Binary Cataract Classification with Deployment API

1. **Data Preprocessing Stage:**
   - **a.** EDA purposes - notebooks/datapreprocess.ipynb
   - b. Data Augmentation - src/preprocessing.py

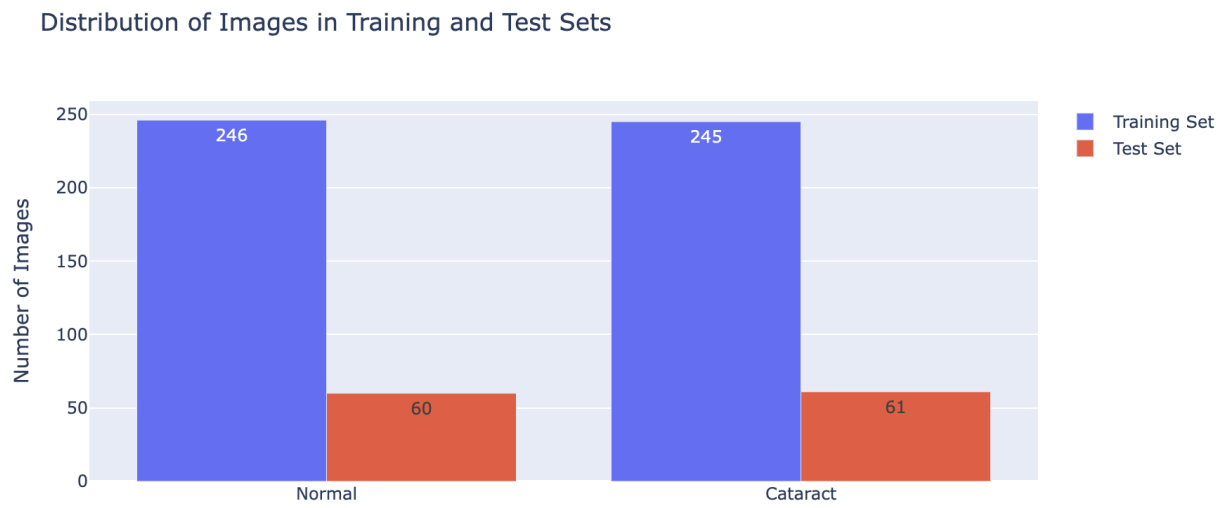
Training Categories Labelled as: ['cataract', 'normal']

i. Found 491 images belonging to 2 classes.

Testing Categories Labelled as: ['cataract', 'normal']

ii. Found 121 images belonging to 2 classes.

We have a balanced distribution for train and test dataset, but overall dataset size is relatively small for deep learning networks to implement and attain accurate results.



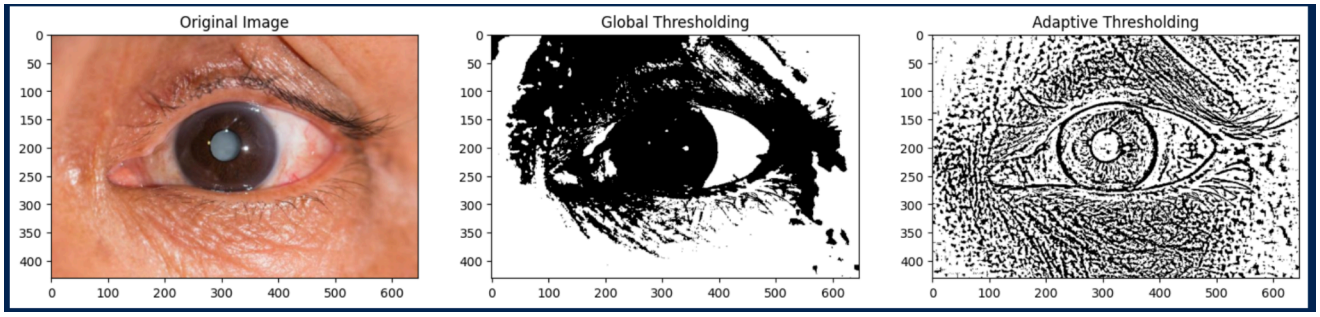
EDA purposes - Image Thresholding

1. Segmentation of Foreground and Background

It separates objects (foreground) from the background by applying a threshold value. Pixels above a threshold become white (255), and those below become black (0).

2. Edge and Shape Detection

Helps detect edges and contours by emphasizing the changes in pixel intensity. Particularly useful in medical imaging, facial recognition, and object tracking.



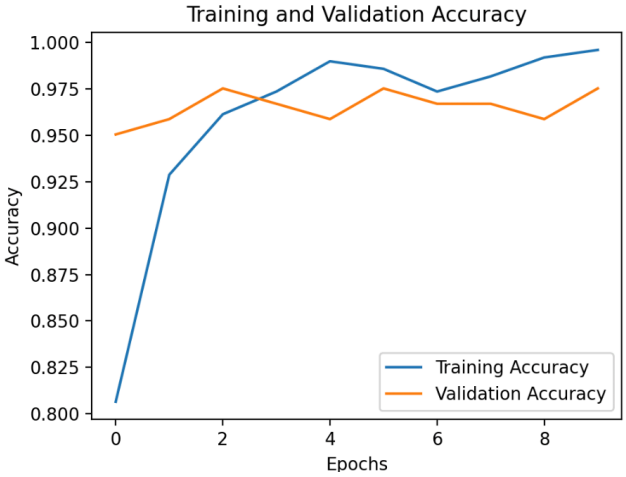
Data Augmentation:
Setting data augmentation tailored for medical imagery [Reference Link] :

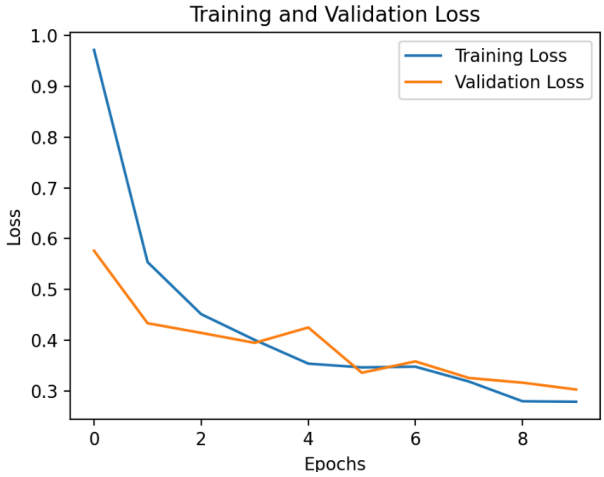
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,              # Subtle rotations (±10°)
        width_shift_range=0.1,          # Minor horizontal translations (10%)
        height_shift_range=0.1,         # Minor vertical translations (10%)
        brightness_range=[0.85, 1.15],   # Subtle brightness adjustments (±15%)
        zoom_range=0.1,                 # Slight zoom variations (±10%)
        horizontal_flip=True,           # Horizontal flips are anatomically valid
        fill_mode='nearest'          # Fill mode for any empty pixels after transformations
    )
```

2. Model Training: src/model.py
   - Using transfer learning from VGG16 architecture pre-trained features.
     - Why VGG16?
       - Pre-trained on ImageNet, has 138M params in total.
       - Useful in medical use cases where data is limited or hard to label.
       - Helps it learn low-level features like edges, shapes, and textures, which are often transferable to medical imaging (e.g. eye textures, cloudiness in cataract cases).
       - VGG16, when used with transfer learning + data augmentation, performs well even with limited data.
   - Adam optimiser with an initial learning rate of 0.0001 running over 20 epochs.
   - Including fine-tuning the last few layers of the base model for improved performance, running over 10 epochs.
   - Applying callbacks for early stopping to prevent overfitting and best model checkpoints.
   - Model saved at - models/best_model_v2_vgg16.h5
   - Label smoothing: Helps handle noisy labels in binary classification.

3. Evaluation Report: notebooks/evaluation_test.ipynb
   a. **Evaluation Metrics:**
      Test Accuracy: 99.95%

Test Recall: 100.00%
Test Precision: 93.75%
Test AUC: 96.69%





i. The model is learning well on training data and generalizing decently on validation data.
ii. High AUC (96.69%) reflects excellent classification capability overall.
iii. Training > Validation Accuracy suggests minor overfitting, but performance is still robust on unseen test data.

b. Threshold Tuning: Choosing optimal threshold for minimizing the False Negatives (max recall), critical in medical diagnosis and keeping precision high, therefore to avoid too many False Positives. And High F1 score to keep in check with the Harmonic balance between Precision & Recall.

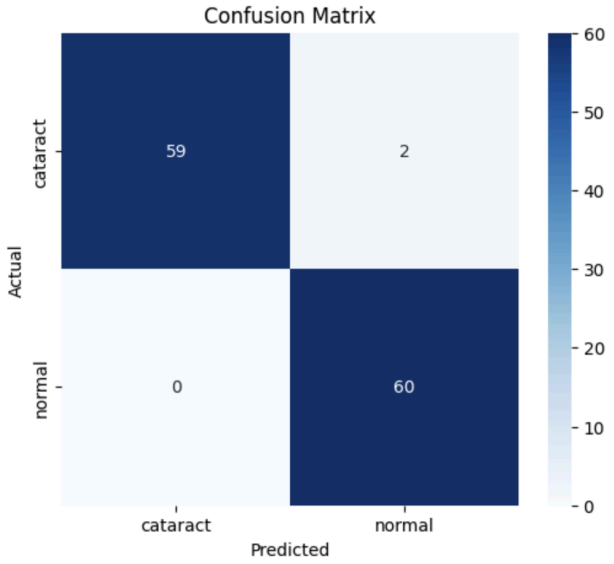
Best fit: Threshold: 0.60 | F1-score: 0.9836 | Precision: 0.9677 | Recall: 1.0000

c. Classification Report:

	precision	recall	f1-score	support
cataract	1.00	0.97	0.98	61
normal	0.97	1.00	0.98	60
accuracy			0.98	121
macro avg	0.98	0.98	0.98	121
weighted avg	0.98	0.98	0.98	121

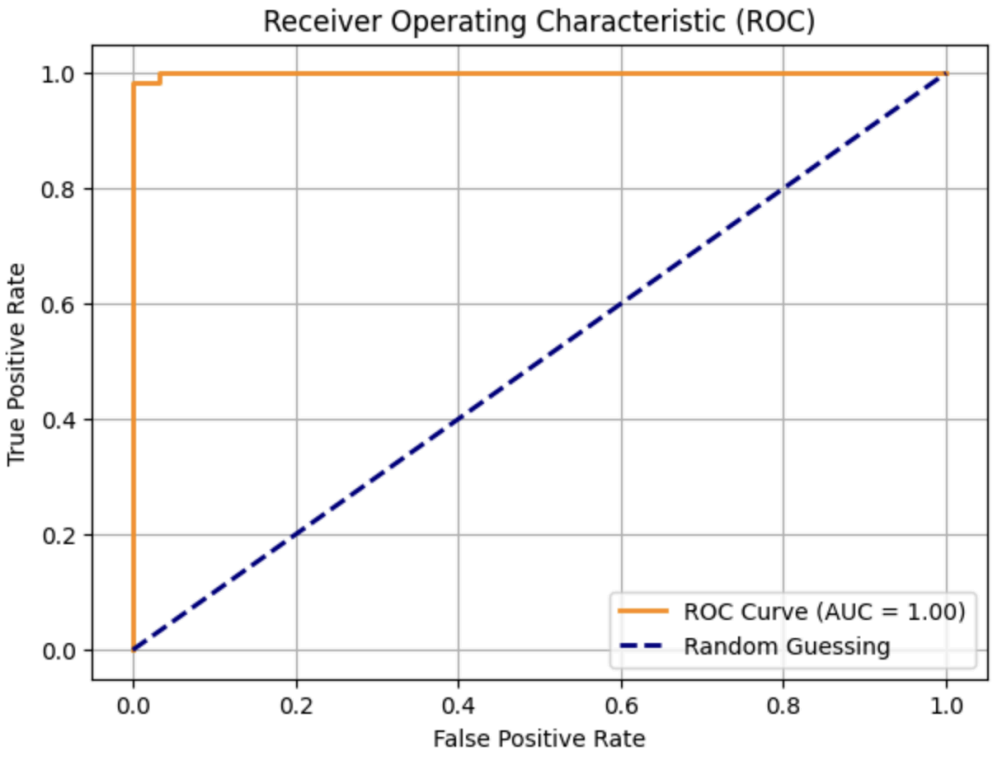
- Model is highly balanced — it's not biased toward one class.
- No cataract cases were missed (Recall = 1.0 for normal = true negatives correctly identified).

d. Confusion Matrix:



Only a few misclassifications (2 or 3 samples at most).
No false positives for class 1 (normal), only 2 false negatives for class 0 (cataract).

e. ROC Curve & AUC:



The orange curve touches the top-left corner, which means the model is perfectly distinguishing between the classes.

AUC = 1.00 (Area Under the Curve) indicates 100% performance thus the ideal.

4. **Model Deployment:**

FastAPI Backend – main.py

This file serves the model backend API for cataract image classification.

- **Model Loading**:
  - Loads a trained `.h5` Keras model (`best_model_vgg16.h5`).

- **Image Preprocessing**:
  - Converts image to RGB.
  - Resizes to 224x224 pixels.
  - Normalizes pixel values to [0, 1].
  - Adds batch dimension.

- **Prediction API Endpoint** (`/predict/`):
  - Method: POST
  - Accepts: Image file (`UploadFile`)
  - Returns: JSON response with:
    - `prediction`: Cataract or Normal
    - `confidence`: Confidence% (rounded to 2 decimals)
- **Threshold Logic** (As per Optimal threshold value received):

  - Prediction < 0.6 → "Cataract"
  - Prediction ≥ 0.6 → "Normal"
  - Confidence computed accordingly.

Streamlit Frontend – streamlit_app.py

This file provides a simple interactive UI for Cataract prediction using the FastAPI backend.

- **Sample Image Testing**:
  - Displays up to 5 sample images from `sample_images/`.
  - Predict button for each sample triggers API call.
  - Shows prediction label and confidence.

- **Image Upload Section**:

    - User can upload `.jpg`, `.jpeg`, or `.png` files.

    - On clicking "Predict Uploaded Image", sends image to FastAPI.
    - Displays prediction result with confidence%.

- **Backend Communication**:
    - Sends POST request to `http://127.0.0.1:8000/predict/`.
    - Handles both success and error responses.

Steps on how to use the API?

STEP 1: Under the /api directory, install all the libraries from requirements.txt file.

STEP 2: First we'll test out on the FastAPI - Swagger UI directly:

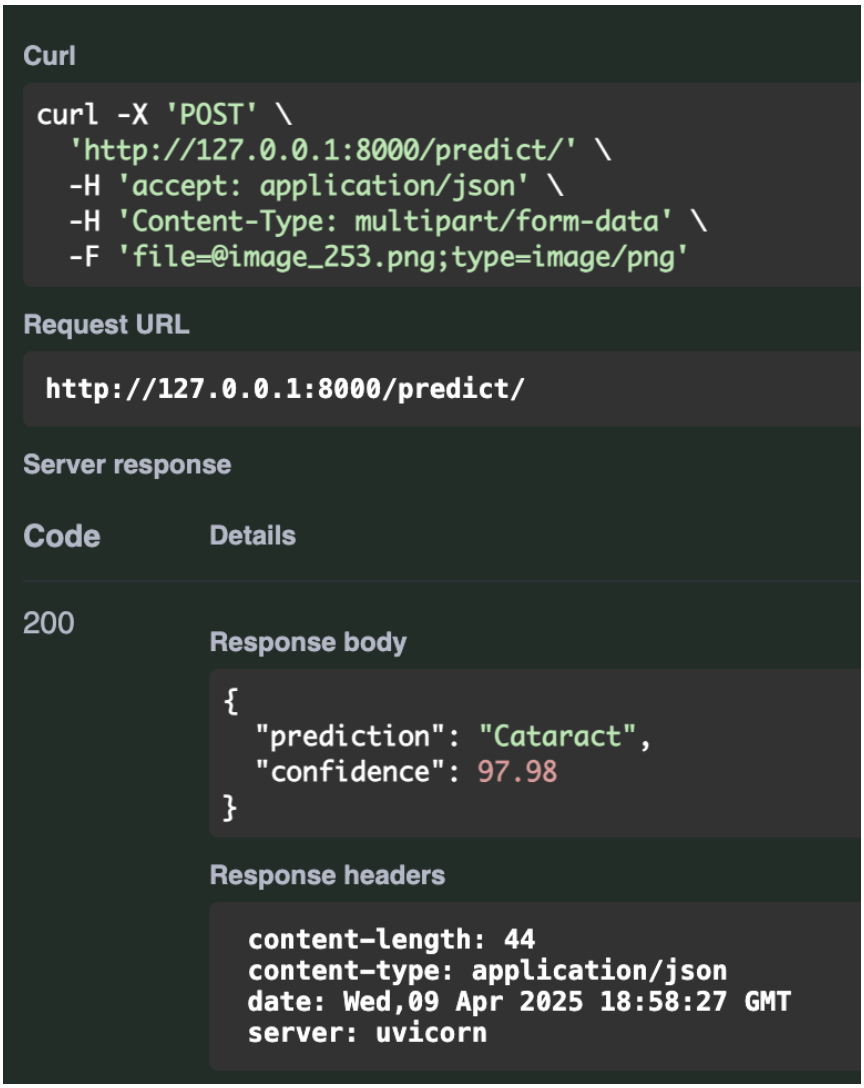
1. Run the API command on terminal: uvicorn main:app --reload
2. Then visit: http://127.0.0.1:8000/docs to test your API.
3. Click on the POST /predict/ endpoint.
4. Click "Try it out".
5. Use the "Choose File" button to upload your image (.jpg, .png, etc.).
6. Click "Execute" to get the prediction with confidence.

Example CURL urls are given below:

```
Example 1:

curl -X 'POST' \
  'http://127.0.0.1:8000/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@image_253.png;type=image/png'

{
  "prediction": "Cataract",
  "confidence": 97.98
}

Example 2:
```

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@image_304.png;type=image/png'
```

```
{
  "prediction": "Normal",
  "confidence": 85.17
}
```



STEP 3: Now, to test on Streamlit (single image file testing and multiple samples output testing combined):

7. Under the /api directory, run this command on a new terminal: streamlit run streamlit_app.py
8. Visit the http://localhost:8501/
9. Click on sample image ‘Predict’ button to see results.
10. Or, upload an image of your own from app/samples/ folder, then click ‘Predict Uploaded image’ to see the results [Predicted Class with Confidence%]