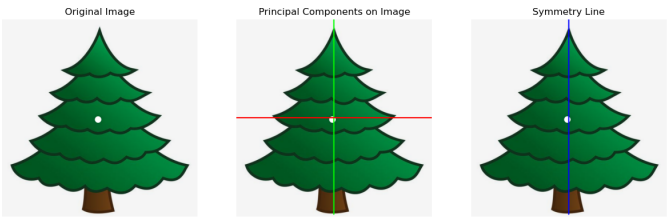

Fig. 9. Line of Symmetry in Irregular Shape : Tree

IMPORTANT FILES AND STEPS TO EXECUTE

Important Files

1) Main Notebook: `pipeline.ipynb`
2) Dataset Creation and ML Model Training: `/ShapeDetector_pixels`
3) Experiments on Algorithms and Techniques: `Algorithmic_Methods/Algorithmic_Approach.ipynb`, `Algorithmic_Methods/iterative_approach.ipynb`, `Algorithmic_Methods/bezier_curve_shapes.ipynb`, `Algorithmic_Methods/solution1.ipynb`

Steps to Execute the Code:

You can either run the hosted notebook on Kaggle (mentioned at start) or follow the steps below.

1) Download the model weights from the following location: Link to Model Weights.
2) Open the `pipeline.ipynb` notebook and provide the relative path of the saved model weights to the `PixelDetector` filter in pipeline.ipynb notebook.
3) Specify the correct path of the input doodle in the `read_csv` function during the initialization of the `Doodle` class.
4) After setting up the correct paths and ensuring the Jupyter notebook environment is properly configured, execute the entire notebook using the `Run All` option.
5) Regularized image and Image with symmetry lines are stored in `doddle.image` and `doddle.symmImage` respectively.

CONCLUSION

The development of algorithmic methods for curve identification and completion presents significant challenges due to the inherent complexity and variability in the shapes. This process often requires meticulous case-by-case analysis and extensive fine-tuning to handle the numerous edge cases that arise. While the use of data-driven and machine learning approaches might appear to be a brute-force solution, they are indispensable when dealing with such unconstrained and complex problems. These methods offer the flexibility and adaptability necessary to capture the nuances of diverse shapes and patterns. However, an optimal solution does not rely solely on one approach; it integrates the rigor of algorithmic techniques with the learning capabilities of machine learning. This hybrid approach leverages the strengths of both worlds, achieving more robust and comprehensive solutions to the challenges of curve identification and completion.

REFERENCES

[1] https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html