PROGRESS REPORT:

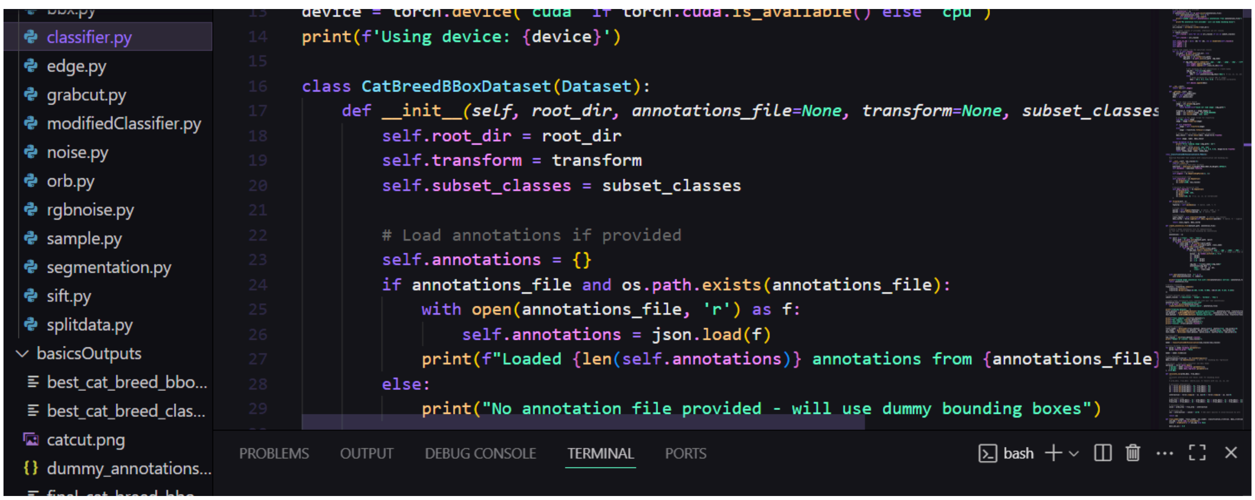
Basic CNN Architectures:

Studied through how basic CNNs work as I was not familiar with various architectures and how image processing happens pixel by pixel with the gradient descent and layer wise convolutions.

Looked at CNN,R-CNN, Faster R-CNN,The YOLO model,Resnets etc.

Training a classifier:

Made a classifier to detect cat breeds from a small cat breed dataset that included various feature factors of color,ear size,body size,patterns and body shape which served as the pixel feature maps to distinguish between the breeds to get a small hands on experience with a CNN.



Medical Imagery with XAi:

Tested out a dataset of ultrasound images to detect cancerous and non cancerous cases showing the features being mapped through various differences in the structure of the human organ. Also tried to implement Grad Cam to find out what part of the image the CNN used to infer its predicted value.

OpenCV Basics:

Started out with basics of open cv on how it's used for images in order to handle datasets and images as a whole for processing. Looked at how masking and pixels are changed overall by playing around with color maps and other opencv methods.

```python
import numpy as np
import matplotlib.pyplot as plt

# Load an RGB image (or generate a synthetic one)
image = np.full((256, 256, 3), 128, dtype=np.uint8)  # Gray background
cv2.circle(image, (128, 128), 60, (0, 255, 0), -1)    # Add a green circle

# Parameters for Gaussian noise
mean = 0
stddev = 25

# Generate Gaussian noise for each channel
noise = np.random.normal(mean, stddev, image.shape).astype(np.float32)

# Add noise and clip the result
noisy_image = image.astype(np.float32) + noise
noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
```

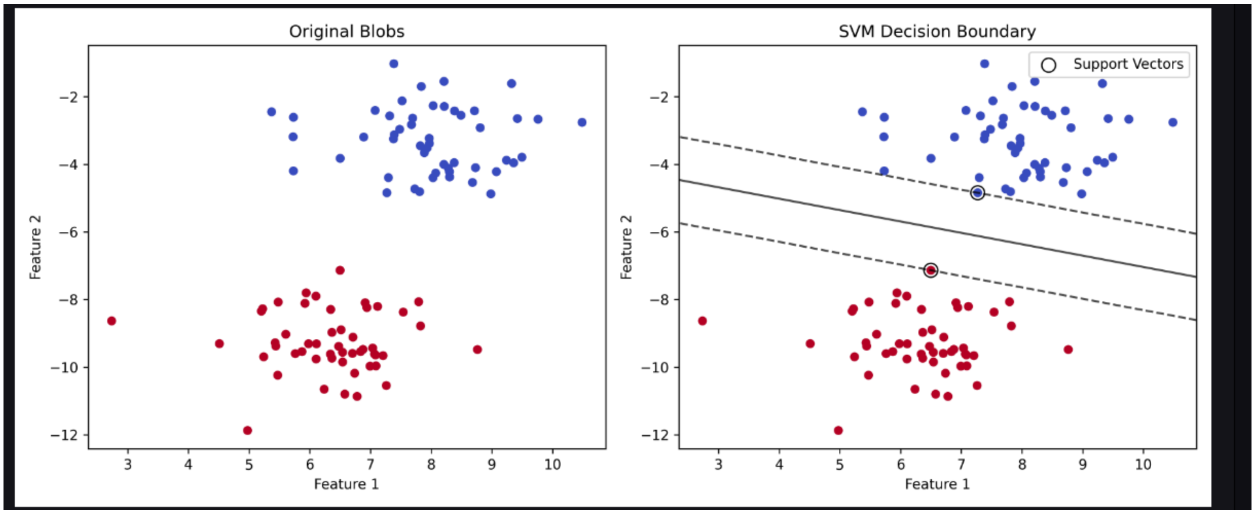
Traditional Algorithms:

Implemented SIFT and SURF along with ORB to get a base understanding of traditional similarity index algorithms with how they use the maximum feature points and then cluster same values to detect similarities between the same images in different scenes or different orientations.

Tested on the previously mentioned cat dataset where the ears and body features were the most prominent ones to be detected. Also looked at how various algorithms improve over others by skipping unwanted steps that reduce complexity and where they can be used.

basics > sift.py > ...

```python

tGray = cv2.cvtColor(train, cv2.COLOR_BGR2GRAY)
qGray = cv2.cvtColor(query,cv2.COLOR_BGR2GRAY)

plt.figure(3)
plt.imshow(cv2.cvtColor(tGray,cv2.CV_32S))
plt.title("gray train img")
plt.figure(4)
plt.imshow(cv2.cvtColor(qGray,cv2.CV_32S))
plt.title("gray query img")


sift = cv2.SIFT.create()

kTrain, dTrain = sift.detectAndCompute(tGray,None)
kQuery, dQeury = sift.detectAndCompute(qGray,None)

```



Advanced Image Processing:

Looked into blurring, noise and various other processing techniques that need to be applied to images before and after processing for a model which can be input or output images.

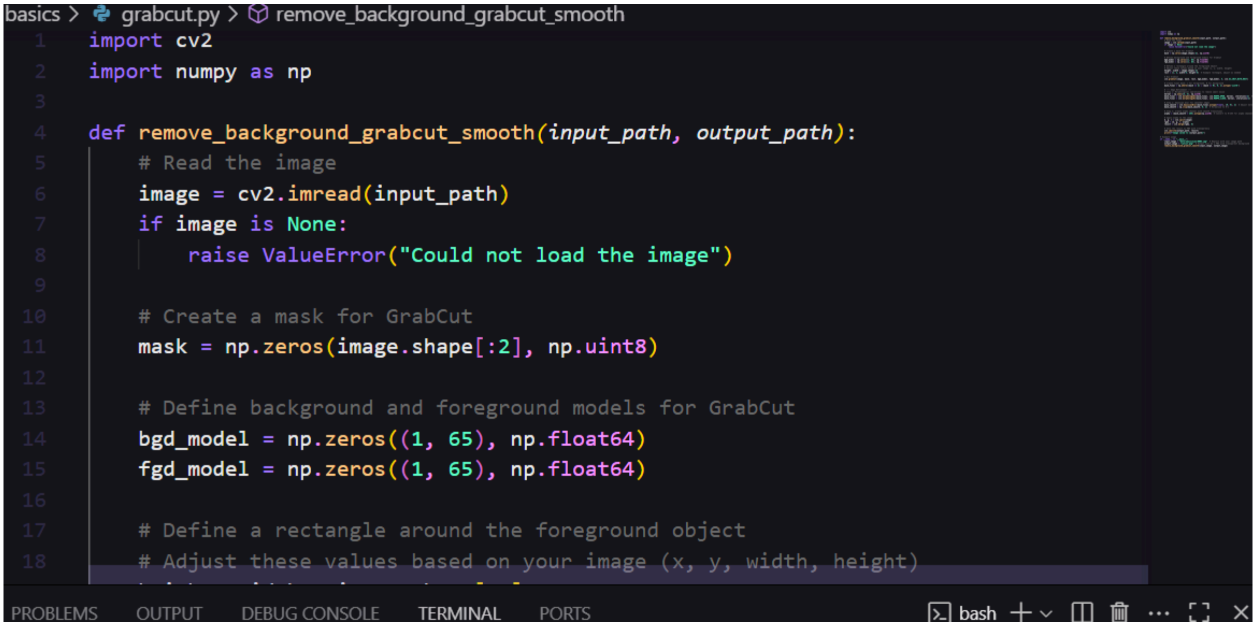
Since models use a single form factor for all image inputs and they give a standard output the images need to be processed in a way that caters all of this.

Also looked into basic methods of edge detection which involved their math including hysteresis and other leveling factors that can affect how much the algorithm needs to focus on various features of an image to show an output like lower edge factor etc through threshold values.

ics > edge.py > ...

```
import matplotlib.pyplot as plt
import numpy as np
# Load image in grayscale
image = cv2.imread("edgesample.jpg", cv2.IMREAD_COLOR_RGB)
#cv2.circle(image, (128, 128), 60, (0, 255, 0), -1)    # Add a green circle

mean = 0
stddev = 25
noise = np.random.normal(mean, stddev, image.shape).astype(np.float32)

noisy_image = image.astype(np.float32) + noise
noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
# Apply Gaussian Blur to reduce noise
blurred = cv2.GaussianBlur(noisy_image, (5, 5), 1.4)
gray = cv2.cvtColor(blurred, cv2.COLOR_RGB2GRAY)
# Apply Canny Edge Detection
edges = cv2.Canny(blurred, 50, 150)
contours, _ = cv2.findContours(edges,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

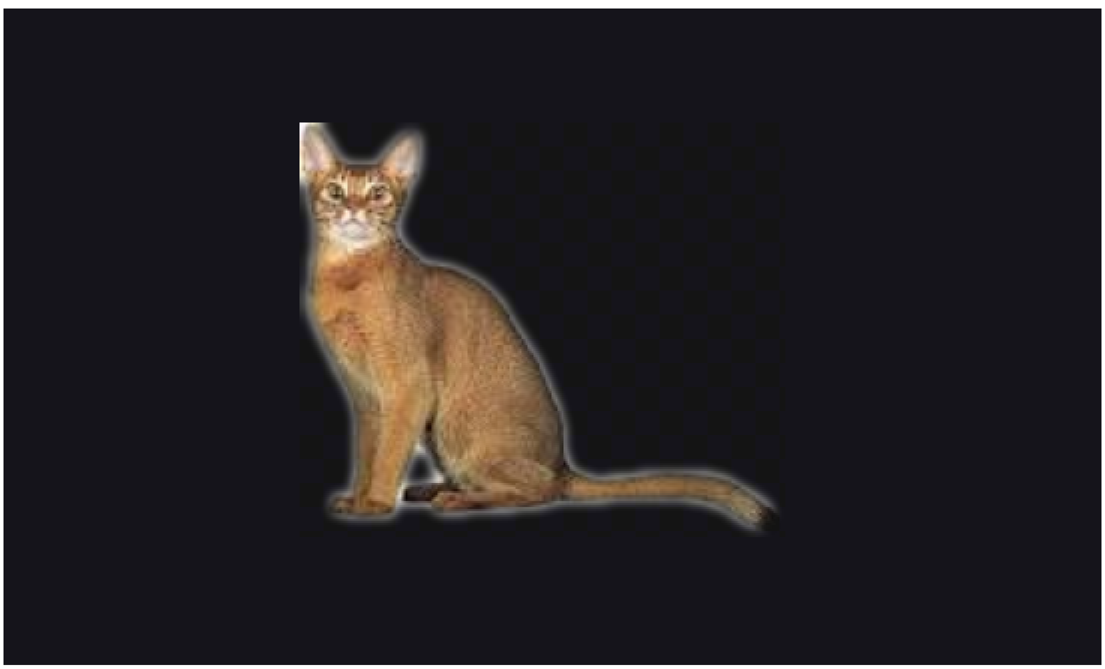
OBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash



basics > grabcut.py > remove_background_grabcut_smooth

```python
import cv2
import numpy as np

def remove_background_grabcut_smooth(input_path, output_path):
    # Read the image
    image = cv2.imread(input_path)
    if image is None:
        raise ValueError("Could not load the image")

    # Create a mask for GrabCut
    mask = np.zeros(image.shape[:2], np.uint8)

    # Define background and foreground models for GrabCut
    bgd_model = np.zeros((1, 65), np.float64)
    fgd_model = np.zeros((1, 65), np.float64)

    # Define a rectangle around the foreground object
    # Adjust these values based on your image (x, y, width, height)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash



WEEK 2:

Annotations:

Looked into different annotation formats that involve COCO,YOLO etc. How to make annotations and how to process them in a model to get ground truth values. Also looked into how to cater for data that does not have annotations entirely or have partial annotations that need to be improved. Bounding boxes and Polygonal annotations along with masks for segmentation

Object Detection:

Tried to train a model on satellite images that contained cars,planes and boats to classify the detected object by cross checking manual annotations and active learning to enhance the dataset iteratively since the whole dataset was not entirely annotated.

Looked into various object detection techniques which involved YOLO models, R-CNNs and more deeper end to end models. Studied through their use cases on where and when to use a model like real time efficiency and accuracy basis.

tations > model.py > ...

```
from ultralytics import YOLO

model = YOLO('yolov8n.pt')


model.train(
    data="dataset/dataset.yaml",
    epochs=15,
    batch=5,
    imgsz=640,
)

metrics = model.val(
    data = "dataset/dataset.yaml",
    split="test",
)
```

notations > pipeline.py > ...

```python
import os
import json

# Paths - update these as per your setup
coco_json_path = 'val.json'
images_dir = 'dataset/images/val'
labels_dir = 'dataset/labels/val'

os.makedirs(labels_dir, exist_ok=True)

# Load COCO JSON
with open(coco_json_path, 'r') as f:
    coco = json.load(f)

# Map image IDs to file names and dimensions
image_id_map = {}
for img in coco['images']:
    image_id_map[img['id']] = {
```

Model Training Variants:

Transfer Learning
Self-supervised learning
Semi-supervised learning
Active learning
Re-inforcement learning

Projects chosen to complete in due time:

OCR summarizer - Uses a 3 model pipeline with a detection model, segmentation model and per character classifier to then feed back to a global map with a library to reassemble words and generate a pdf or word document.

Reinforcement Learning Simulation - Requires an open source simulator to train a model which utilizes computer vision such as drone navigation or a model that learns to track various objects in a live scene to label and detect them showing information

GAN - A model to use the GAN architecture and generate basic 2d images since 3d images and datasets are hard to obtain and train on my own machine.

Attached Image For Reference:

Some things that I could not include in the report are mentioned in this summary.

CNN architectures
End to End models.

→ Need to review math behind all major tasks.

Classic algorithms
SIFT, SURF, ORB
Image alterations.
Rotation, Blur, Noise, De-noise
Edge detection, Background removal.
Validation parameters.
Accuracies and scores, for fine tuning.
Data Cleaning → duplicates, bad samples etc.
Data Preprocessing →
Data Augmentation → } scaling, rotation, noise, blur.
Annotations
↳ Classification, (labels)
↳ Detection (Bounding boxes)
↳ Segmentation (Masking, pixel level masks)
} Basic, COCO, Yolo

Active Learning
Self supervised Learning
Transfer Learning.
GANs.

Projects chosen
1. OCR → working on. (Detect, Segment, Classify)
2. Reinforcement Learning through a simulator. (Detection + Segmentati
3. 2d image generation through GAN. (Comparing equilibrium over generator and detect

xtras → Go Lang, Flutter, LLMs / Transformers (need to look at).