LAMPIRAN

1. Lampiran Kode Program

Kode program RGB Ke Grayscale

```
private Bitmap toGrayscale(Bitmap bitmap) {
        int width = bitmap.getWidth();
        int height = bitmap.getHeight();
        Bitmap grayscaleBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(grayscaleBitmap);
        Paint paint = new Paint();
        ColorMatrix colorMatrix = new ColorMatrix();
        colorMatrix.setSaturation(0);
        ColorMatrixColorFilter colorMatrixColorFilter = new
ColorMatrixColorFilter(colorMatrix);
        paint.setColorFilter(colorMatrixColorFilter);
        canvas.drawBitmap(bitmap, 0, 0, paint);
        return grayscaleBitmap;
}
```

```
Button rgbtogrey = findViewById(R.id.rgbkegrey);
rgbtogrey.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        convertToGrayscale(bitmap);
        ImageView imageView = findViewById(R.id.imageView);
        Bitmap resizedBitmap = resizeBitmapFromImageView(imageView, 500,
500);
        fetchImages(MainActivity.this);
    }
});
```

```
<Button
        android:id="@+id/rgbkegrey"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#4AD2FD"
        android:text="RGB ke-gray"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.057"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.06"
        app:toggleCheckedStateOnClick="false" />
```

Kode Greyscale ke Biner

```
private Bitmap threshold(Bitmap bitmap) {
        Bitmap output = Bitmap.createBitmap(bitmap.getWidth(),
bitmap.getHeight(), Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(output);

        Paint paint = new Paint();
        paint.setAntiAlias(true);
        paint.setColor(Color.BLACK);

        canvas.drawBitmap(bitmap, 0, 0, null);

        // Iterate through each pixel and set to black or white based
on threshold
        for (int x = 0; x < output.getWidth(); x++) {
            for (int y = 0; y < output.getHeight(); y++) {
                int pixel = output.getPixel(x, y);
                int red = Color.red(pixel);
                int green = Color.green(pixel);
                int blue = Color.blue(pixel);

                // Calculate grayscale value of pixel
                int grayscale = (int) (0.2989 * red + 0.5870 * green +
0.1140 * blue);

                // Set to black or white based on threshold
                if (grayscale < 128) {
                    output.setPixel(x, y, Color.BLACK);
                } else {
                    output.setPixel(x, y, Color.WHITE);
                }
            }
        }

        return output;
    }
```

```
<Button
    android:id="@+id/greykebiner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:backgroundTint="#4AD2FD"
    android:text="gray ke-biner"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.06"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.14"
    app:toggleCheckedStateOnClick="false" />
```

Kode Ekstraksi Fitur

a. Kode R (Red), G (Green), B (Blue)

```
private float[] calculateAverageRGB(Bitmap bitmap) {
        int width = bitmap.getWidth();
        int height = bitmap.getHeight();
        float[] averageRGB = new float[3];

        int r = 0, g = 0, b = 0;
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                int color = bitmap.getPixel(x, y);
                r += Color.red(color);
                g += Color.green(color);
                b += Color.blue(color);
            }
        }

        averageRGB[0] = (float) r / (width * height);
        averageRGB[1] = (float) g / (width * height);
        averageRGB[2] = (float) b / (width * height);
       TextView textView =
      findViewById(R.id.average_color);
        textView.setText(" R = " + averageRGB[0] + ", G =
" + averageRGB[1] + ", B = " + averageRGB[2]);
        return averageRGB;
    }
```

b. Kode Aspect ratio

```
private float calculateAspectRatio(Bitmap bmpBinarized) {
        double[] axisLengths =
calculateAxisLengths(convertToBinaryMatrix(bmpBinarized));
        double axisMajorLength = axisLengths[0];
        double axisMinorLength = axisLengths[1];
        float aspectRatio = (float) (axisMajorLength /
axisMinorLength);
        TextView resultTextView =
findViewById(R.id.aspectratio);
        String resultString = "AspectRatio:" +aspectRatio;
        resultTextView.setText(resultString);
        return aspectRatio;
    }
```

c. Kode Rectangularity

```
private float calculateRectangularity(Bitmap bmpBinarized) {
        int[][] binaryMatrix = convertToBinaryMatrix(bmpBinarized);
        double[] axisLengths = calculateAxisLengths(binaryMatrix);
        double axisMajorLength = axisLengths[0];
        double axisMinorLength = axisLengths[1];
        int area = calculateArea(bmpBinarized);
        float rectangularity = (float) (area / (axisMajorLength *
axisMinorLength));
        return rectangularity;
    }
```

d. Kode Compactness

```
public double calculateCompactness(Bitmap bitmap) {
      int width = bitmap.getWidth();
      int height = bitmap.getHeight();
      int[] pixels = new int[width * height];
      bitmap.getPixels(pixels, 0, width, 0, 0, width, height);
      int count = 0;
      int perimeter = 0;
      int[][] image = new int[height][width];

      // Konversi gambar ke citra biner dan hitung jumlah piksel
      for (int y = 0; y < height; y++) {
          for (int x = 0; x < width; x++) {
              int gray = Color.red(pixels[y * width + x]);
              image[y][x] = (gray > 128) ? 1 : 0;
              if (image[y][x] == 1) {
                  count++;
              }
          }
      }
for (int y = 0; y < height; y++) {
          for (int x = 0; x < width; x++) {
              if (image[y][x] == 1) {
                  if (y == 0 || x == 0 || y == height - 1 || x == width -
1) {
                      perimeter++;
                  } else if (image[y - 1][x] == 0 || image[y + 1][x] == 0
|| image[y][x - 1] == 0 || image[y][x + 1] == 0) {
                      perimeter++;
                  }
              }
          }
      }

      // Hitung compactness
      double area = count;
      double compactness = (4 * Math.PI * area) / (perimeter * perimeter);
      TextView compactnessTextView = findViewById(R.id.compactness);
      compactnessTextView.setText("Compactness: " + compactness);
      return compactness;
  }
```

e. Kode Program Roundess

```
public double calculateRoundness(Bitmap bitmap) {
        // Thresholding
        Bitmap thresholdedBitmap = threshold(bitmap);

        // Contour detection
        List<Point> contourPoints = findContour(thresholdedBitmap);

        // Calculate area
        double area = calculateArea(contourPoints);

        // Calculate perimeter
        double perimeter = calculatePerimeter(contourPoints);

        // Calculate roundness
        double roundness = (4 * Math.PI * area) / (perimeter *
perimeter);
        TextView roundnessTextView = findViewById(R.id.roundness);
        roundnessTextView.setText("Roundness: " + roundness);
        return roundness;
    }
```

f. Kode Program Area

```
public int calculateArea(Bitmap bitmap) {
        int width = bitmap.getWidth();
        int height = bitmap.getHeight();
        int[] pixels = new int[width * height];
        bitmap.getPixels(pixels, 0, width, 0, 0, width, height);
        Bitmap canny = cannyEdge(bitmap, 1.4, 69,128);

        //Bitmap sobel = sobelEdgeDetection(bitmap);
        // apply thinning to the bitmap
        //Bitmap thinnedBitmap = thinning(bitmap);

        // count the area of the thinned bitmap
        int area = 0;
        for (int i = 0; i < pixels.length; i++) {
            if (canny.getPixel(i % width, i / width) == Color.WHITE) {
                area++;
            }
        }

        // update the text view
        TextView textView = findViewById(R.id.area);
        textView.setText("Area: " + area);

        return area;
    }
```

g. Kode Program Perimeter

```
private double calculatePerimeter(List<Point> contourPoints) {
    double perimeter = 0.0;

    // Calculate the length of each side of the contour
    for (int i = 0; i < contourPoints.size(); i++) {
        Point p1 = contourPoints.get(i);
        Point p2 = contourPoints.get((i + 1) % contourPoints.size());
        double length = Math.sqrt(Math.pow(p2.x - p1.x, 2) +
Math.pow(p2.y - p1.y, 2));
        perimeter += length;
    }

    return perimeter;
}
```

Kode Program Button Ekstraksi

```
extractButton = findViewById(R.id.ekstraksi);
        extractButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (bitmap != null) {
                    Log.e("kkkk", "click");
                    calculateAverageRGB(bitmap);
                    int area = calculateArea(bitmap);
                    calculateAspectRatio(bitmap);
                    displayRectangularityResult(bitmap);
                    convertToBinaryMatrix(bitmap);
                    imageView.getDrawable()).getBitmap();
                    int perimeter = calculatePerimeter(bitmap);
                    calculateEccentricity(bitmap);
                    calculateRoundness(bitmap);
                    calculateCompactness(bitmap);

                }
            }
        });

<Button
    android:id="@+id/ekstraksi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:backgroundTint="#4AD2FD"
    android:text="ekstraksi"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.054"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.235"
    app:toggleCheckedStateOnClick="false" />
```

Kode Program proses klasifikasi metode KNN

```
private float calculateEuclideanDistance(float[] features,
float[]trainingData) {
    float distance  = 0;
    for (int i = 0; i < features.length; i++) {
        float diff = features[i] - trainingData[i];
        distance += diff * diff;
    }
    return (float) Math.sqrt(distance);

private void calculateAndDisplayEuclideanDistances(Bitmap bitmap,
float[][] trainingData, String[] trainingLabels) {
        float[] features = extractFeatures(bitmap);
        ArrayList<Float> distances = new ArrayList<>();
        ArrayList<String> labels = new ArrayList<>();
        for (int i = 0; i < trainingData.length; i++) {
            float distance = calculateEuclideanDistance(features,
trainingData[i]);
            distances.add(distance);
            labels.add(trainingLabels[i]);
        }
        displayEuclideanDistances(distances, labels);
    }
```

Kode Program Proses menghitung ekstraksi fitur

```
private float[] extractFeatures(Bitmap bitmap) {
    Bitmap bmpBinarized = binarizeImage(bitmap);
    float[] averageRGB = calculateAverageRGB(bitmap);
    float area = calculateArea(bmpBinarized);
    float aspectRatio = calculateAspectRatio(bmpBinarized);
    float rectangularity = calculateRectangularity(bmpBinarized);
    double eksentrik = calculateEccentricity(bitmap);
    float perimeter = calculatePerimeter(bitmap);
    float roundness = (float) calculateRoundness(bitmap);
    double compactness = calculateCompactness(bitmap);

    Log.e("kkk average", String.valueOf(averageRGB[0]));
    float normalizedArea = (float) area / (bitmap.getWidth() *
bitmap.getHeight());
    float normalizedPerimeter = (float) perimeter / (2 *
(bitmap.getWidth() + bitmap.getHeight()));
    float[] features = new float[2];
    // Set the extracted features
    features[0] = normalizedArea;
    features[1] = normalizedPerimeter;


    return new float[]{averageRGB[0], averageRGB[1], averageRGB[2],
area, aspectRatio, rectangularity, (float) eksentrik, perimeter,
roundness,(float) compactness}; //

}
```

Kode Program Proses pengambilan citra dari galery

```
ImageView selectFromGalleryButton = findViewById(R.id.imageView);
selectFromGalleryButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        Intent selectFromGalleryIntent = new Intent(Intent.ACTION_PICK,
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(selectFromGalleryIntent,
REQUEST_GALLERY_IMAGE);
    }
});

}
```

```
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK) {
        if (requestCode == REQUEST_IMAGE_CAPTURE) {
            Bundle extras = data.getExtras();
            bitmap = (Bitmap) extras.get("data");
            imageView.setImageBitmap(bitmap);
            Uri imageUri = data.getData();
            int quality = 50;
            Bitmap.CompressFormat format = Bitmap.CompressFormat.JPEG;
            compressImage(imageUri, quality, format);
            clear();
        } else if (requestCode == REQUEST_GALLERY_IMAGE) {
            Uri selectedImage = data.getData();
            try {

                bitmap =
MediaStore.Images.Media.getBitmap(getContentResolver(), selectedImage);
                imageView.setImageBitmap(bitmap);
                clear();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Kode Program perhitungan jarak euclidean

```
private float calculateEuclideanDistance(float[] features,
float[]trainingData) {
    float distance  = 0;
    for (int i = 0; i < features.length; i++) {
        float diff = features[i] - trainingData[i];
        distance += diff * diff;
    }
    return (float) Math.sqrt(distance);
```

```
private void displayEuclideanDistances(ArrayList<Float> distances,
ArrayList<String> labels) {
    ArrayList<Integer> indices = sortArrayList(distances);
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("Euclidean distances :\n");
    for (int i : indices) {
        //stringBuilder.append(labels.get(i));
        //stringBuilder.append(": ");
        stringBuilder.append(distances.get(i));
        stringBuilder.append("\n");
    }
    TextView textView = findViewById(R.id.euclidean);
    textView.setText(stringBuilder.toString());
}

private void calculateAndDisplayEuclideanDistances(Bitmap bitmap,
float[][] trainingData, String[] trainingLabels) {
    float[] features = extractFeatures(bitmap);
    ArrayList<Float> distances = new ArrayList<>();
    ArrayList<String> labels = new ArrayList<>();
    for (int i = 0; i < trainingData.length; i++) {
        float distance = calculateEuclideanDistance(features,
trainingData[i]);
        distances.add(distance);
        labels.add(trainingLabels[i]);
    }
    displayEuclideanDistances(distances, labels);
}


private ArrayList<Integer> sortArrayList(ArrayList<Float> distances) {
    ArrayList<Integer> indices = new ArrayList<>();
    for (int i = 0; i < distances.size(); i++) {
        indices.add(i);
    }
    for (int i = 0; i < distances.size(); i++) {
        for (int j = i + 1; j < distances.size(); j++) {
            if (distances.get(j) < distances.get(i)) {
                float tempDistance = distances.get(i);
                distances.set(i, distances.get(j));
                distances.set(j, tempDistance);
                int tempIndex = indices.get(i);
                indices.set(i, indices.get(j));
                indices.set(j, tempIndex);
            }
        }
    }
    return indices;
}
```

Kode Program meghitung nilai K

```
private float[] calculateAllDistances(float[] features, float[][]
trainingData) {
    float[] distances = new float[trainingData.length];
    for (int i = 0; i < trainingData.length; i++) {
        float distance = calculateEuclideanDistance(features,
trainingData[i]);
        distances[i] = distance;
    }
    return distances;
}

private int[] getKNearestIndices(float[] distances, int k) {
    int[] indices = new int[k];
    for (int i = 0; i < k; i++) {
        int closestIndex = 0;
        float closestDistance = Float.MAX_VALUE;
        for (int j = 0; j < distances.length; j++) {
            if (distances[j] < closestDistance) {
                closestIndex = j;
                closestDistance = distances[j];
            }
        }
        distances[closestIndex] = Float.MAX_VALUE;
        indices[i] = closestIndex;
    }
    return indices;
}

private String[] getKNearestLabels(int[] kNearestIndices, String[]
trainingLabels) {
    String[] kNearestLabels = new String[kNearestIndices.length];
    for (int i = 0; i < kNearestIndices.length; i++) {
        int index = kNearestIndices[i];
        kNearestLabels[i] = trainingLabels[index];
    }
    return kNearestLabels;
}

private String voteKNearest(String[] kNearestLabels) {
    HashMap<String, Integer> labelCount = new HashMap<>();
    int maxCount = 0;
    String maxLabel = "";

    for (String label : kNearestLabels) {
        int count = labelCount.getOrDefault(label, 0) + 1;
        labelCount.put(label, count);

        if (count > maxCount) {
            maxCount = count;
            maxLabel = label;
        }
    }

    return maxLabel;
}
```

Kode Program Klasifikasi makanan

```
private void classifyFood(Bitmap bitmap) {
    // Extract features from input image
    float[] features = extractFeatures(bitmap);

    // Classify the food image based on the extracted features
    String predictedLabel = "";
    float minDistance = Float.MAX_VALUE;
    for (int i = 0; i < trainingData.length; i++) {
        float[] trainFeatures = trainingData[i];
        // Calculate the distance between the input image features and
the training image features
        float distance = calculateEuclideanDistance(features,
trainFeatures);
        // Check if the current distance is the smallest so far
        if (distance < minDistance) {
            minDistance = distance;
            predictedLabel = trainingLabels[i];
        }
    }

    // Display the predicted label with the corresponding calorie value
    TextView hasilTextView = findViewById(R.id.hasilknn1);
    hasilTextView.setText("Hasil Klasifikasi:" + predictedLabel + "|" +
getKaloriKeterangan(predictedLabel));
}
```

```
private String getKaloriKeterangan(String label) {
    switch (label) {
        case "Bakso":
            return "Bakso\t1 porsi (108 g) 218 kalori untuk bakso Sapi"+
                   "\t1 porsi (108 g) 174 kalori untuk bakso Ayam\n"
                   ;
        case "Rendang":
            return "\t1 porsi (240 g) 468 kalori untuk daging\n" +
                   "\t1 porsi (380 g) 664 kalori dalam 1 bungkus\n" +
                   "\t1 porsi (125 g) 189 kalori untuk jengkol";
        case "Rujak Buah":
            return "Rujak Buah\t1 porsi (95 g) 202 kalori";
        case "Sate":
            return "Sate\t 1 porsi(45g) 101  kalori untuk ayam\n" +
                   "\t 1 porsi (45g) 97 kalori untuk kambing\n" +
                   "\t 1 porsi (30g) 42 kalori untuk usus";
        case "Gulai":
            return "Gulai\t 1 porsi (240 g) 301 kalori untuk gulai
kambing\n" +
                   "\t 1 porsi (240 g) 404 kalori untuk gulai ayam"+
                   "\t 1 porsi (100 g) 66 kalori untuk sayur nangka\n"
+
                   "\t 1 porsi (187 g) 204 kalori untuk gulai daging
sapi\n";
        case "Nasi Goreng":
            return "Nasi Goreng\t1 porsi (149 g) 250 kalori";
        case "Soto":
            return "Soto\t1 porsi (241 g) 312 kalori untuk ayam\n" +
                   "\t1 porsi (241 g) 219 kalori untuk daging\n" +
                   "\t1 porsi (200 g) 200 kalori untuk kikil";
        default:
            return "";
    }
}
```