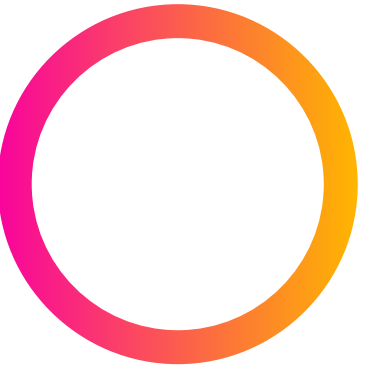


DOCUMENT SEGMENTATION



USING DEEPLABV3 - MOBILENETV3LARGE MODEL




Hardianto Tandi Seno



hardiantotandiseno@gmail.com

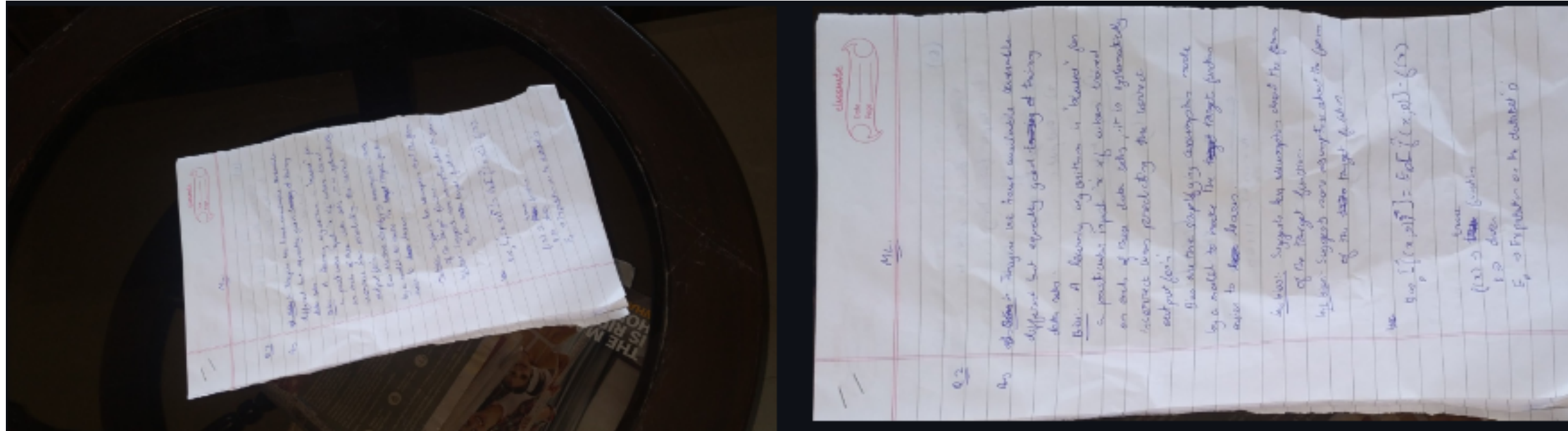


CONTENT

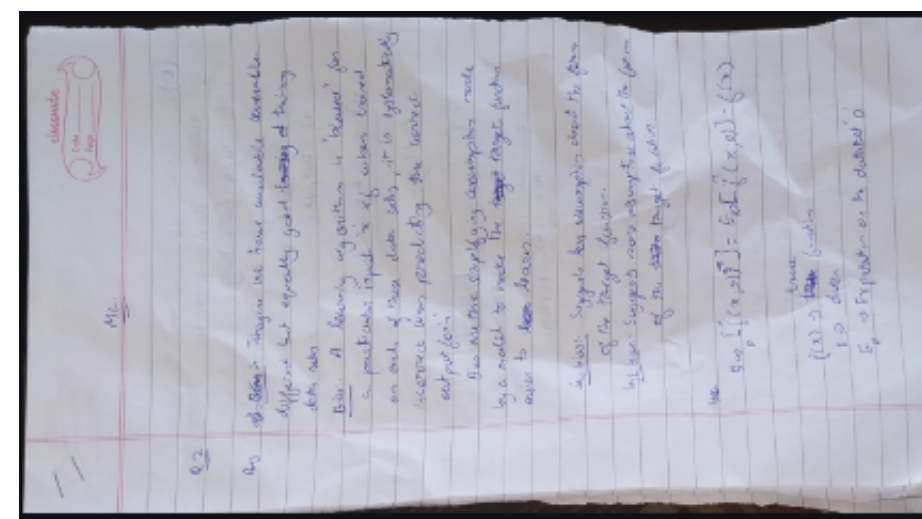
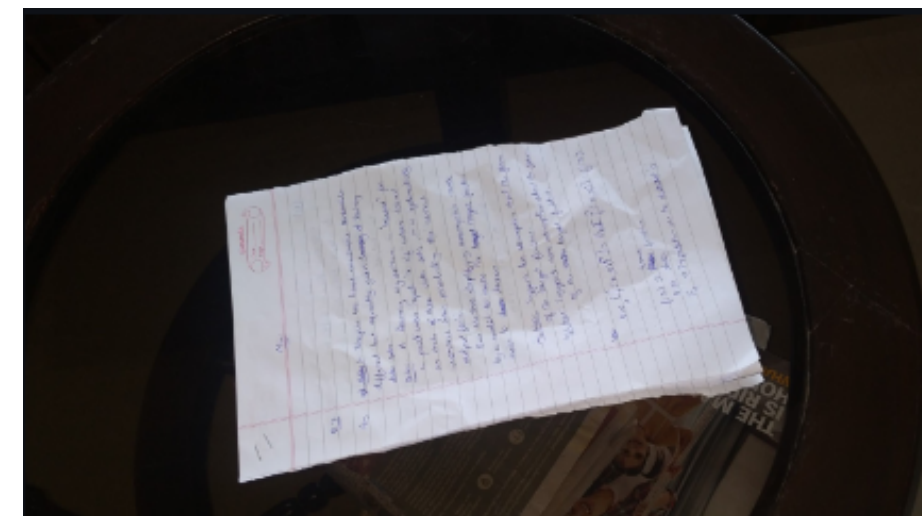
- 
- ABOUT DOCUMENT SEGMENTATION
 - PROJECT WORKFLOW
 - PROJECT IMPLEMENTATION
 - CHALLENGES & CONCLUSION



ABOUT DOCUMENT SEGMENTATION



Document Segmentation (Scanner) is a method in **computer vision (can applied to deep learning based image segmentation model)** to change document from physical docs to digital docs. The **essence of this method** is that the **system will detect the background and objects** from the document image, then the **background will be removed** & only the **object will remain** (the document image). After that, **perspective adjustments** are made so that the **document can be straight**

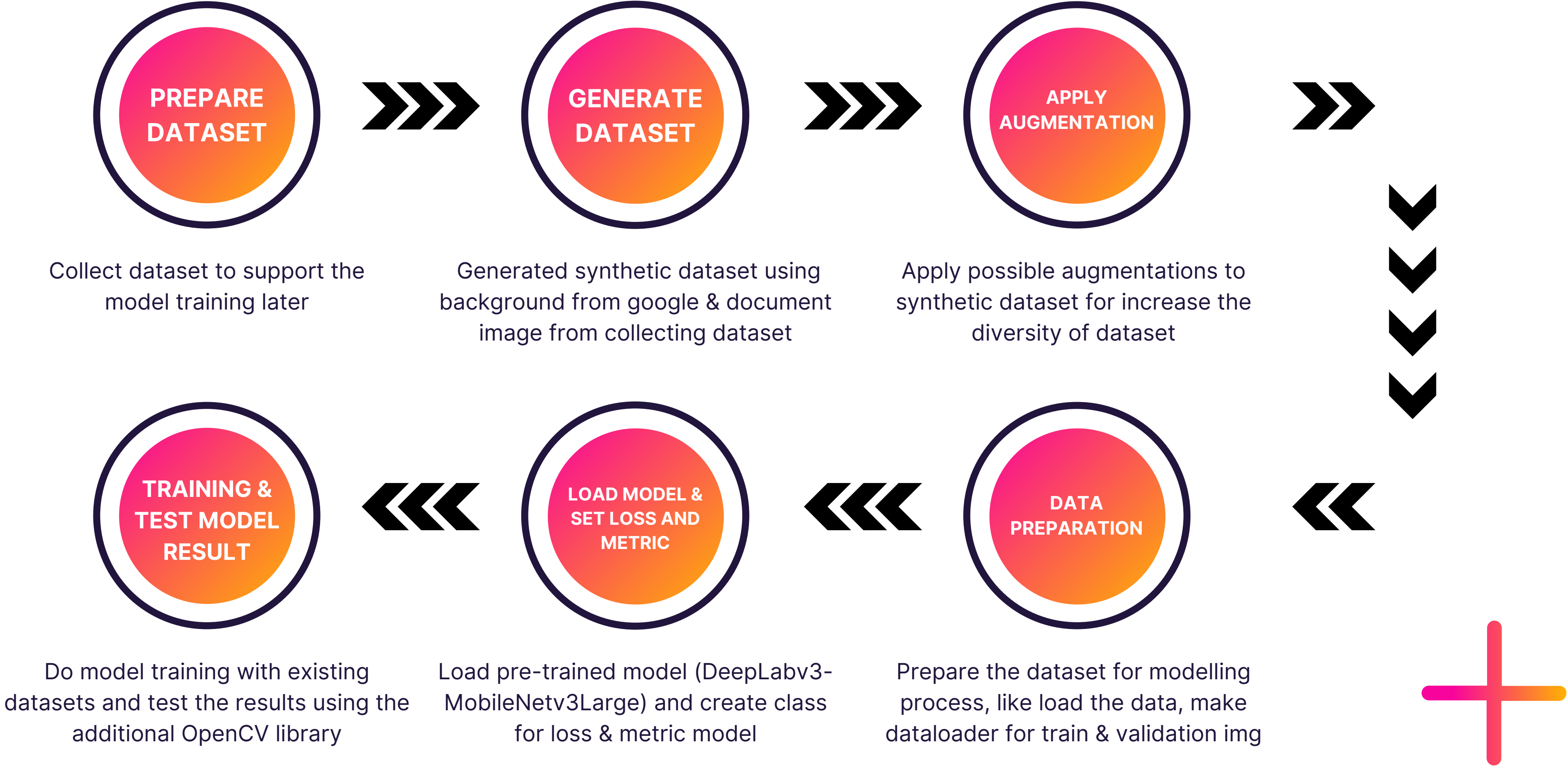




PROJECT WORKFLOW



PROJECT WORKFLOW

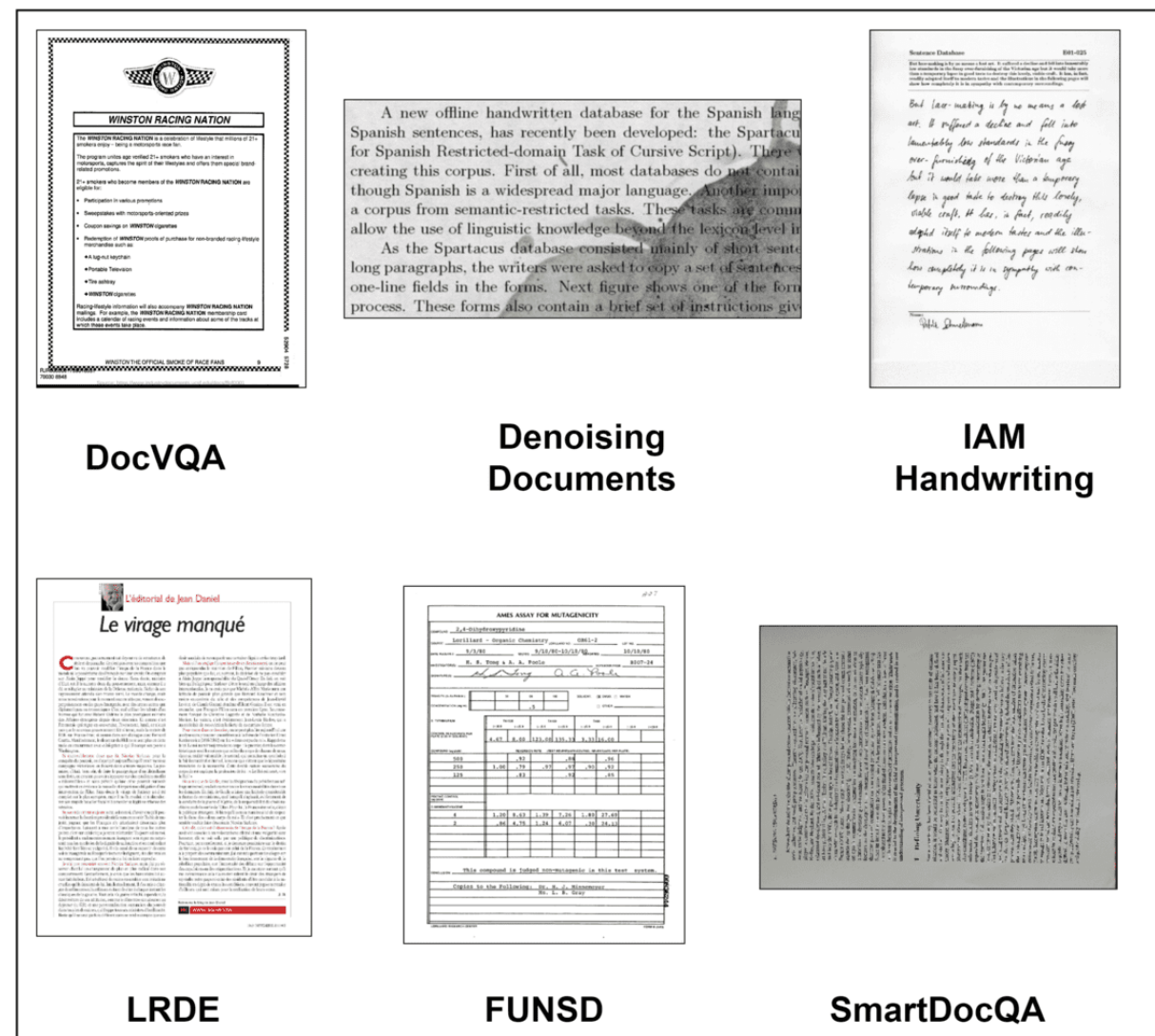




PROJECT IMPLEMENTATION



PREPARE DATASET



Prepare Dataset

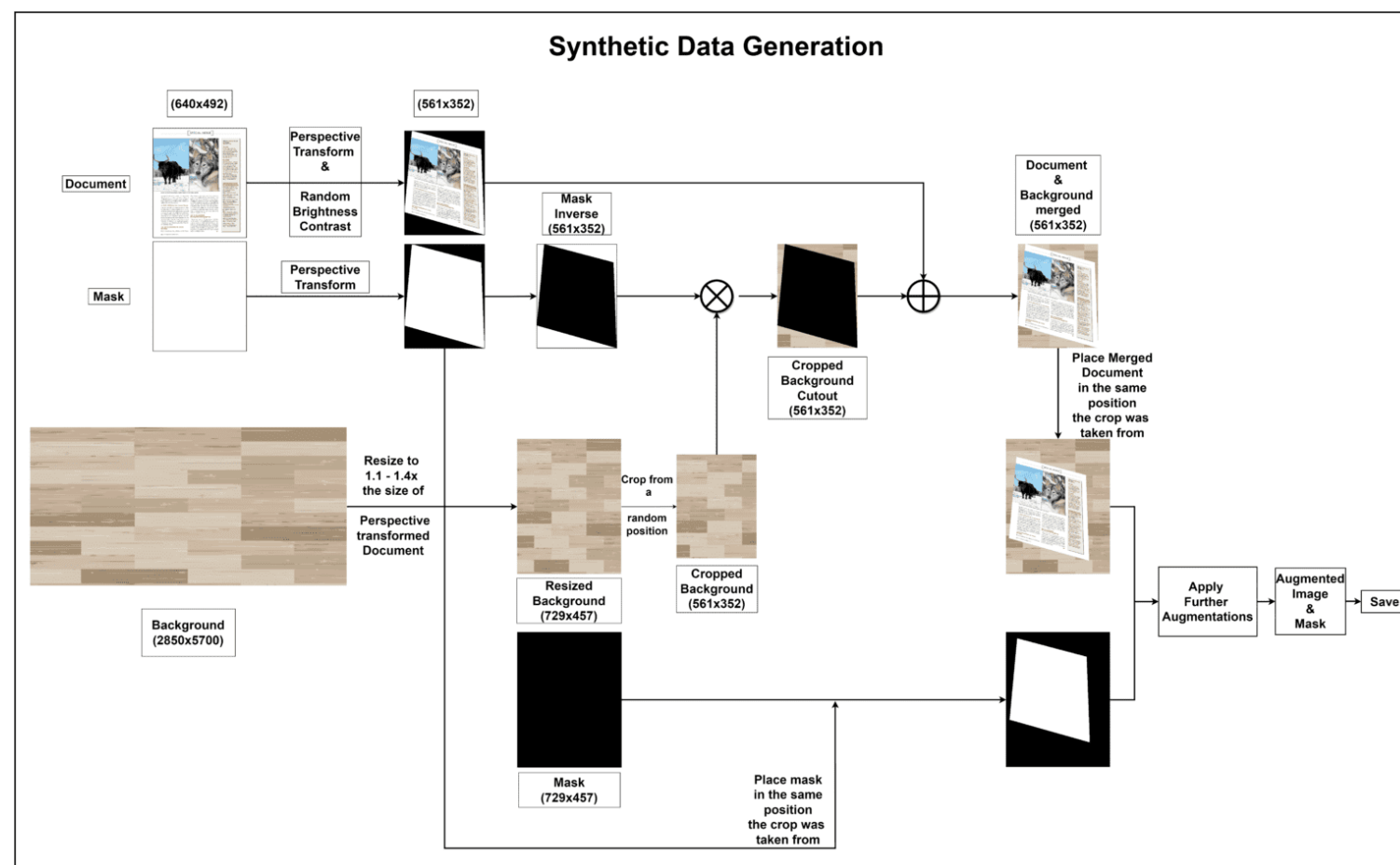
The **dataset** was taken from several sources, such as **DocVQA**, **IAM Handwriting Database**, **Denoising Dirty Documents**, **FUNSD**, **LRDE Document Dataset**, and **SmartDoc QA**. Pre-processing is done by changing the dimension size to 640 (the aspect ratio is maintained) and creating an array to mask each image with the same shape, where the **value** of the array is filled with a **value of 255**.

GENERATE DATASET & APPLY AUGMENTATION

Generate Dataset

The picture beside shows the **process for generating synthetic dataset** using random background from google & using perspective transform for image (The **result of this process is one image & mask pair**). The purpose of this is to **produce background and position variations** in the dataset so that the **model can later be better trained** to recognize various conditions.

Once this process is complete, the **next step is augmentation of results** to replicate **real-world scenarios as closely as possible**. Various augmentations were carried out such as **Horizontal and Vertical Flipping, RandomRotation, Color Jitter, and others**

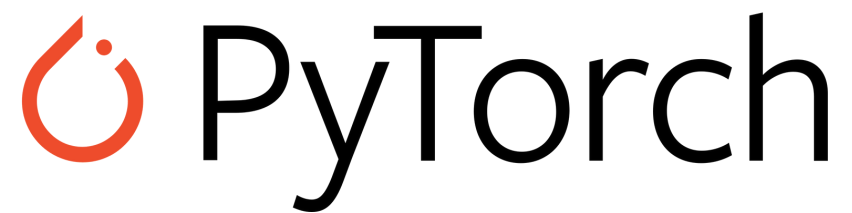




IMPLEMENTATION IN PYTORCH

DOWNLOAD DATASET

Because I didn't implement the dataset preparation process, I just wanted to know the process, so I decided to just download the prepared dataset via Google Drive.



DATA PREPARATION

After the dataset is downloaded, the next step is to do **data preparation**, where several **functions will be made to be able to process the dataset**, such as :

- Accessing the location of the dataset,
- Carry out the transformation process on the training & validation dataset,
- Extract the mask file (because it has 2 channels) to get the background mask & document mask, and
- Create a dataloader for training & validation of image datasets.

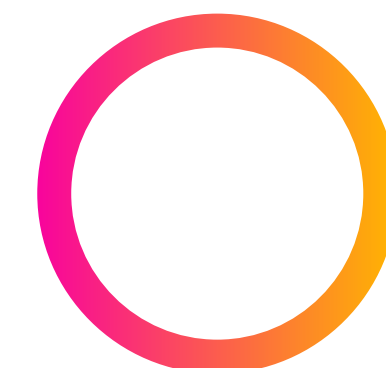
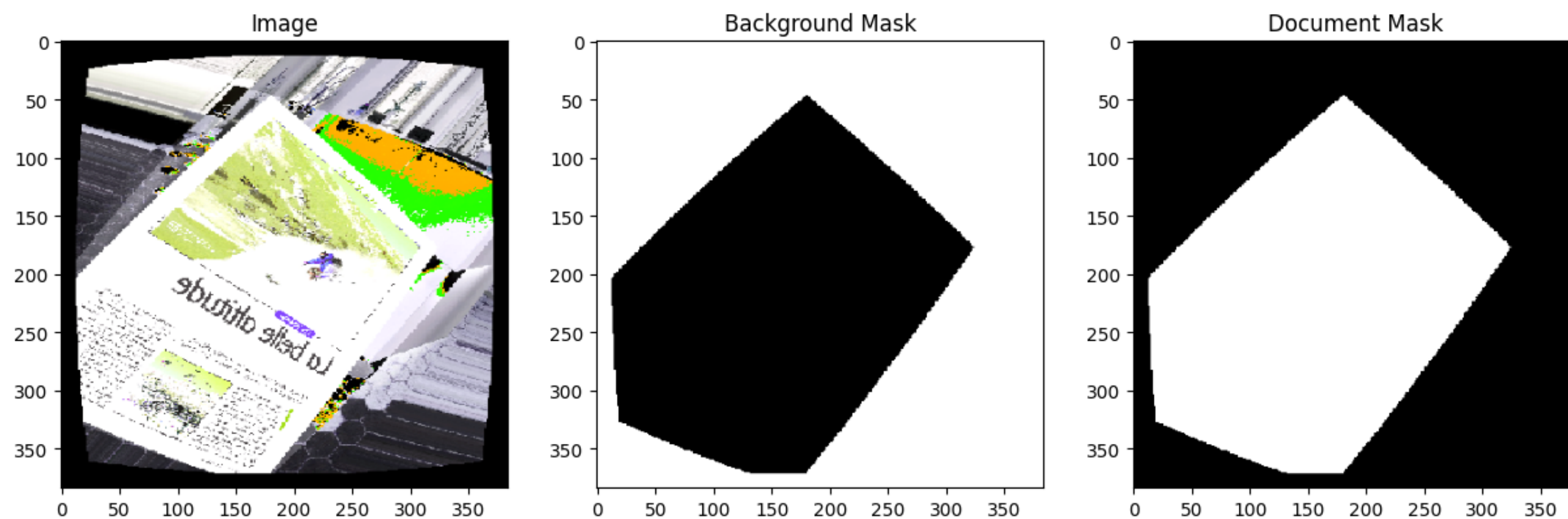
***for more detailed code, can see the ipynb file on repository, because I only explained the outline**



IMPLEMENTATION IN PYTORCH

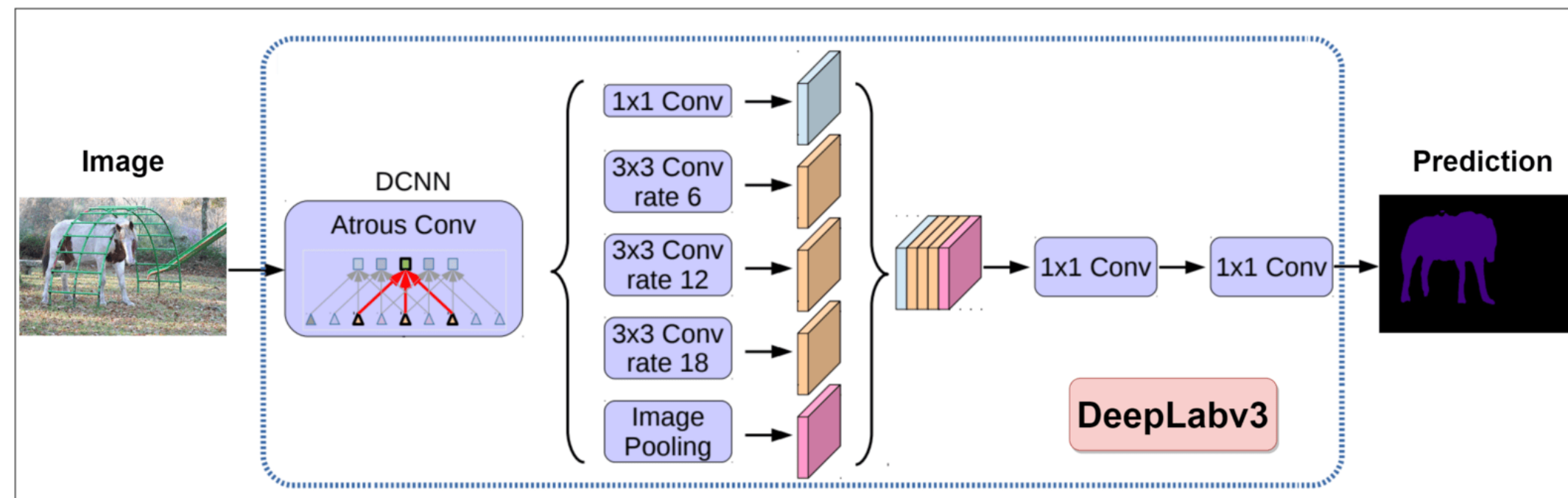
DATA PREPARATION

The following is an example of the results of one of the samples in the dataset after going through the data preparation process



IMPLEMENTATION IN PYTORCH

LOAD PRE-TRAINED MODEL



It's time to enter the series of modeling processes. The **first step** is to **load the pre-trained model**. The **model chosen** for this process is **DeepLabv3 with MobileNetv3Large as the backbone**. The reason for **choosing DeepLabv3 as the model** for this method is because this **deep learning model is more used for semantic segmentation**. In addition, the choice of **MobileNetv3Large as the backbone** is because it has **good performance even though this model has a smaller size**.

After the model is successfully loaded, we must **change the output channels on the output layer** so that the **model can run properly** and **without any errors**. The **effect of this change** will be **remove pretrained weights on last layer**.



IMPLEMENTATION IN PYTORCH

MAKE LOSS CLASS FOR USE IN TRAINING MODEL

Before creating a class for Loss and Evaluation Metric, a **function** is created to **calculate the metric**, because this function will be called on both classes.

In **this function**, there are **two evaluation metric options**, you can use **IoU (Intersection over Union)** or **Dice (F1-score)**. For the **next process**, the **selected option is IoU evaluation metric**.

The next step is to create a **class to calculate the loss in the model**. There are **two loss calculations at once** that are used to calculate the loss, namely:

- **Binary Class Entropy (Pixel-Loss)**. This loss is often **used in image segmentation** to **calculate the difference** between the **predicted pixel value (prediction)** and the **target pixel value (target)**.
- **IoU (Mask-Loss)**. This loss **uses Evaluation Metric** to **calculate the metric value** based on the **predicted mask** and the **target**. The **method of calculation is by subtracting the evaluation metric by 1** to obtain a **value that represents the similarity or accuracy** of the predicted masks.

These two losses are summed after each loss has been obtained as this will result in combining the pixel level differences and mask level dissimilarities, providing a comprehensive measure of the overall error between the predicted and target outputs.



IMPLEMENTATION IN PYTORCH

MAKE EVALUATION METRIC CLASS FOR USE IN TRAINING MODEL

Next is to create a **class for Evaluation Metric**. This class will take into account the **number of classes (num_classes)**, **smoothing factor (smooth)**, and whether to use the Dice coefficient (`use_dice`) for calculations (in this project, the focus was on IoU).

1. **Before entering the metric calculation**, the **convert_onehot function is called** to **convert the predicted value into one-hot encoded** format if it is not already in that format.
2. After that, the **metric_calculation function is called** to calculate the metric value based on the converted prediction and target.

IMPLEMENTATION IN PYTORCH

MAKE TRAIN MODEL PROCESSING FUNCTION & CALL IT

It's time to enter the **training model process**. The first step is to **create a training model function**. There are **several parameters** that can be used, such as `model`, `epoch_num`, `loader`, `optimizer_fn`, `loss_fn`, `metric_fn`, `is_train`, & `metric_name`.

The essence of this function is :

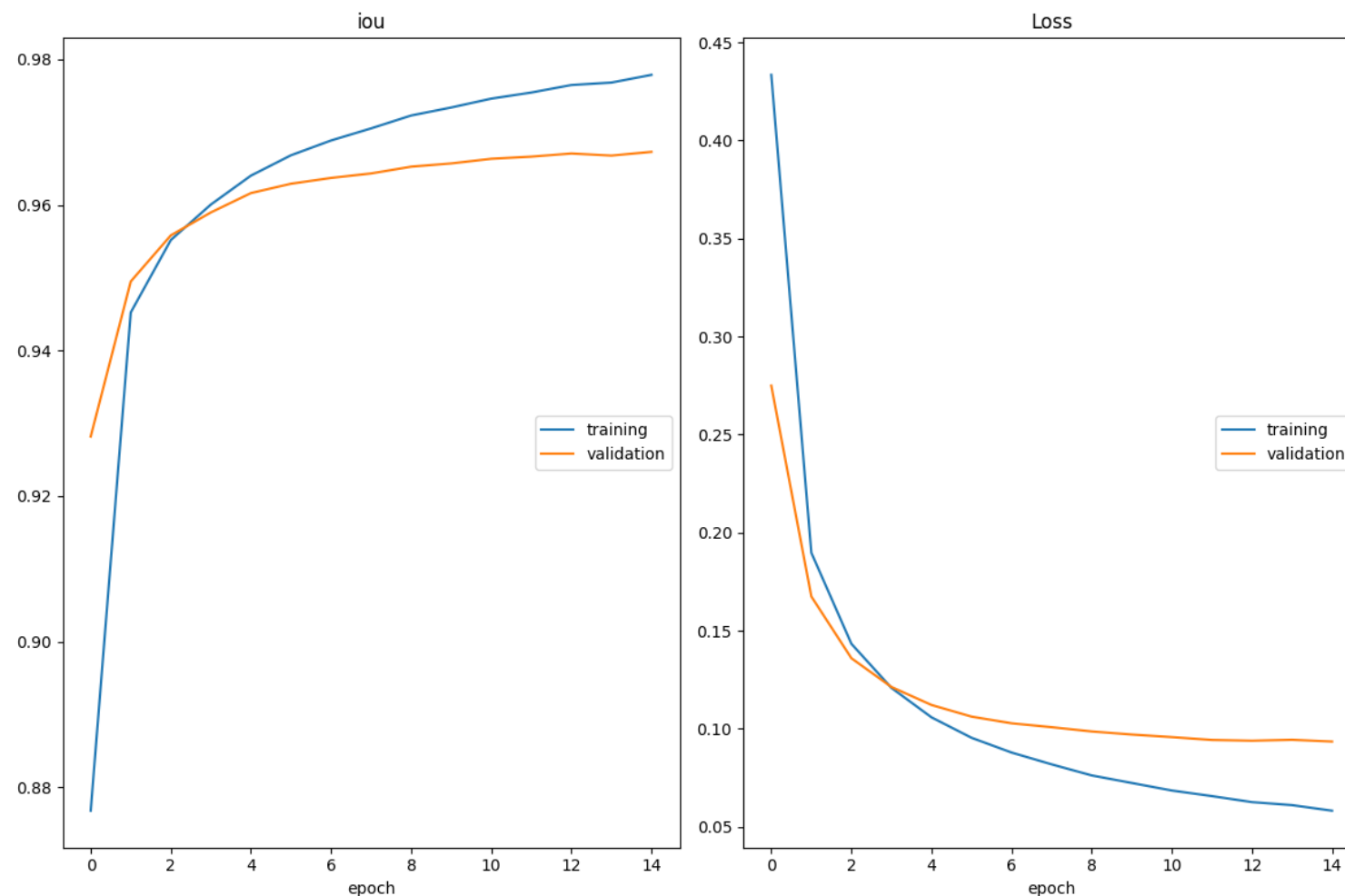
- There is **an instance to keep track of the average loss and metric values during the step**.
- There is **a progress bar during iteration**. Each iteration corresponds to the processing of the data set.
- There is **an option for the training mode**, whether in **Training mode or in Validation (Evaluation) mode**
- The **loss values and metrics are separated from the compute graph**, and the **item values are retrieved**.

After that, an **update is made on the loss and metric values** with the **appropriate values to update the internal status**.

- After the **loop is completed**, the **average loss values and metrics for that step are calculated**.

IMPLEMENTATION IN PYTORCH

MAKE TRAIN MODEL PROCESSING FUNCTION & CALL IT

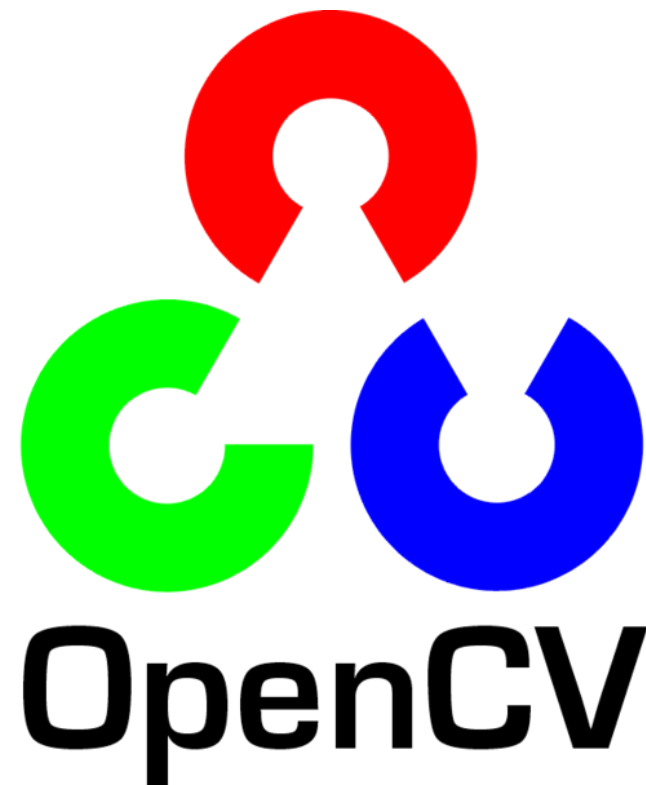


iou				
training	(min:	0.877,	max:	0.978, cur: 0.978)
validation	(min:	0.928,	max:	0.967, cur: 0.967)
Loss				
training	(min:	0.058,	max:	0.433, cur: 0.058)
validation	(min:	0.094,	max:	0.275, cur: 0.094)

After the **function is created**, it's time to **call each function that has been created** by entering the appropriate parameters. The following is the **final result of the training** model process :

1. **Maximum value** obtained which is stored **for 15 iterations on IoU (Evaluation Metric)** for **training (0.978) & validation (0.967)**
2. **Maximum value** obtained which is stored **for 15 iterations in the Loss Function** for **training (0.058) & validation (0.094)**
3. In these results, it can be seen that the **maximum value obtained for Evaluation Metrics & Loss** is what will be **saved** even though the **next iteration can produce a lower value**.
4. Be careful when **determining iterations for this modeling process**, because the **graph shows that with only 15 iterations**, the **validation results (orange line)** are **slowly stabilizing** and **may gradually increase again**, resulting in the **potential for overfitting**.

IMPLEMENTATION IN OPENCV



LOAD MODEL

Load the model that has been trained to do a test sample

TRANSFORM IMAGE BEFORE BEING PROCESSED BY THE MODEL

The image to be segmented must be transformed first. Therefore, a function is created to facilitate the transform process.

***for more detailed code, can see the ipynb file on repository, because I only explained the outline**



IMPLEMENTATION IN OPENCV


MAKE FUNCTION FOR FIND CORNER POINTS FROM DETECTED CONTOUR

In the segmentation process, this function will be **called to rearrange the four points in the standard order** because the **detected corner sequence will be unconventional**. The core process of this function is :

- **Changing the corner** that has been **detected into one dimension** according **to its designation**.
- For the **top left point (smallest sum point)** and **bottom right (largest sum point)**, the **x and y coordinates of each point** are **added together**, the **final result is a 1D np.array**.
- For the **top right point (smallest point of difference)** and **bottom left (biggest point of difference)**, the **difference between the x and y coordinates** of each point is **calculated**, the **result is a 1D array**.

MAKE FUNCTION FOR FIND DESTINATION COORDINATES

In the segmentation process, this **function can calculate the maximum width, maximum height, and generate the coordinates of the destination angles** which will be used to **transform the found object or shape into a rectangle** with **predetermined dimensions**.



IMPLEMENTATION IN OPENCV

MAKE FUNCTION FOR SEGMENTATION PROCESSING

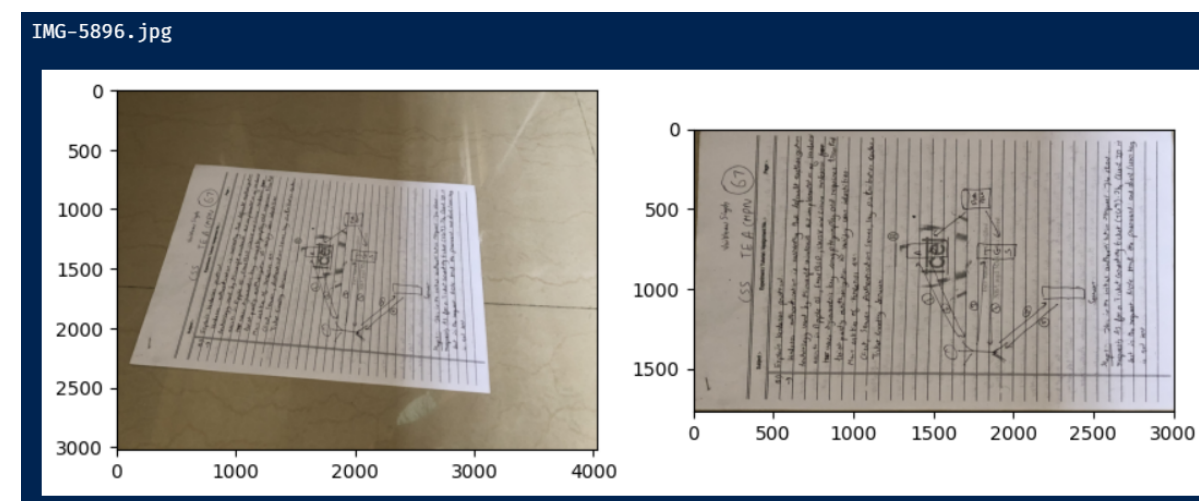
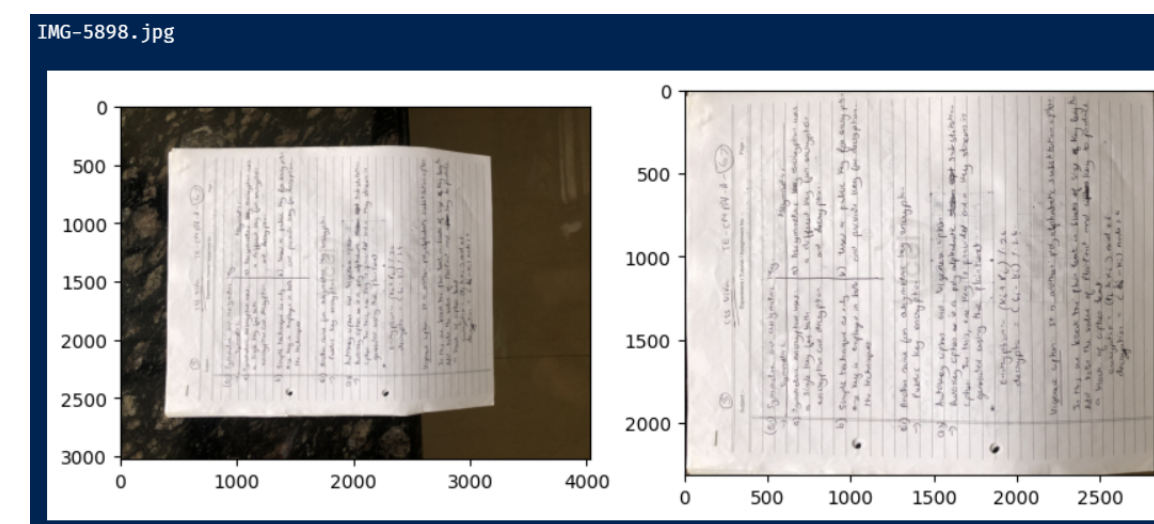
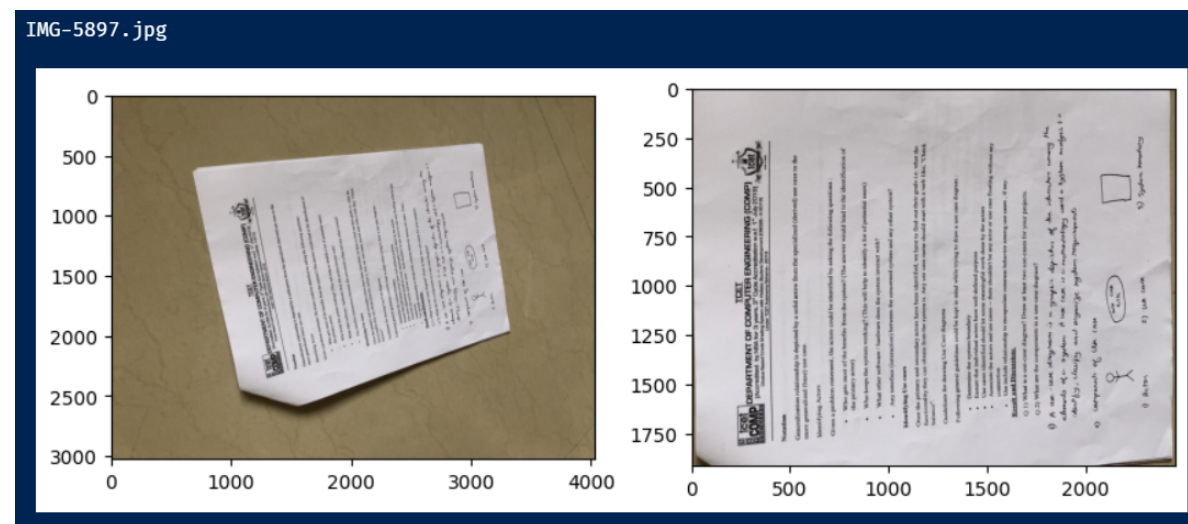
All functions that have been **created before**, its **implementation in this function**. The following are some important explanations/stages in this segmentation process :

- **Resize Image input** to standardize the dimensions **for processing with the trained model**.
- Do a **transform on the image** with the function that was created before
- After being transformed into a Torch tensor, the next step is to **prepare the dimensions to match the input dimensions in the model**
- Perform **forward pass on transformed image** with trained model (**variable out**). After that, **remove the unnecessary variable** (image_model)
- **Variable out from the trained model**, are **processed to obtain the segmentation mask**, then create an **extended array in which the segmentation mask is centered**.
- Do the **edge detection process with the canny method**. Then **extract the contours from the edge detection results** and **simplify the previously detected contours to only a few corners** with the Ramer-Douglas-Peucker algorithm.
- The **angular coordinates need to be corrected** and **remapped** to match the original image.
- Next, it checks that the **document corner is within the image bounds** and **adjusts the corner and image by adding padding if necessary to ensure the document is completely contained within the image**.
- **Perform a perspective transformation operation** on the input image **based on the detected angle**, resulting in an **output image** that represents the **document in a corrected and aligned form**.

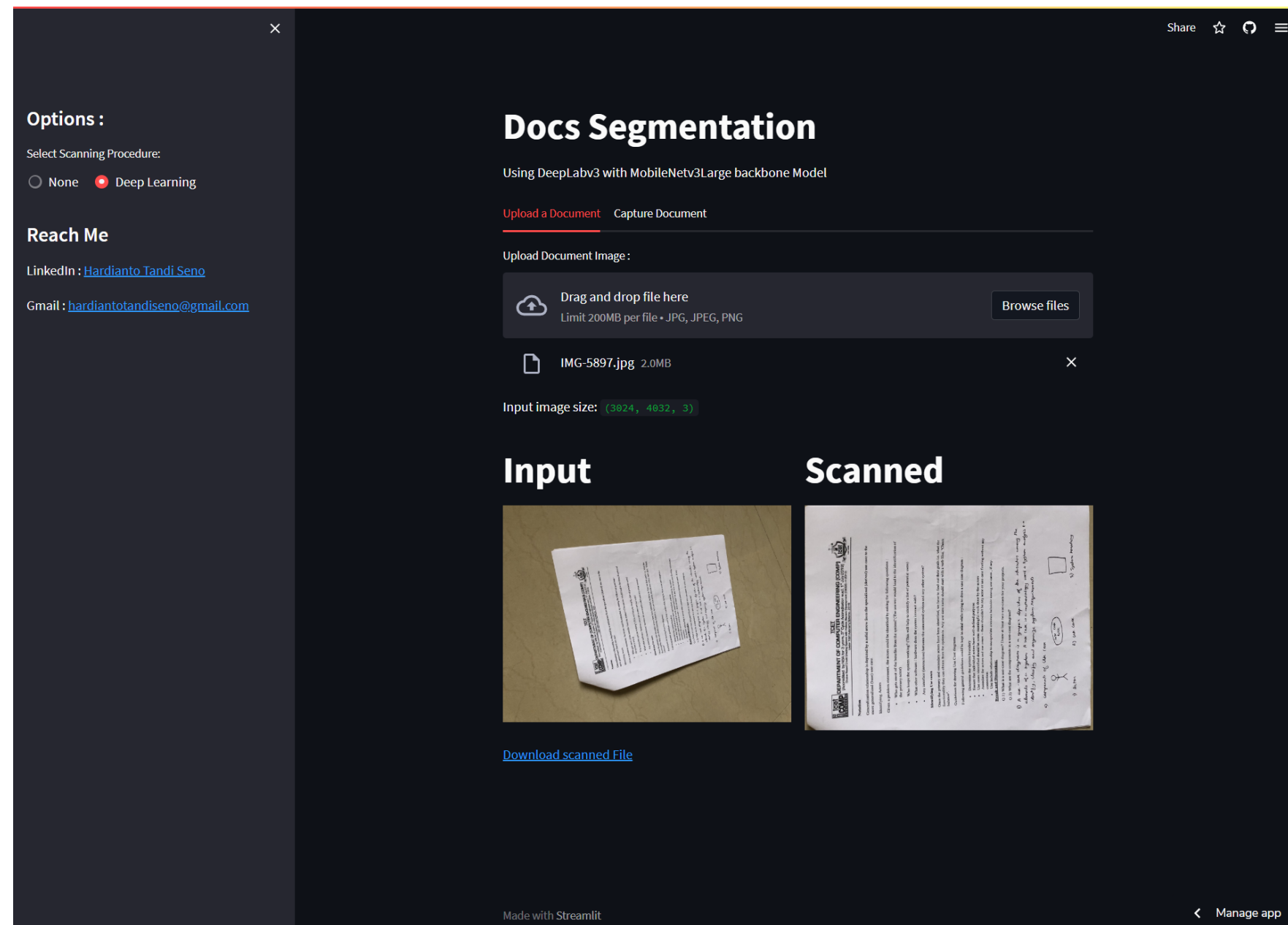
IMPLEMENTATION IN OPENCV

TEST SEGMENTATION RESULT

At this stage, the **segmentation processing function** will be called with the **trained model & OpenCV** and the **results of the segmentation process** will be **displayed on several test images** using matplotlib library. The following are **some of the results of document segmentation** in images :

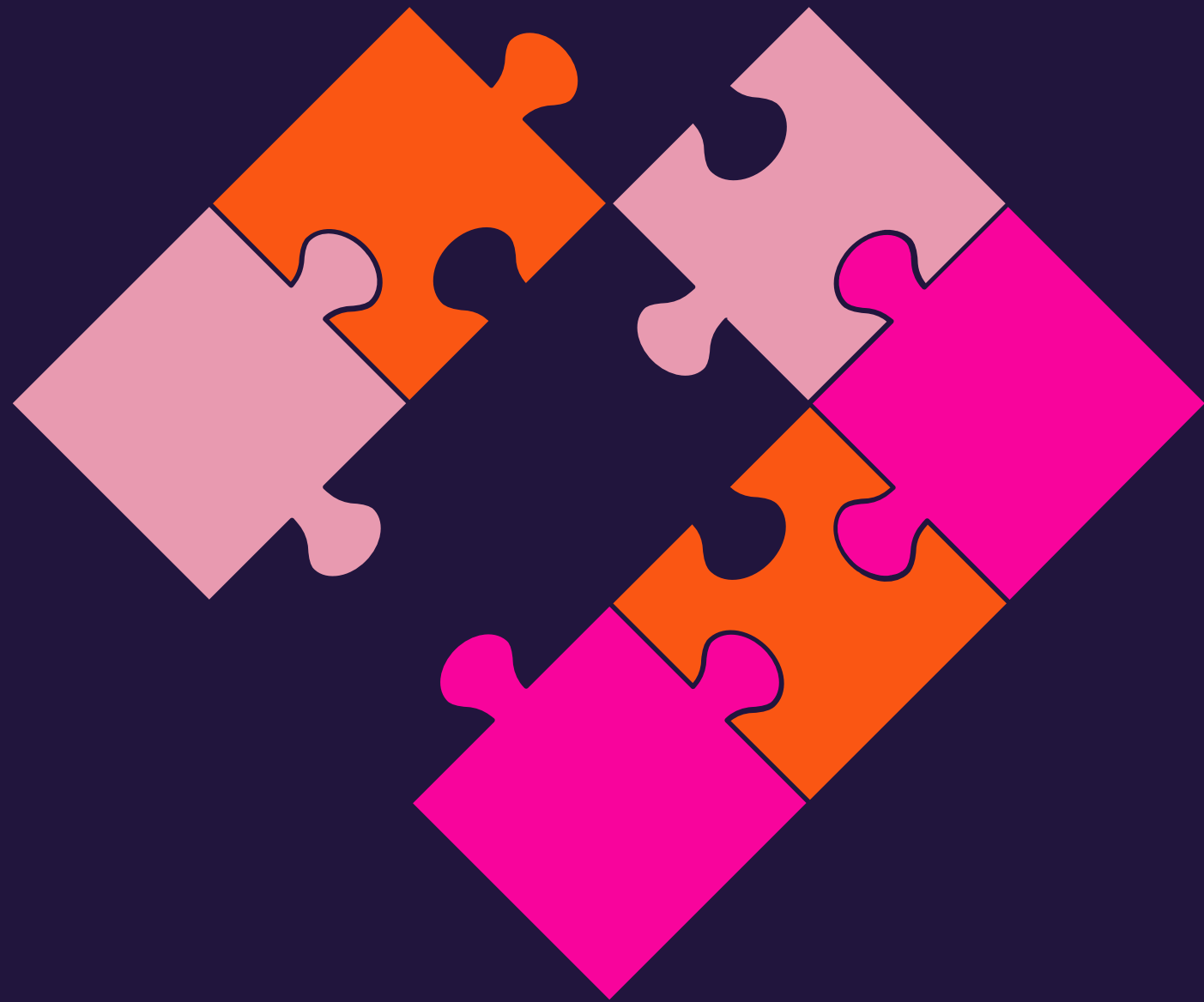


DEPLOYING AT STREAMLIT



TEST ON STREAMLIT

In order for this **project to be used by others**, it needs to be **deployed to streamlit**. For the process itself, it only **includes and adjusts each segmentation process (code)** with the help of the **Trained Model and the OpenCV library** into the **Streamlit structure itself**.



CHALLENGES & CONCLUSION





CHALLENGES

- Even though I only **downloaded the previously prepared dataset**, I see that the **data preparation process** is **quite complex**, because I have to create a **synthesis dataset** from **random backgrounds from Google** and **various perspective transforms** for images. It doesn't stop there, this stage continues until the **augmentation process** to get a **very diverse image dataset** that is **useful for the effectiveness of a model** that can **read various conditions**.
- **Every array manipulation process** in this project must be **considered**, because I saw quite **a lot of array manipulation** being done for the **document segmentation process in images**.





CONCLUSION

- This project will be useful for people who take photos of a document that still has a **background behind it**, but only want to **leave specific parts of the page**, so that the **file can be used for important purposes**, whether it's **administration and so on**.
- Every plot in this project must be considered very well, because the plot is quite long and each process is quite complex.
- **Dataset diversity is one of the important things** in this project. This is because the **various conditions trained by the model** will make the **model able to recognize each condition** when we try to implement it by trying to input images to be able to detect documents in these images.





THANK YOU



Hardianto Tandi Seno



hardiantotandiseno@gmail.com

