Block B: Computer Vision & Robotics

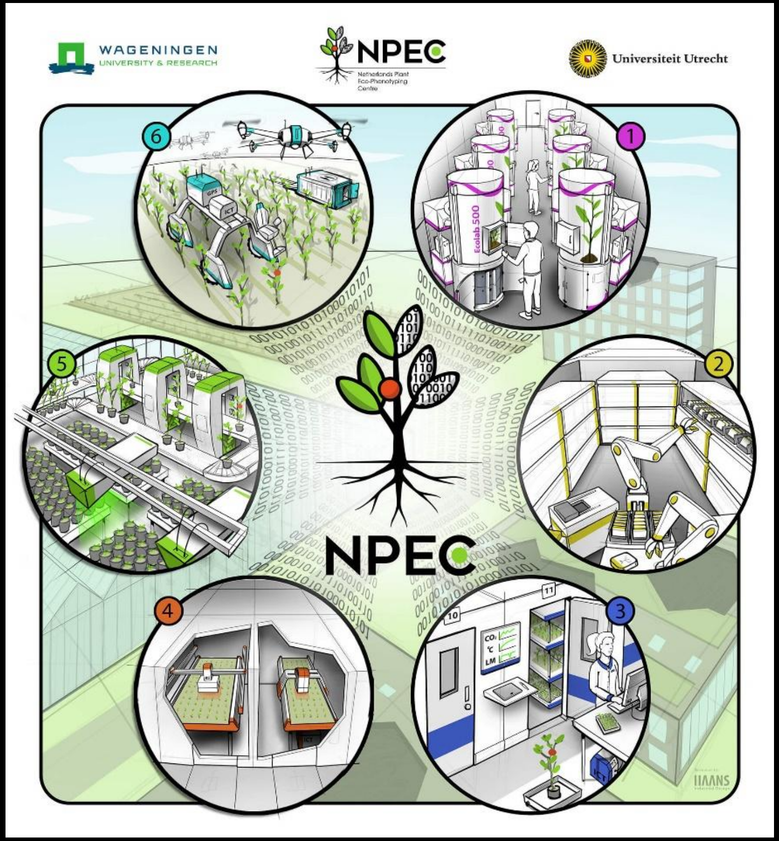
Monika Stangenberg 231648

Problem Overview

Client: Netherlands Plant Eco-phenotyping Centre

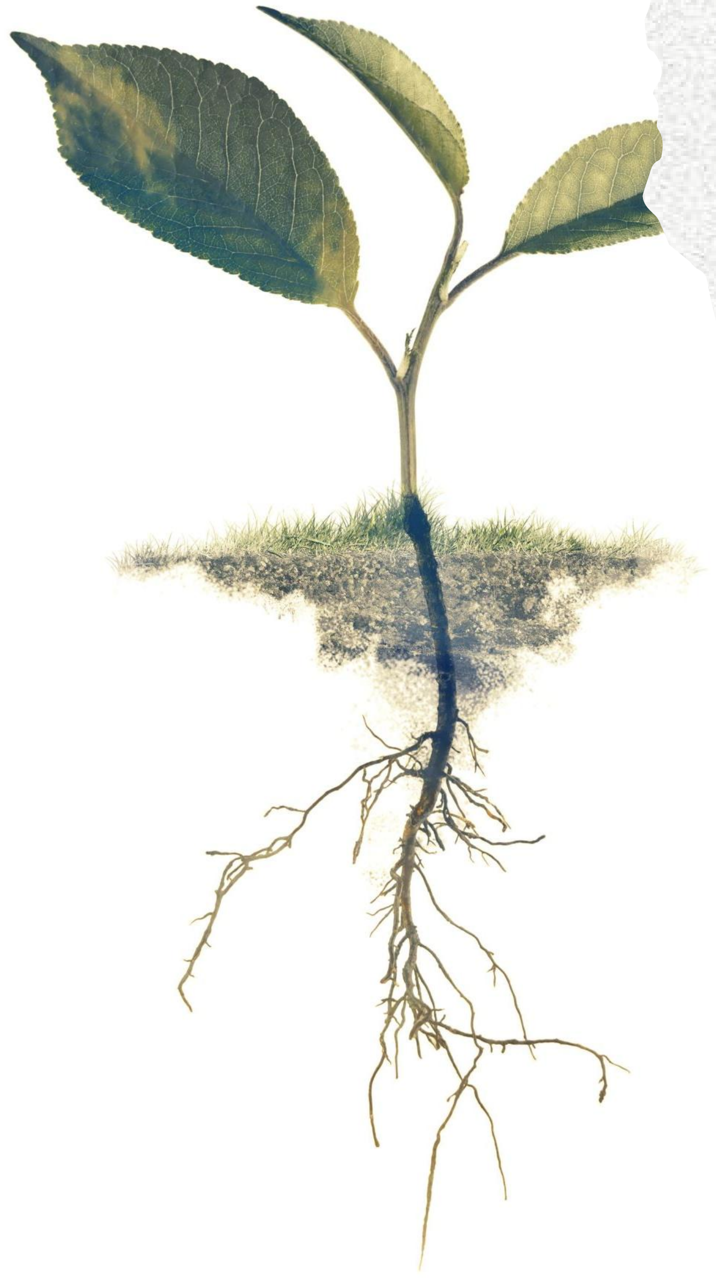
Automatic segmentation of plant roots from images provided by NPEC

Precisely controlled robot to perform in-vitro inoculation of plants with microbes or synthetic communities


WAGENINGEN UNIVERSITY & RESEARCH
NPEC
Netherlands Plant Eco-Phenotyping Centre
Universiteit Utrecht
1
2
3
4
5
6
NPEC
Ecolab 500

Proposed AI-Powered Solution

- Implementing CV pipeline which combines image segmentation techniques and deep learning
- Training a model which identifies plants’ roots from images provided by the client.
- Developing and benchmarking PID and RL controllers to precisely target plant roots using robotics pipettes.
- Integrating computer vision outputs to map root tips’ (pixel coordinates) to robotic arm positions.

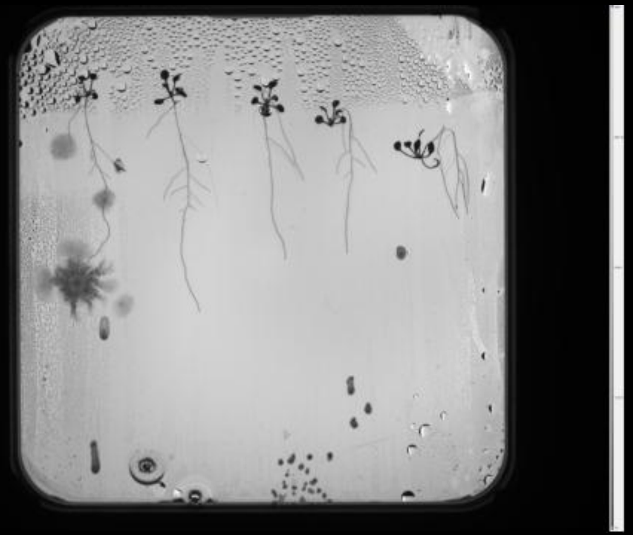


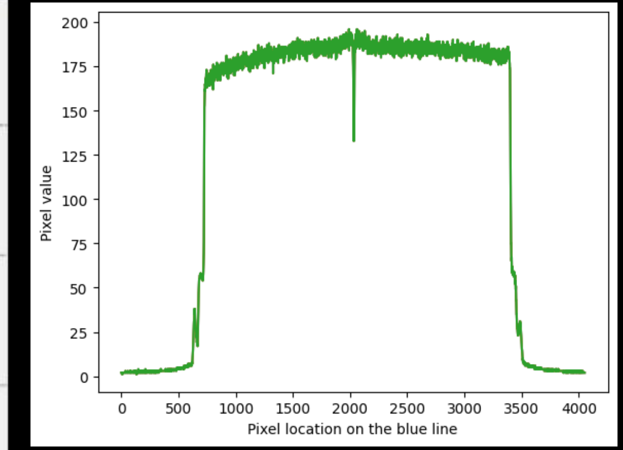
Benefits for NPEC

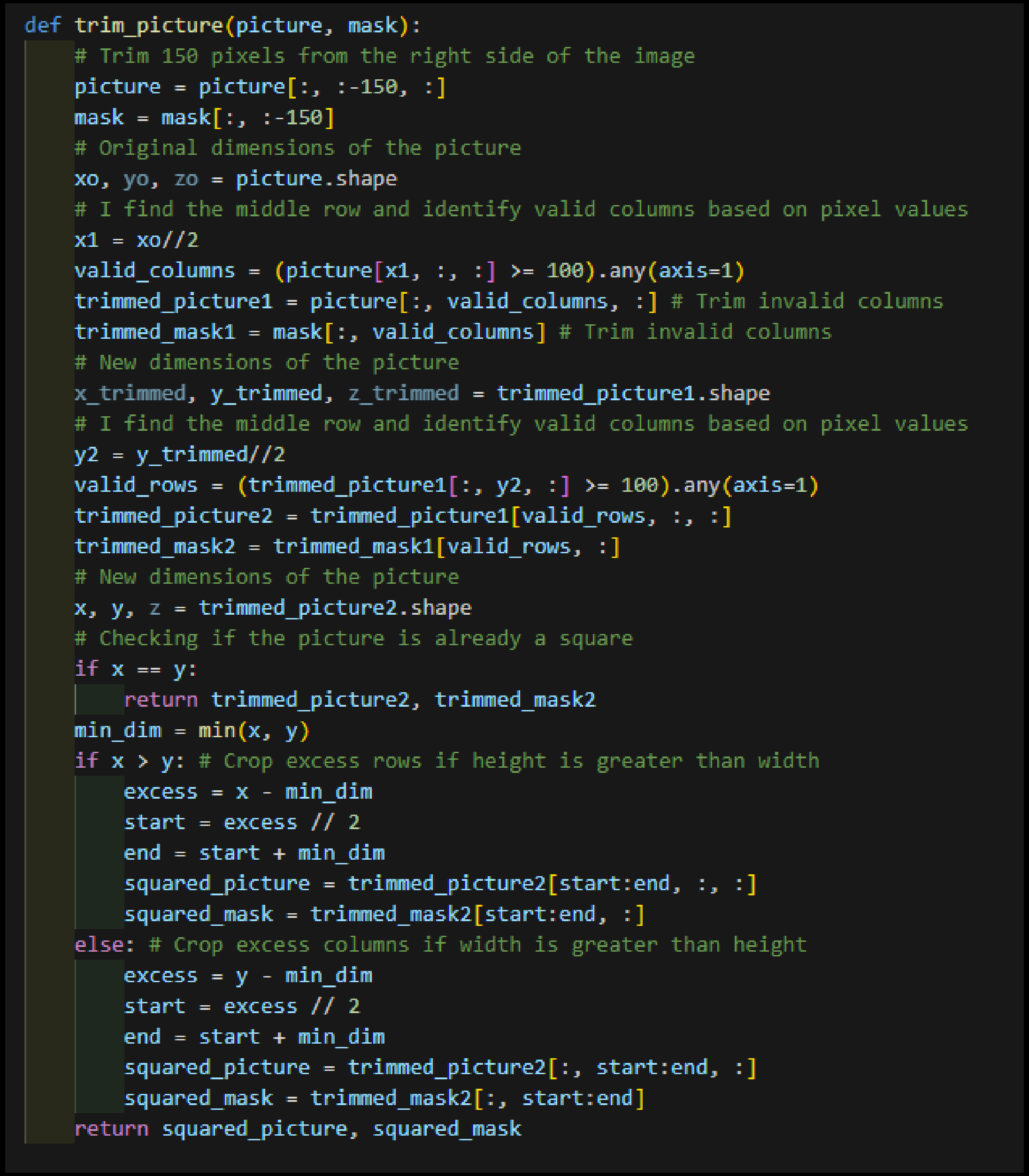
- Help with identifying and tracking root growth.
- Automating process of measuring plants’ roots, which is originally very time consuming.
- Precise and automated root targeting by robot (Hades).

Pre-processing

- Trimming 150 pixels form right side, because images have white strip there
- Using middle x value to find valid columns (based on pixel values)
- Using middle y value to find valid rows (based on pixel values)
- Checking if the image is square, if not cropping excess rows/columns to make the image square



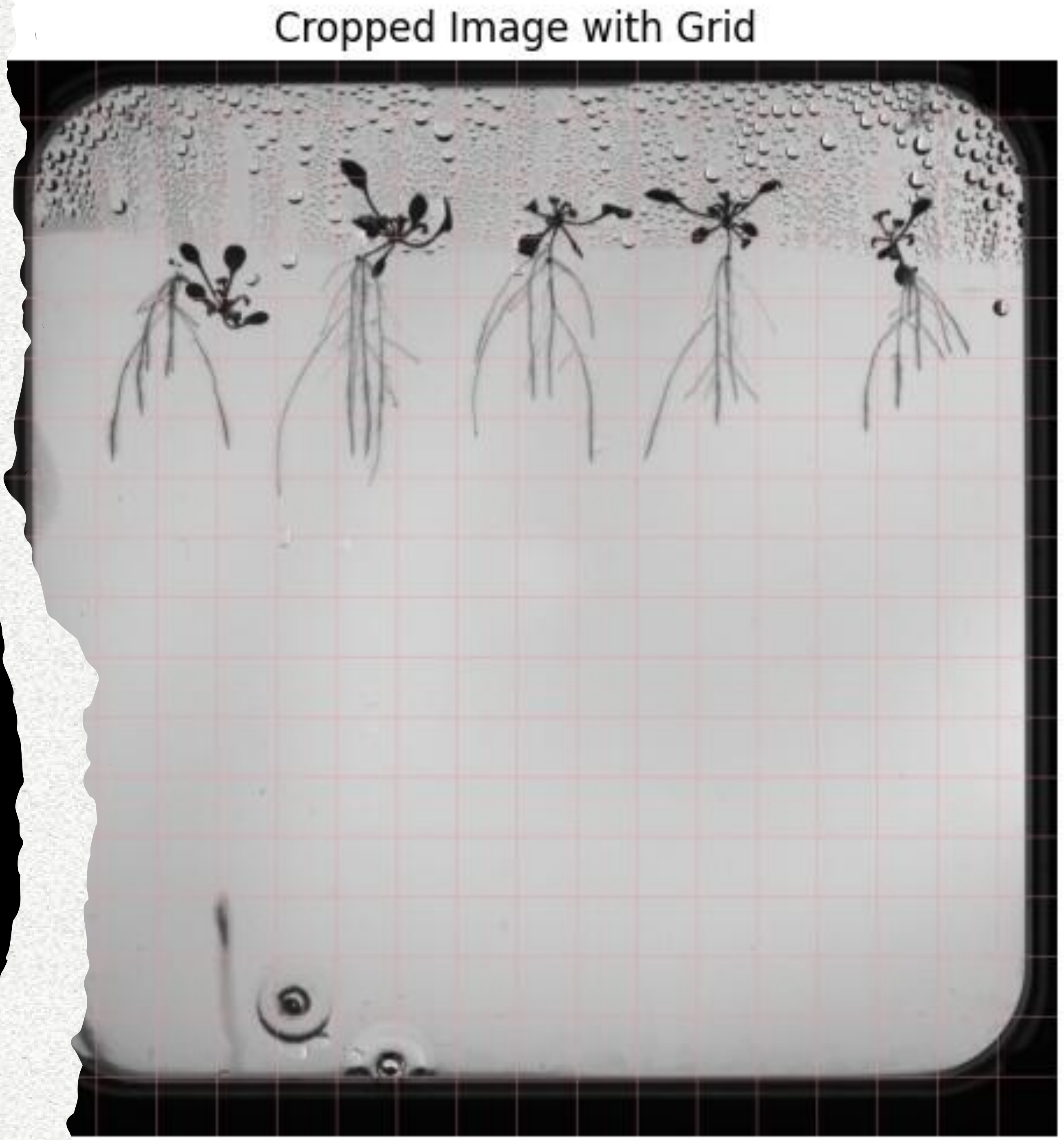




```
def trim_picture(picture, mask):
    # Trim 150 pixels from the right side of the image
    picture = picture[:, :-150, :]
    mask = mask[:, :-150]
    # Original dimensions of the picture
    xo, yo, zo = picture.shape
    # I find the middle row and identify valid columns based on pixel values
    x1 = xo//2
    valid_columns = (picture[x1, :, :] >= 100).any(axis=1)
    trimmed_picture1 = picture[:, valid_columns, :] # Trim invalid columns
    trimmed_mask1 = mask[:, valid_columns] # Trim invalid columns
    # New dimensions of the picture
    x_trimmed, y_trimmed, z_trimmed = trimmed_picture1.shape
    # I find the middle row and identify valid columns based on pixel values
    y2 = y_trimmed//2
    valid_rows = (trimmed_picture1[:, y2, :] >= 100).any(axis=1)
    trimmed_picture2 = trimmed_picture1[valid_rows, :, :]
    trimmed_mask2 = trimmed_mask1[valid_rows, :]
    # New dimensions of the picture
    x, y, z = trimmed_picture2.shape
    # Checking if the picture is already a square
    if x == y:
        return trimmed_picture2, trimmed_mask2
    min_dim = min(x, y)
    if x > y: # Crop excess rows if height is greater than width
        excess = x - min_dim
        start = excess // 2
        end = start + min_dim
        squared_picture = trimmed_picture2[start:end, :, :]
        squared_mask = trimmed_mask2[start:end, :]
    else: # Crop excess columns if width is greater than height
        excess = y - min_dim
        start = excess // 2
        end = start + min_dim
        squared_picture = trimmed_picture2[:, start:end, :]
        squared_mask = trimmed_mask2[:, start:end]
    return squared_picture, squared_mask
```

Pre-processing Patchifying

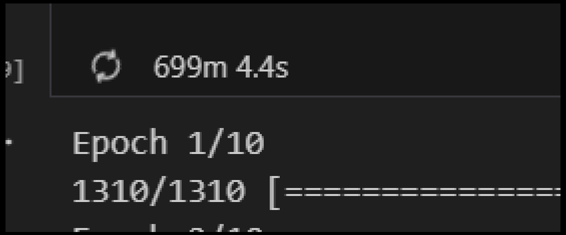
- patch size = 256
- I used padding to make sure that dimensions of all images and masks are divisible by patch size
- I save patched images with unique filename (001, 002, …)



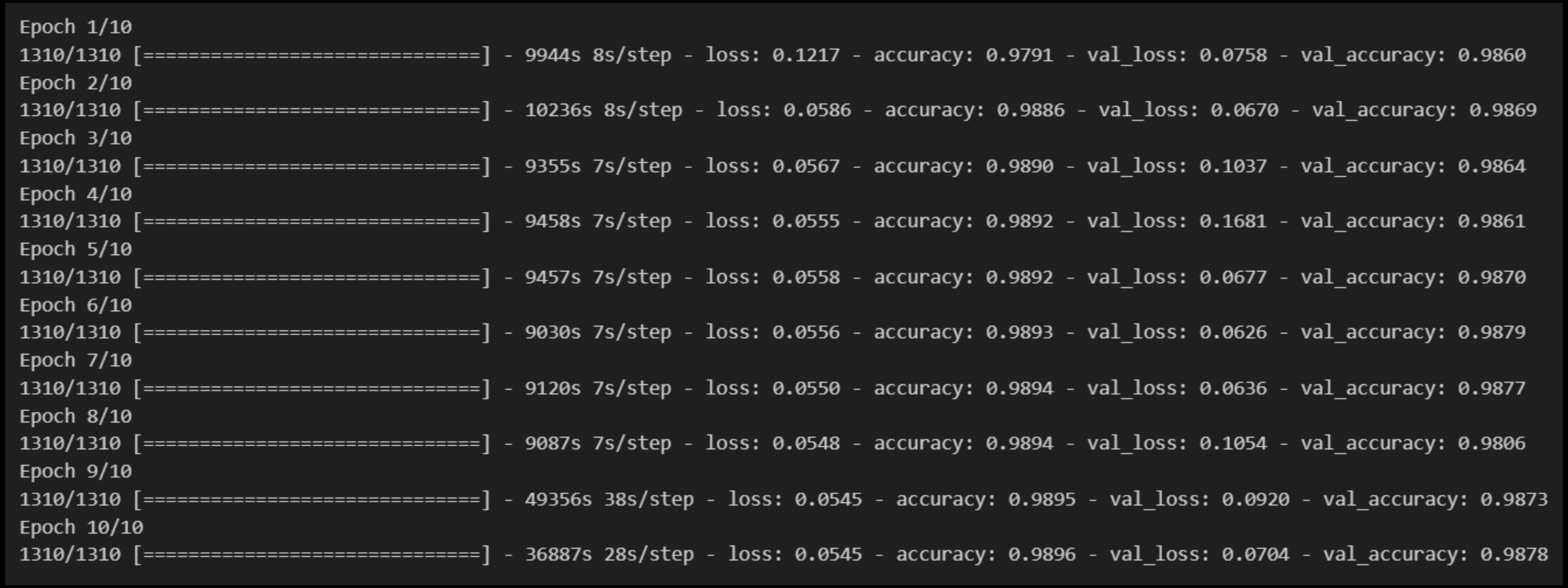
Model 1 Results

- Best score:
  - Accuracy/val accuracy: 0.9896/0.9878
  - Loss/val loss: 0.0545/0.0704
  - I did not include f1 score

Because of the lack of a GPU, my computer used the CPU to train the model, which significantly increased the training time.



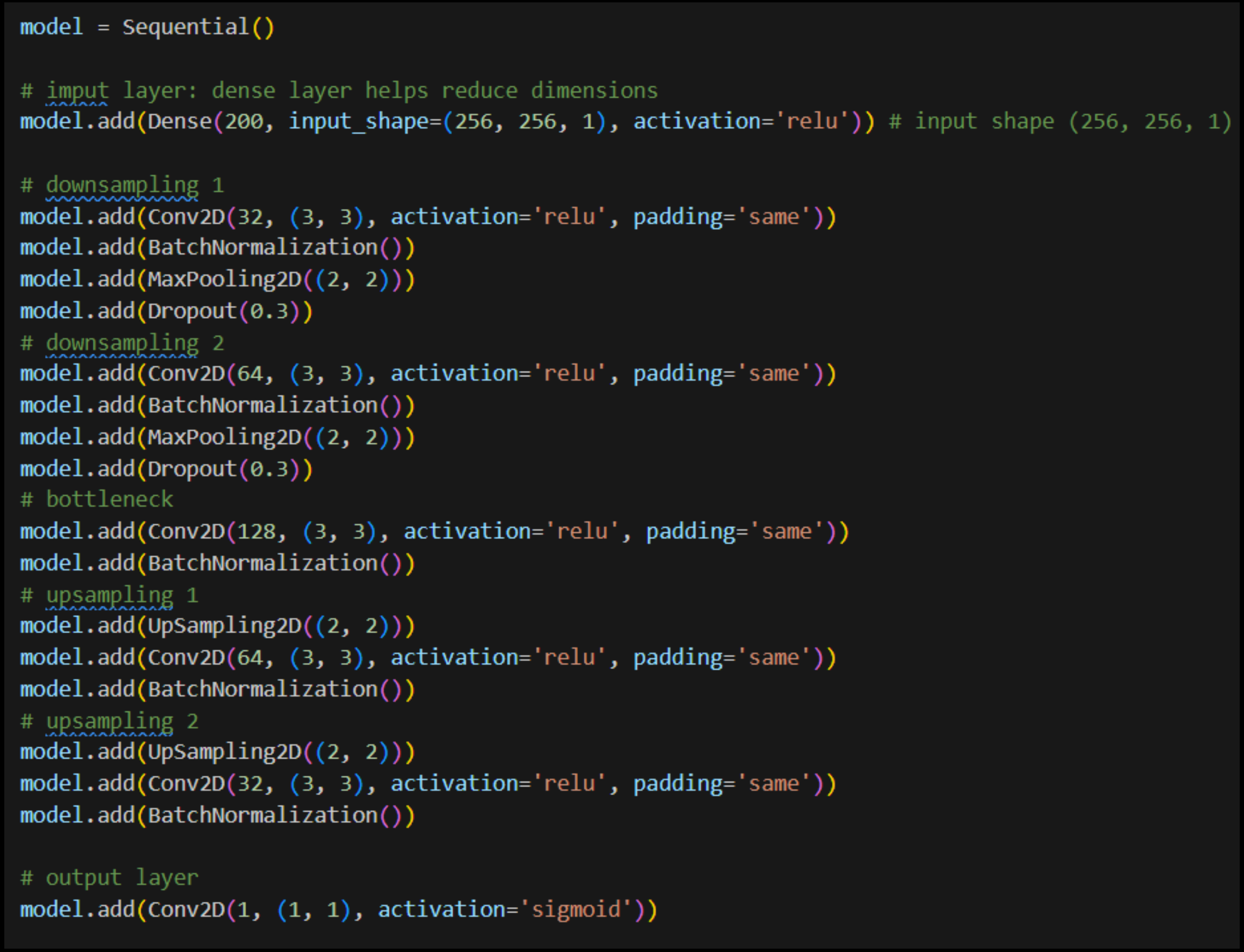
```
699m 4.4s

Epoch 1/10
1310/1310 [==============
```



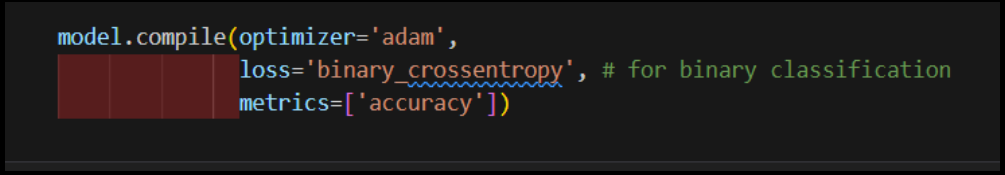
```
Epoch 1/10
1310/1310 [==============================] - 9944s 8s/step - loss: 0.1217 - accuracy: 0.9791 - val_loss: 0.0758 - val_accuracy: 0.9860
Epoch 2/10
1310/1310 [==============================] - 10236s 8s/step - loss: 0.0586 - accuracy: 0.9886 - val_loss: 0.0670 - val_accuracy: 0.9869
Epoch 3/10
1310/1310 [==============================] - 9355s 7s/step - loss: 0.0567 - accuracy: 0.9890 - val_loss: 0.1037 - val_accuracy: 0.9864
Epoch 4/10
1310/1310 [==============================] - 9458s 7s/step - loss: 0.0555 - accuracy: 0.9892 - val_loss: 0.1681 - val_accuracy: 0.9861
Epoch 5/10
1310/1310 [==============================] - 9457s 7s/step - loss: 0.0558 - accuracy: 0.9892 - val_loss: 0.0677 - val_accuracy: 0.9870
Epoch 6/10
1310/1310 [==============================] - 9030s 7s/step - loss: 0.0556 - accuracy: 0.9893 - val_loss: 0.0626 - val_accuracy: 0.9879
Epoch 7/10
1310/1310 [==============================] - 9120s 7s/step - loss: 0.0550 - accuracy: 0.9894 - val_loss: 0.0636 - val_accuracy: 0.9877
Epoch 8/10
1310/1310 [==============================] - 9087s 7s/step - loss: 0.0548 - accuracy: 0.9894 - val_loss: 0.1054 - val_accuracy: 0.9806
Epoch 9/10
1310/1310 [==============================] - 49356s 38s/step - loss: 0.0545 - accuracy: 0.9895 - val_loss: 0.0920 - val_accuracy: 0.9873
Epoch 10/10
1310/1310 [==============================] - 36887s 28s/step - loss: 0.0545 - accuracy: 0.9896 - val_loss: 0.0704 - val_accuracy: 0.9878
```

Model 1 Architecture

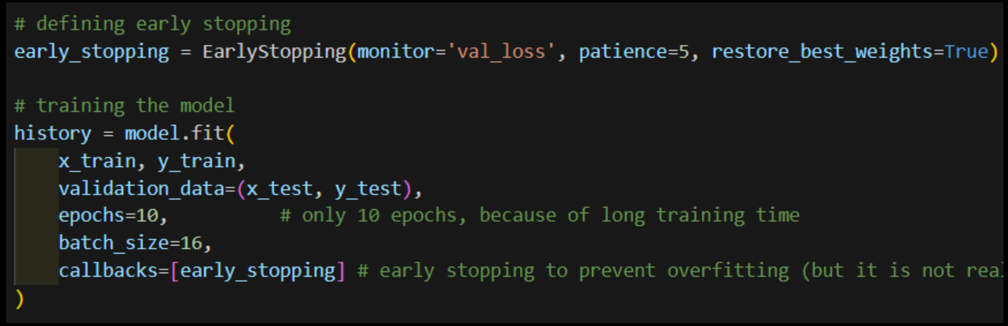
To meet client requirements, I used "val_..." images for validation and Y2B_24 data for training.



```
model = Sequential()

# imput layer: dense layer helps reduce dimensions
model.add(Dense(200, input_shape=(256, 256, 1), activation='relu')) # input shape (256, 256, 1)

# downsampling 1
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# downsampling 2
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# bottleneck
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# upsampling 1
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# upsampling 2
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())

# output layer
model.add(Conv2D(1, (1, 1), activation='sigmoid'))
```



```
model.compile(optimizer='adam',
              loss='binary_crossentropy', # for binary classification
              metrics=['accuracy'])
```



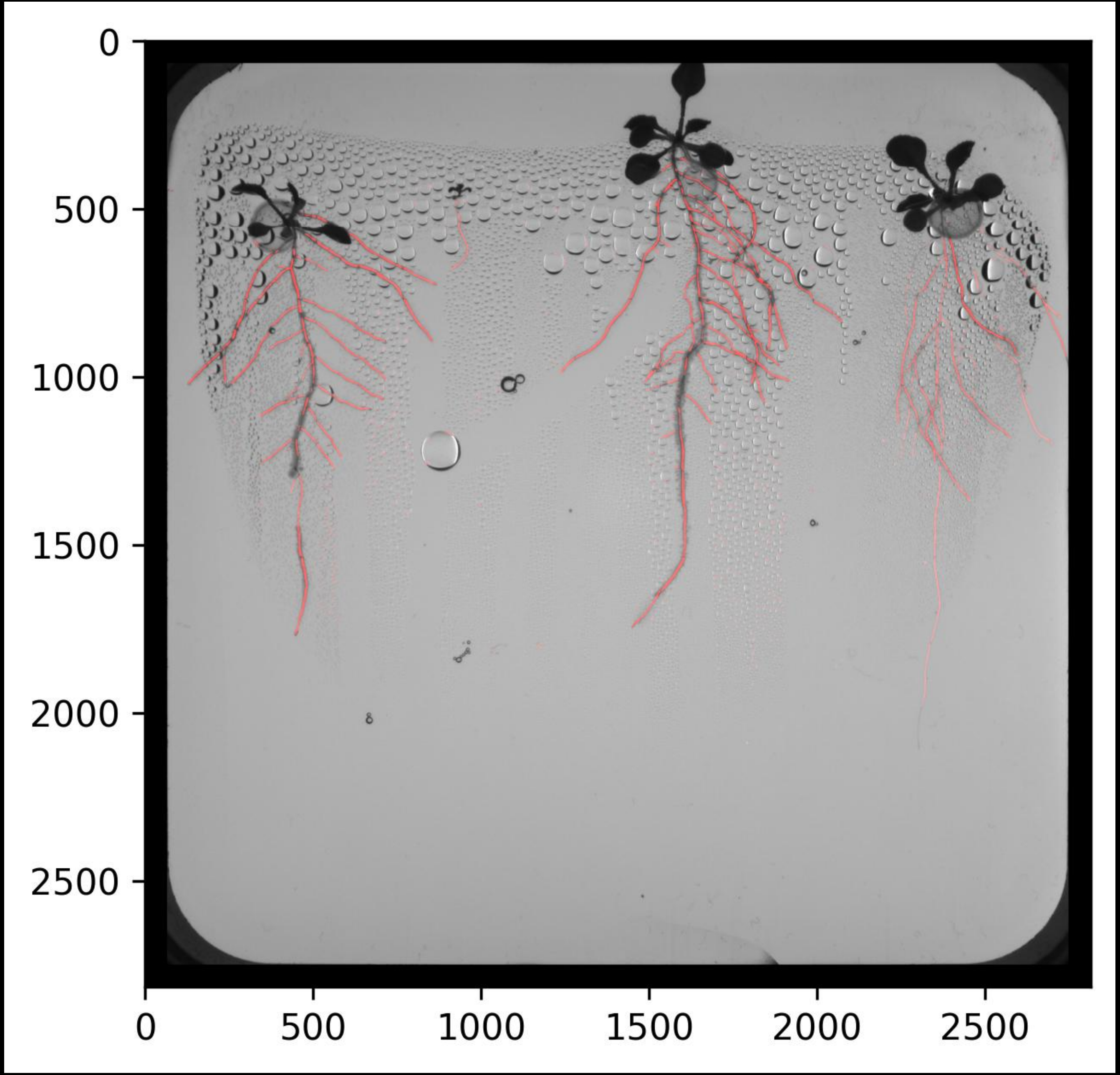
```
# defining early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# training the model
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    epochs=10,          # only 10 epochs, because of long training time
    batch_size=16,
    callbacks=[early_stopping] # early stopping to prevent overfitting (but it is not rea
)
```

- Early stopping is not necessary, since I am training only with 10 epochs
- I am using a convolutional autoencoder

Model 1 Template Image Results

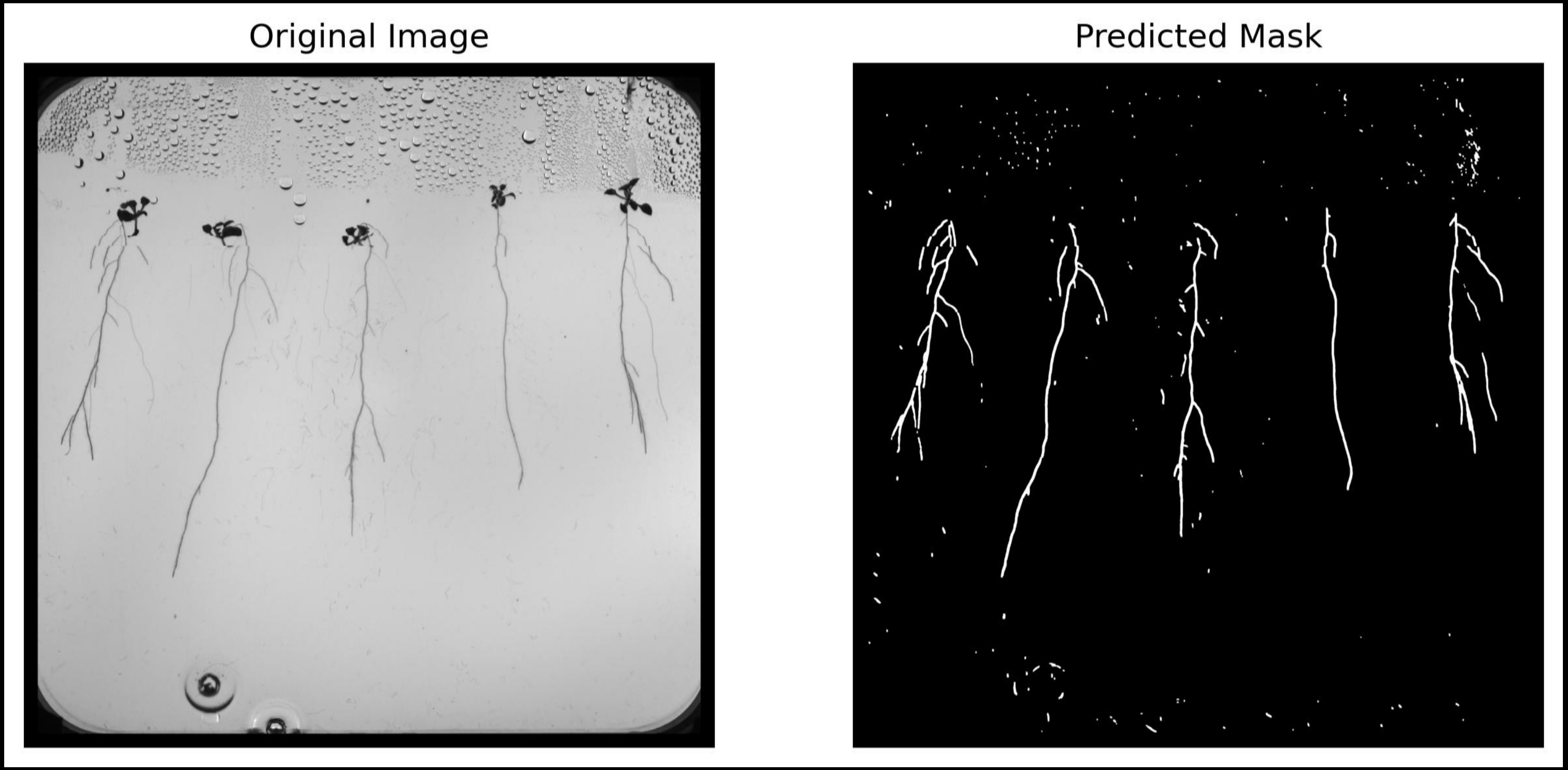
The model is detecting roots quite well. However, in some places they are not connected.
It is also detecting background as root system part.

Inference template image →



Model 1 Visual Results

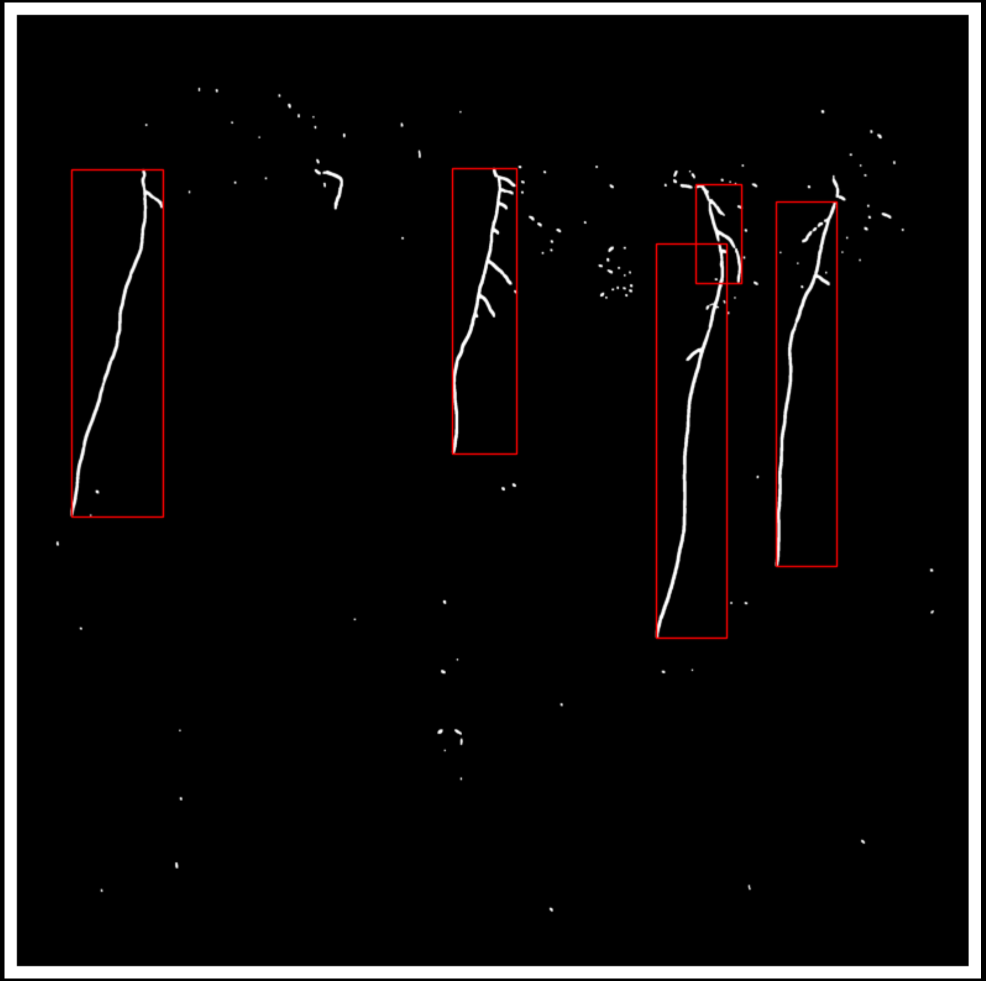
```
kernel = np.ones((5, 5), dtype="uint8")
im_rod_d1 = cv2.dilate(predicted_mask, kernel, iterations=2)
im_rod_closing = cv2.erode(im_rod_d1, kernel, iterations=1)
```

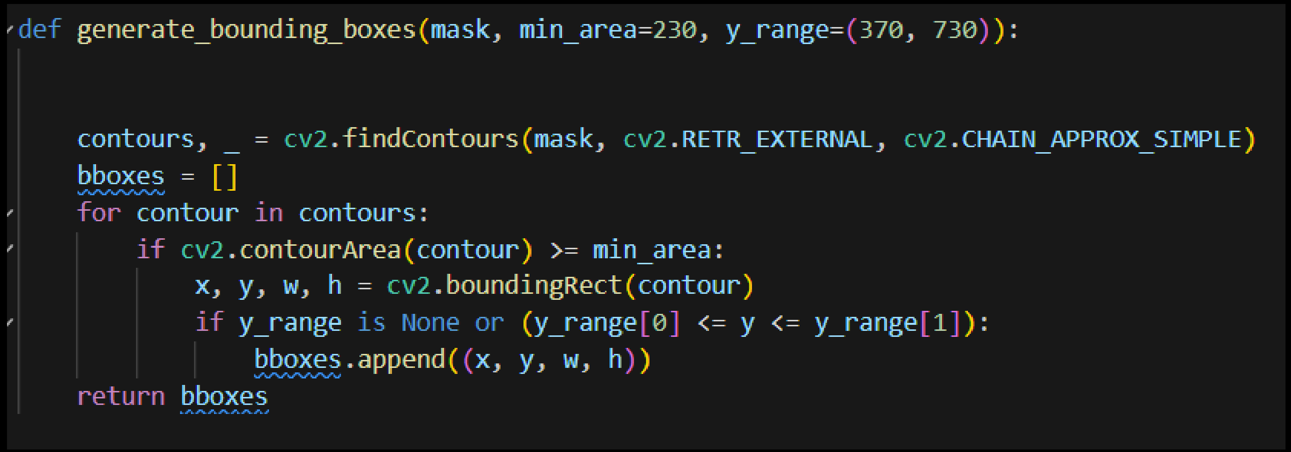


- Detecting big amount of background noise
- Parts of roots are not connected witch each other
- I am using closing to remove small gaps in masks

Mask after closing

Model 1
Visual Results





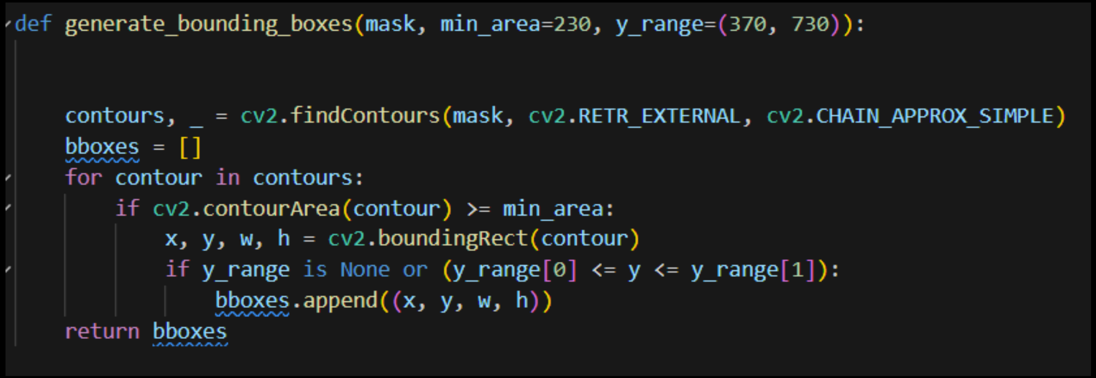
```python
def generate_bounding_boxes(mask, min_area=230, y_range=(370, 730)):


    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    bboxes = []
    for contour in contours:
        if cv2.contourArea(contour) >= min_area:
            x, y, w, h = cv2.boundingRect(contour)
            if y_range is None or (y_range[0] <= y <= y_range[1]):
                bboxes.append((x, y, w, h))
    return bboxes
```

Because of gaps in root masks, I can not detect roots correctly.

I detect objects on the masks and create bounding boxes around the 5 tallest ones.

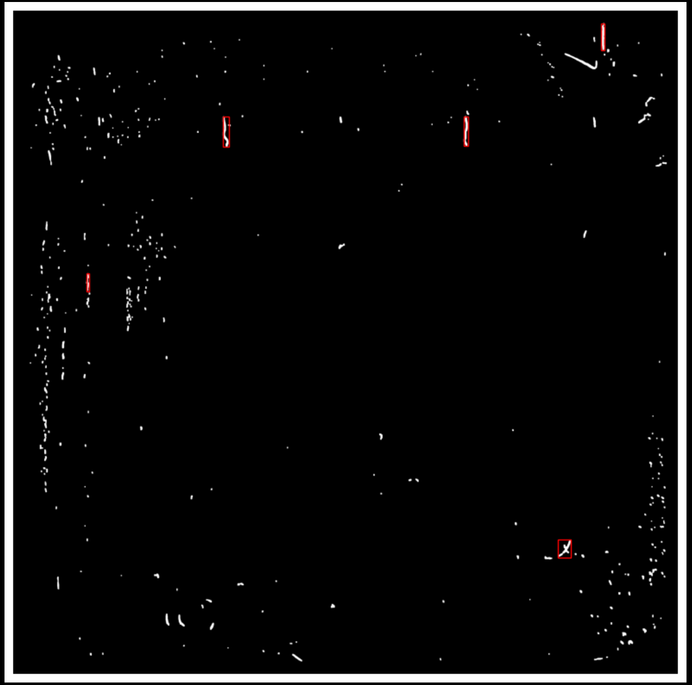
I set a minimum size to avoid detecting background noise as roots in pictures where there are fewer than 5 roots.

Model 1 Bounding Boxes



```
def generate_bounding_boxes(mask, min_area=230, y_range=(370, 730)):


    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    bboxes = []
    for contour in contours:
        if cv2.contourArea(contour) >= min_area:
            x, y, w, h = cv2.boundingRect(contour)
            if y_range is None or (y_range[0] <= y <= y_range[1]):
                bboxes.append((x, y, w, h))
    return bboxes
```

Before implementation



Original image

Bounding boxes need to start in y_range = (370,730)

I implemented this solution because I noticed that in some cases a bounding box with background noise was detected as one of the 5 largest boxes.

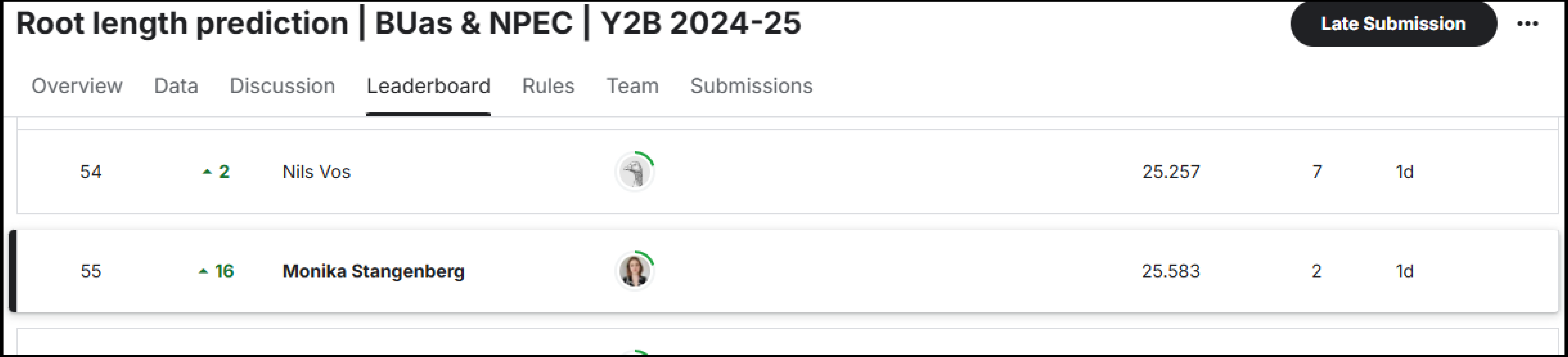
This appeared mainly in images where the roots were very small. By limiting the space where bounding boxes could start, I was able to target the roots more effectively.

Model 1
Bounding Boxes

- To process root images I skeletonize them, analyzed their structures and extracted metrics.

- I processes CSV files containing branch data extracted from test images. It calculates the maximum root length for each plant in every image from Kaggle submission photos and generates a consolidated CSV file.

- Kaggle score:



Root length prediction | BUas & NPEC | Y2B 2024-25

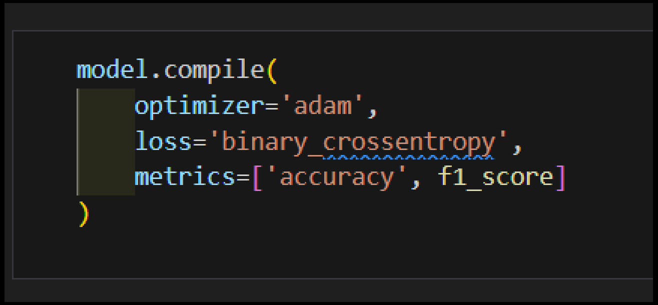
54	▲ 2	Nils Vos	25.257	7	1d
55	▲ 16	**Monika Stangenberg**	25.583	2	1d

Model 2 structure

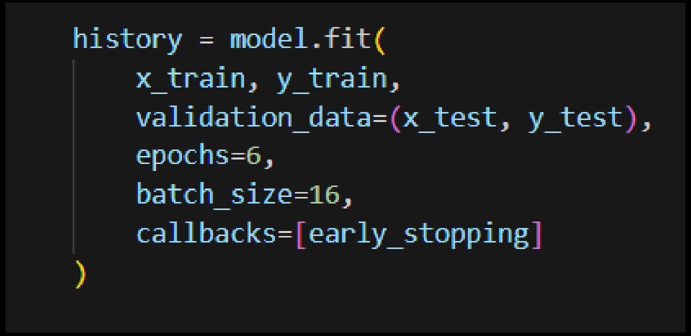
I used model provided in self-study notebook 7

I also added f1 to metrics.

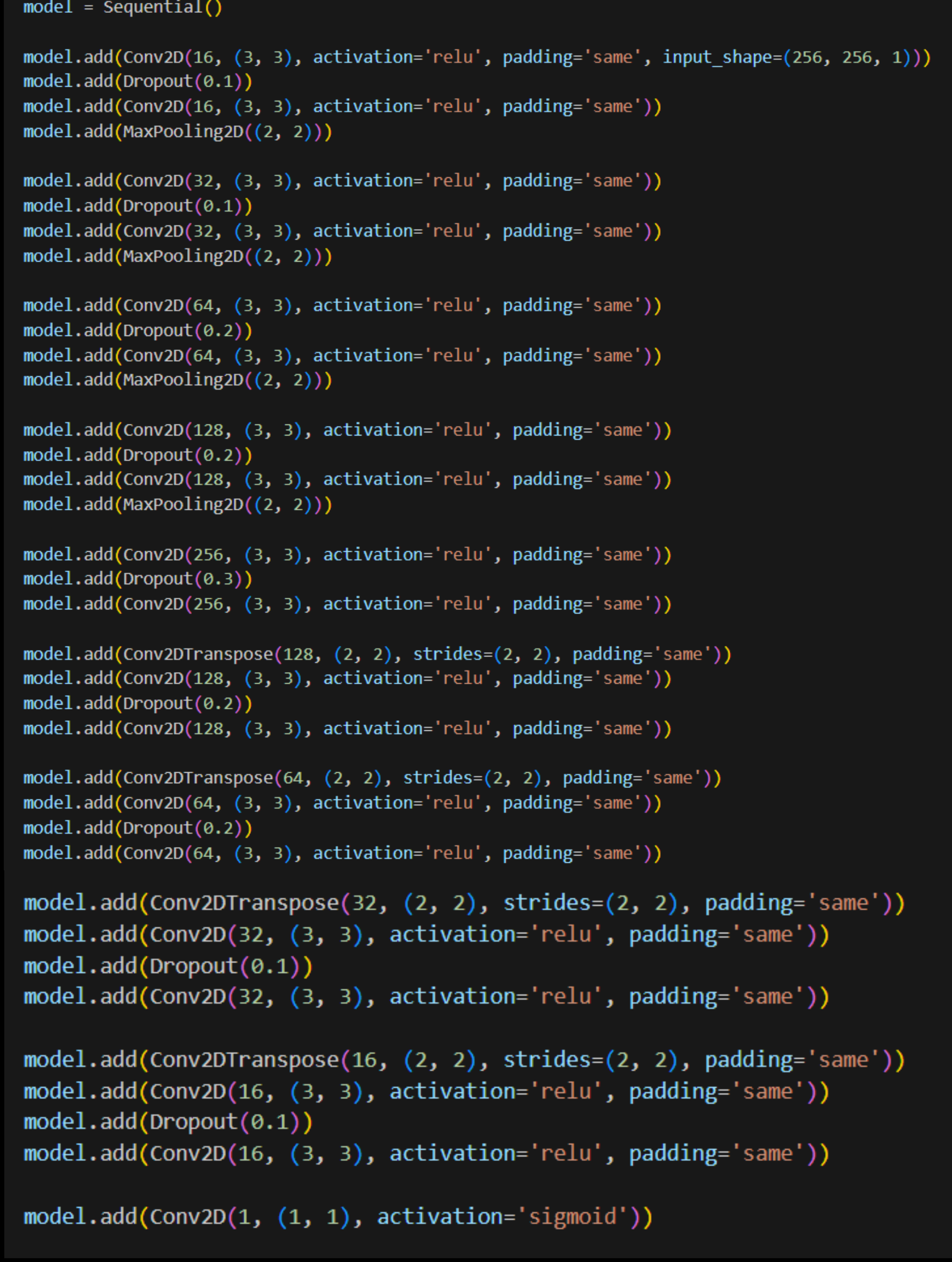
Because of long training time, epochs = 6.



```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy', f1_score]
)
```



```
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    epochs=6,
    batch_size=16,
    callbacks=[early_stopping]
)
```



```
model = Sequential()

model.add(Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=(256, 256, 1)))
model.add(Dropout(0.1))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.1))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

model.add(Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

model.add(Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.1))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))

model.add(Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same'))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.1))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(1, (1, 1), activation='sigmoid'))
```

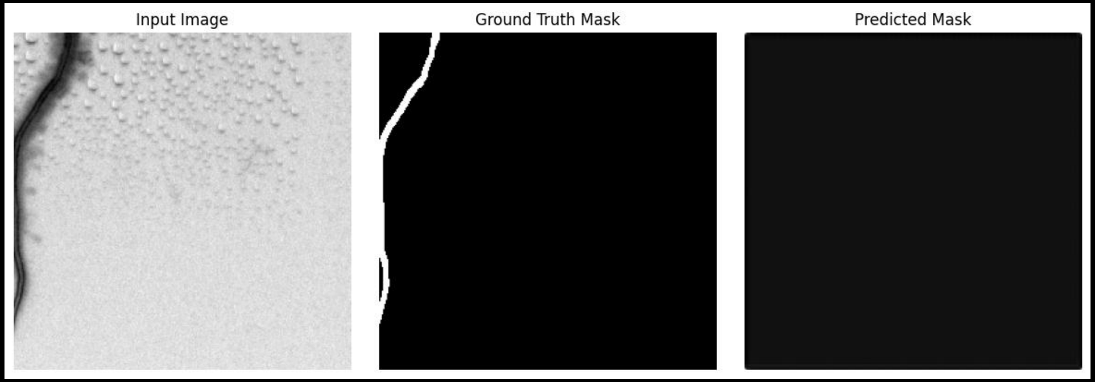
Model 2
Visual Results

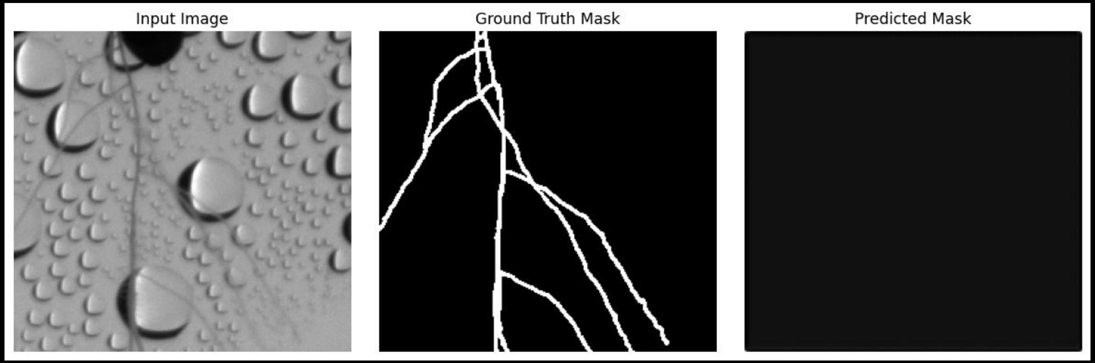
This model didn't work well.
Output is just plain black screen.

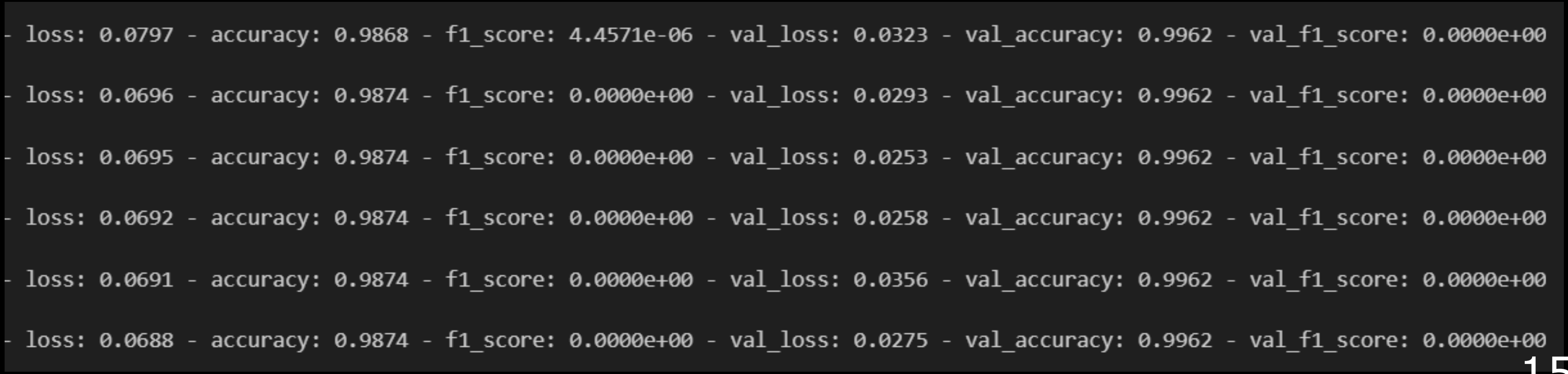
The model structure is correct.

Loss function should be changed.

Because of this results, I didn't continue working on this model.



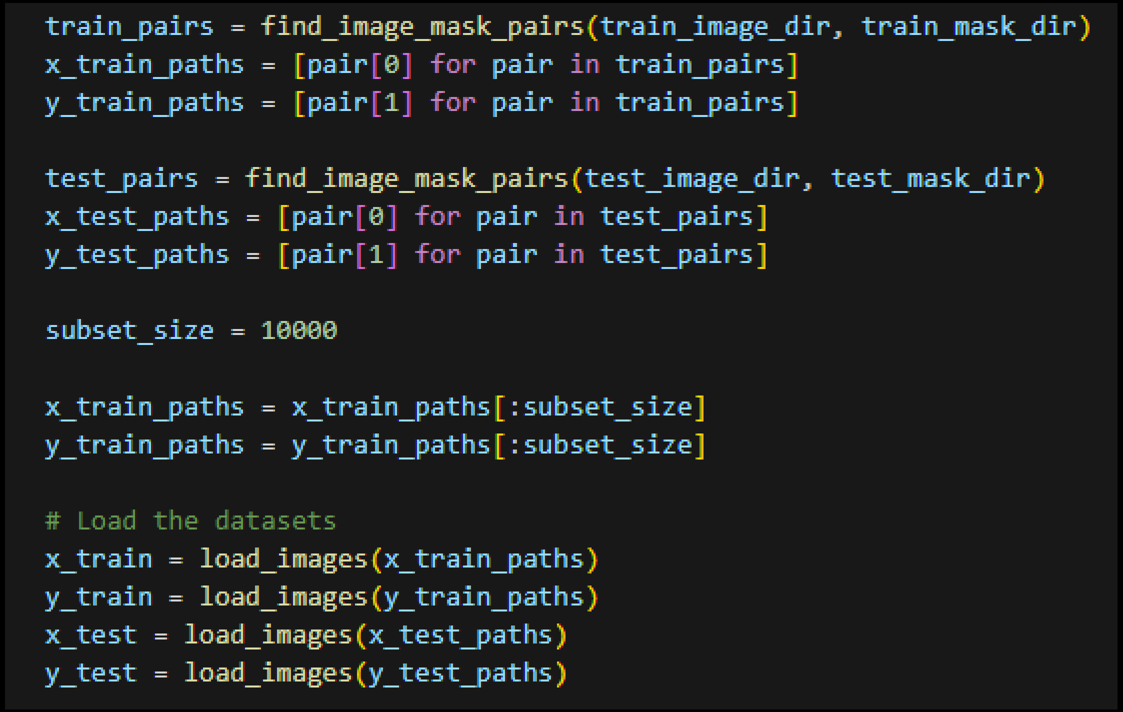




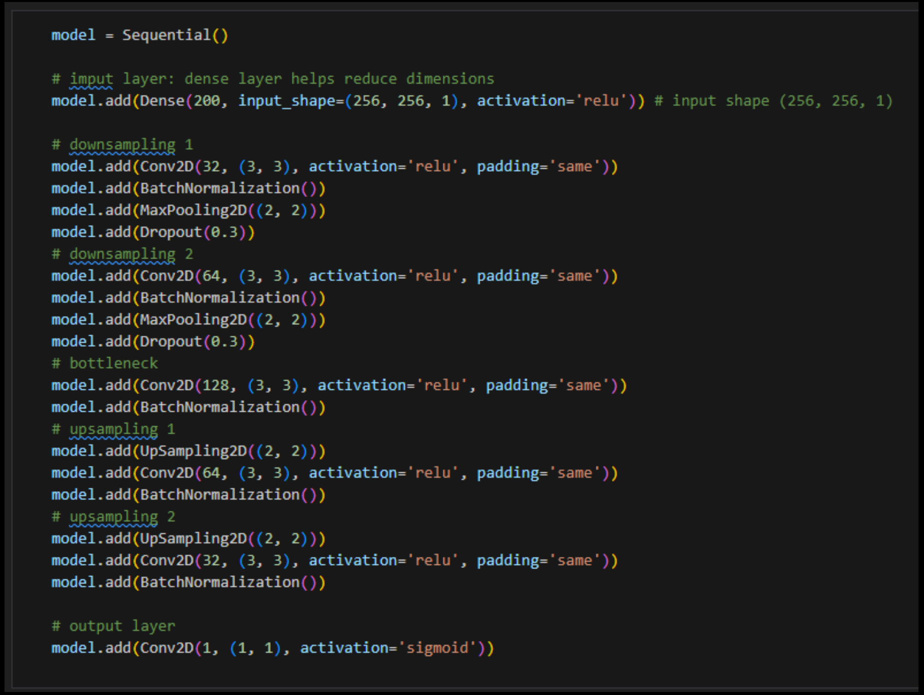
```
- loss: 0.0797 - accuracy: 0.9868 - f1_score: 4.4571e-06 - val_loss: 0.0323 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00

- loss: 0.0696 - accuracy: 0.9874 - f1_score: 0.0000e+00 - val_loss: 0.0293 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00

- loss: 0.0695 - accuracy: 0.9874 - f1_score: 0.0000e+00 - val_loss: 0.0253 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00

- loss: 0.0692 - accuracy: 0.9874 - f1_score: 0.0000e+00 - val_loss: 0.0258 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00

- loss: 0.0691 - accuracy: 0.9874 - f1_score: 0.0000e+00 - val_loss: 0.0356 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00

- loss: 0.0688 - accuracy: 0.9874 - f1_score: 0.0000e+00 - val_loss: 0.0275 - val_accuracy: 0.9962 - val_f1_score: 0.0000e+00
```

Model 3

I didn’t include f1 score in my previous working model.
Since my model training time is very long, I decided to make my dataset smaller to decrease training time.



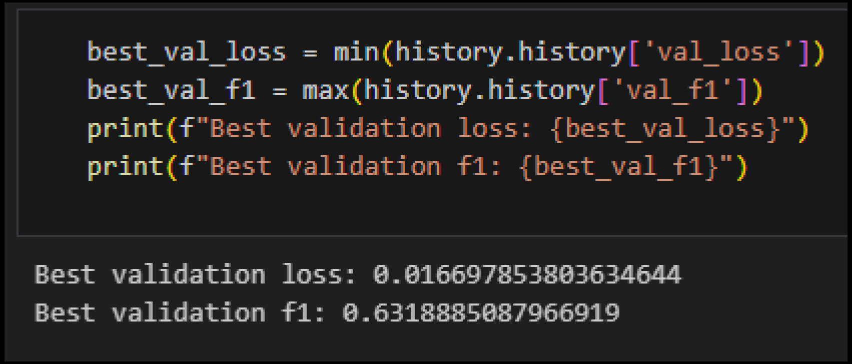
```
train_pairs = find_image_mask_pairs(train_image_dir, train_mask_dir)
x_train_paths = [pair[0] for pair in train_pairs]
y_train_paths = [pair[1] for pair in train_pairs]

test_pairs = find_image_mask_pairs(test_image_dir, test_mask_dir)
x_test_paths = [pair[0] for pair in test_pairs]
y_test_paths = [pair[1] for pair in test_pairs]

subset_size = 10000

x_train_paths = x_train_paths[:subset_size]
y_train_paths = y_train_paths[:subset_size]

# Load the datasets
x_train = load_images(x_train_paths)
y_train = load_images(y_train_paths)
x_test = load_images(x_test_paths)
y_test = load_images(y_test_paths)
```



```
model = Sequential()

# imput layer: dense layer helps reduce dimensions
model.add(Dense(200, input_shape=(256, 256, 1), activation='relu')) # input shape (256, 256, 1)

# downsampling 1
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# downsampling 2
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# bottleneck
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# upsampling 1
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# upsampling 2
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())

# output layer
model.add(Conv2D(1, (1, 1), activation='sigmoid'))
```

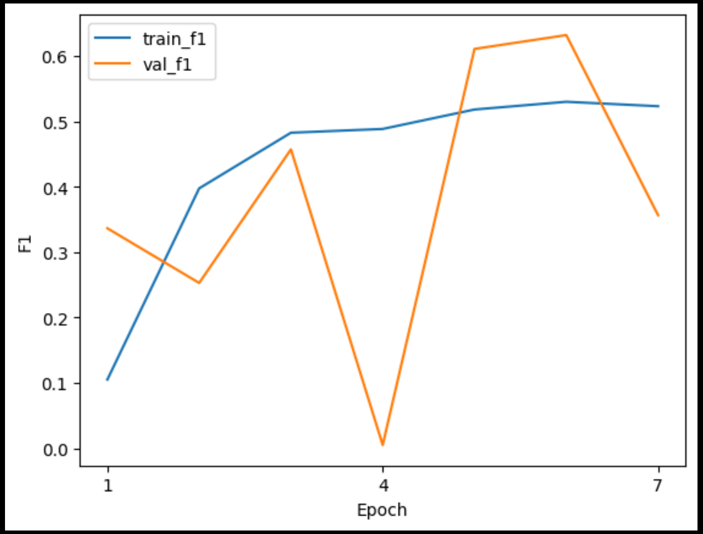
Model architecture is the same as in model 1

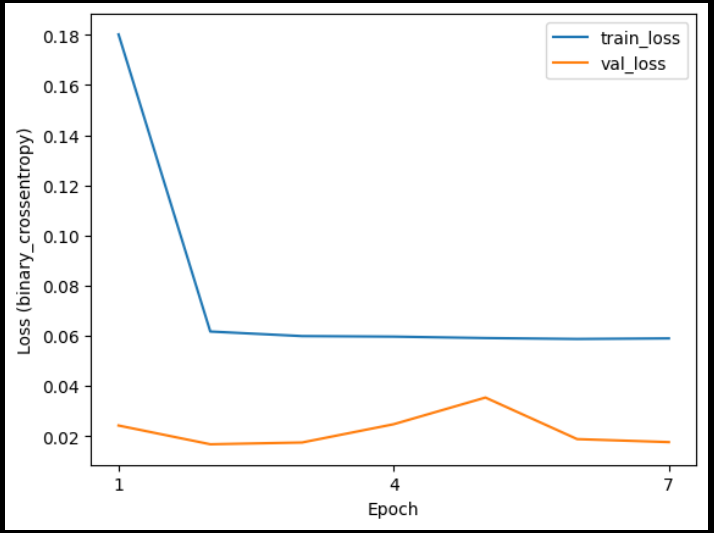
Model 3 Results

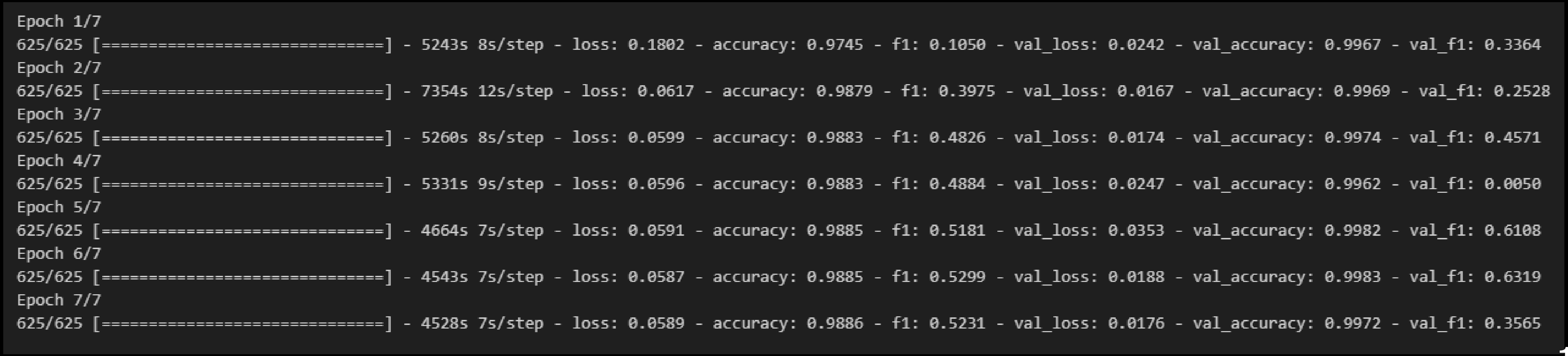


```
best_val_loss = min(history.history['val_loss'])
best_val_f1 = max(history.history['val_f1'])
print(f"Best validation loss: {best_val_loss}")
print(f"Best validation f1: {best_val_f1}")
```

```
Best validation loss: 0.016697853803634644
Best validation f1: 0.6318885087966919
```





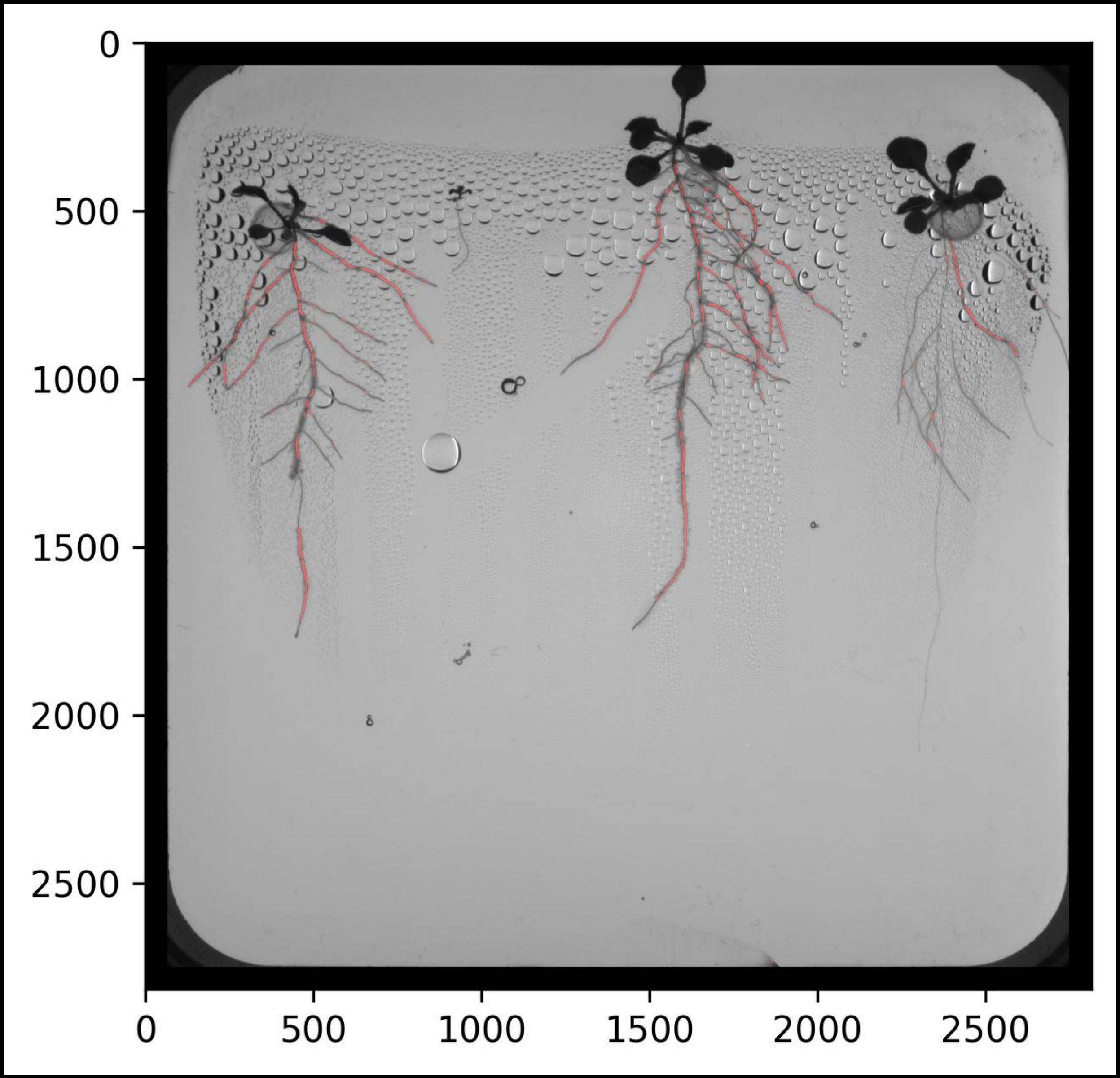


```
Epoch 1/7
625/625 [==============================] - 5243s 8s/step - loss: 0.1802 - accuracy: 0.9745 - f1: 0.1050 - val_loss: 0.0242 - val_accuracy: 0.9967 - val_f1: 0.3364
Epoch 2/7
625/625 [==============================] - 7354s 12s/step - loss: 0.0617 - accuracy: 0.9879 - f1: 0.3975 - val_loss: 0.0167 - val_accuracy: 0.9969 - val_f1: 0.2528
Epoch 3/7
625/625 [==============================] - 5260s 8s/step - loss: 0.0599 - accuracy: 0.9883 - f1: 0.4826 - val_loss: 0.0174 - val_accuracy: 0.9974 - val_f1: 0.4571
Epoch 4/7
625/625 [==============================] - 5331s 9s/step - loss: 0.0596 - accuracy: 0.9883 - f1: 0.4884 - val_loss: 0.0247 - val_accuracy: 0.9962 - val_f1: 0.0050
Epoch 5/7
625/625 [==============================] - 4664s 7s/step - loss: 0.0591 - accuracy: 0.9885 - f1: 0.5181 - val_loss: 0.0353 - val_accuracy: 0.9982 - val_f1: 0.6108
Epoch 6/7
625/625 [==============================] - 4543s 7s/step - loss: 0.0587 - accuracy: 0.9885 - f1: 0.5299 - val_loss: 0.0188 - val_accuracy: 0.9983 - val_f1: 0.6319
Epoch 7/7
625/625 [==============================] - 4528s 7s/step - loss: 0.0589 - accuracy: 0.9886 - f1: 0.5231 - val_loss: 0.0176 - val_accuracy: 0.9972 - val_f1: 0.3565
```

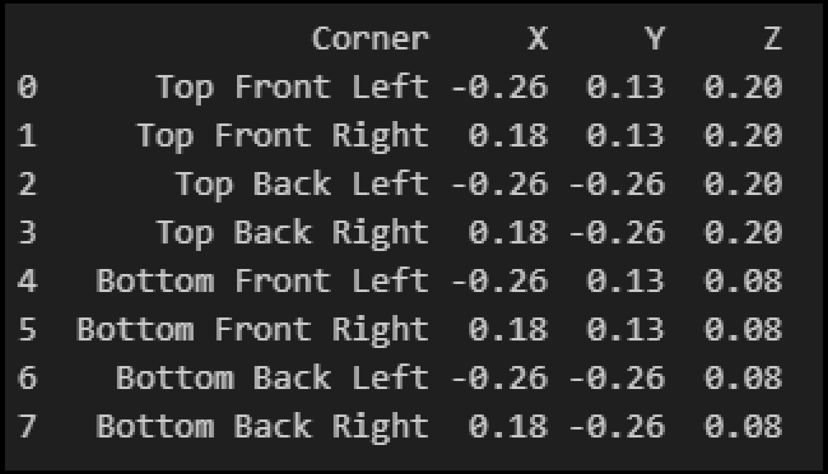
Model 3 Visual Results

This model was not trained enough, which explains this results. Model is detecting roots correctly, but it is also not detecting them on lighter sport (plant 2/5).

Best f1 = 0.63



Robotics



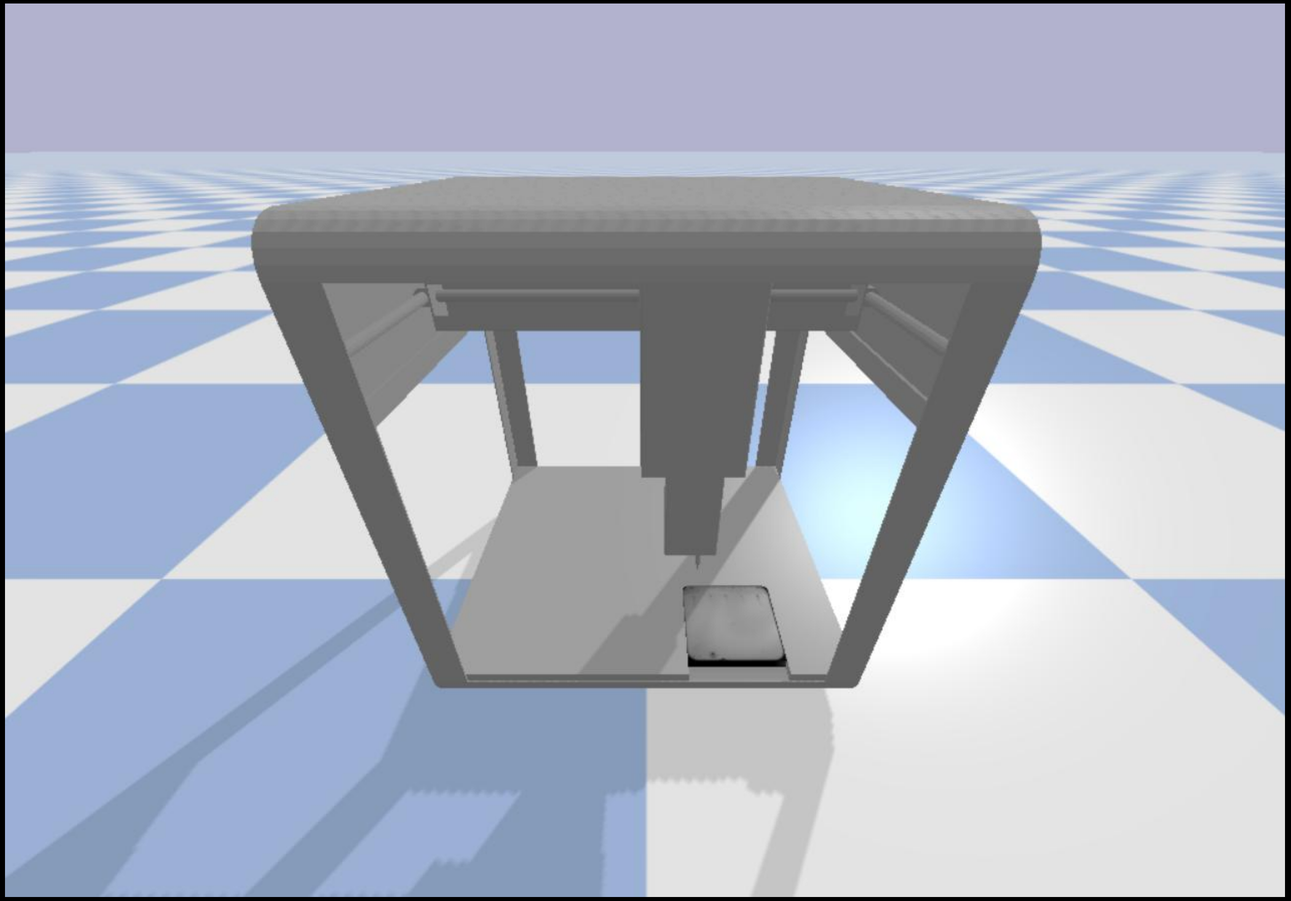
```
              Corner     X     Y     Z
0     Top Front Left -0.26  0.13  0.20
1    Top Front Right  0.18  0.13  0.20
2      Top Back Left -0.26 -0.26  0.20
3     Top Back Right  0.18 -0.26  0.20
4  Bottom Front Left -0.26  0.13  0.08
5 Bottom Front Right  0.18  0.13  0.08
6   Bottom Back Left -0.26 -0.26  0.08
7  Bottom Back Right  0.18 -0.26  0.08
```

Simulation Environment

- I identified the maximum and minimum X, Y, Z coordinates
- Sent movement commands in all directions to determine robot's range. During robot’s movement I recorded robot’s positions.
- Displayed maximum/minimum coordinated in a table

Creating a Gym Environment

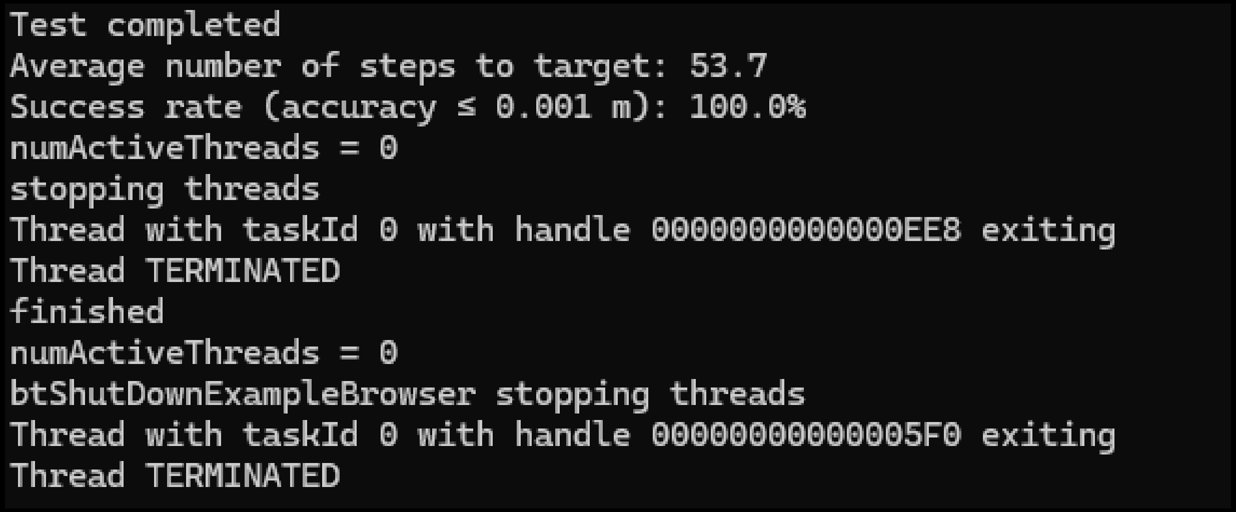
- Developed a custom environment to integrate the robot's simulation with RL
- Defined the range of movements
- Defined observation space



Reinforcement Learning

The goal is to train an RL agent to move the pipette to any target position.

Test results:



```
Test completed
Average number of steps to target: 53.7
Success rate (accuracy ≤ 0.001 m): 100.0%
numActiveThreads = 0
stopping threads
Thread with taskId 0 with handle 0000000000000EE8 exiting
Thread TERMINATED
finished
numActiveThreads = 0
btShutDownExampleBrowser stopping threads
Thread with taskId 0 with handle 00000000000005F0 exiting
Thread TERMINATED
```

Hyperparameters:
Learning Rate: 0.0003
Batch Size: 64
n_steps: 1000
Gamma: 0.99

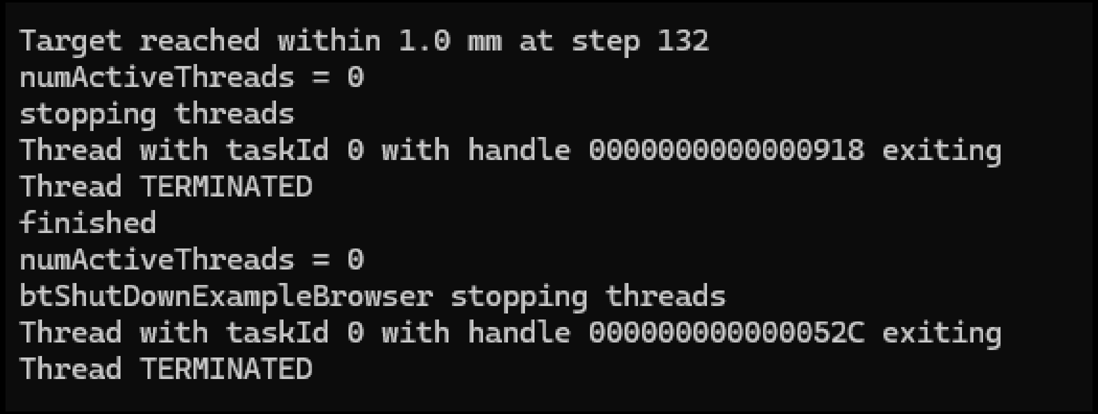
Best Result:
Reached pipette goal with accuracy < 1mm with (average 53.7 steps)

Creating a Controller

Move the pipette tip accurately within the working envelope using PID control.

Performing Gains:
Kp = 0.8
Ki = 0.2
Kd = 0.4

PID controller test results:



```
Target reached within 1.0 mm at step 132
numActiveThreads = 0
stopping threads
Thread with taskId 0 with handle 0000000000000918 exiting
Thread TERMINATED
finished
numActiveThreads = 0
btShutDownExampleBrowser stopping threads
Thread with taskId 0 with handle 000000000000052C exiting
Thread TERMINATED
```

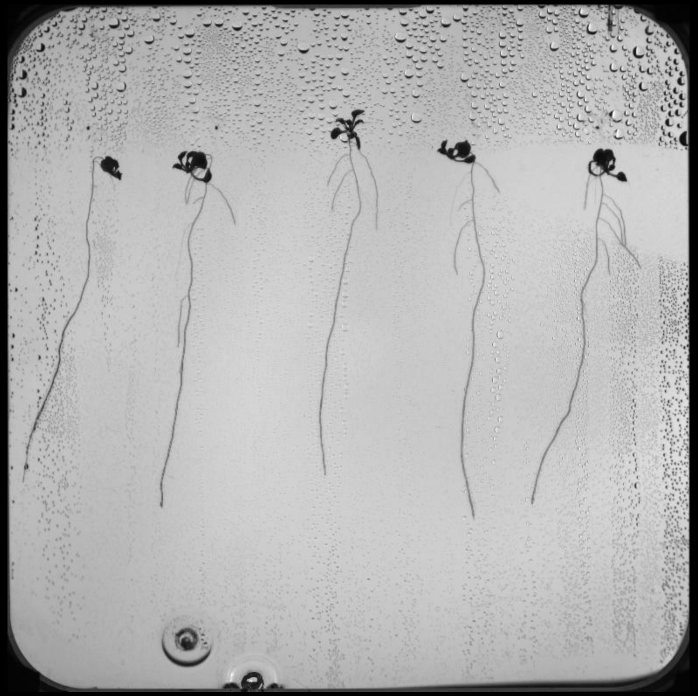
Achieved accuracy < 0.001 m

Assumptions

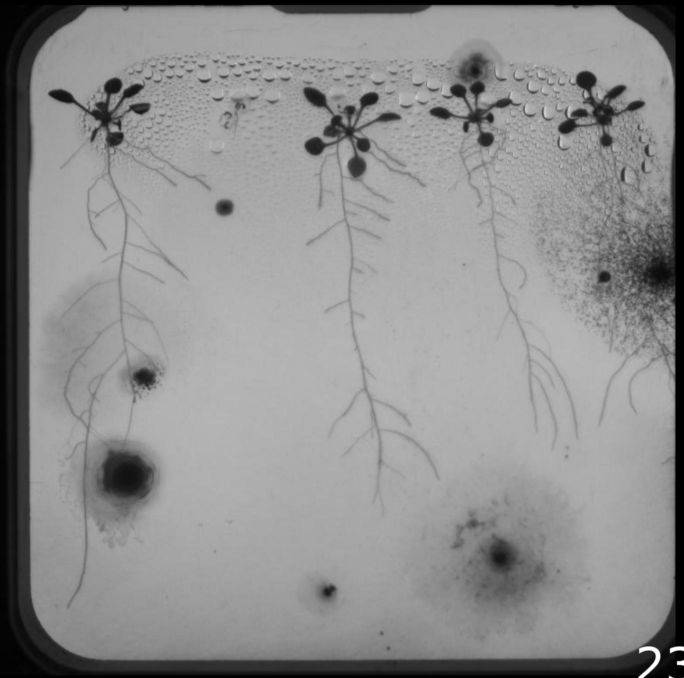
Photo with only plants

- I designed the pipeline with the assumption that the train/validation/test images provided by NPEC would contain only plants in Petri dish area.

Kaggle image →



Train image →

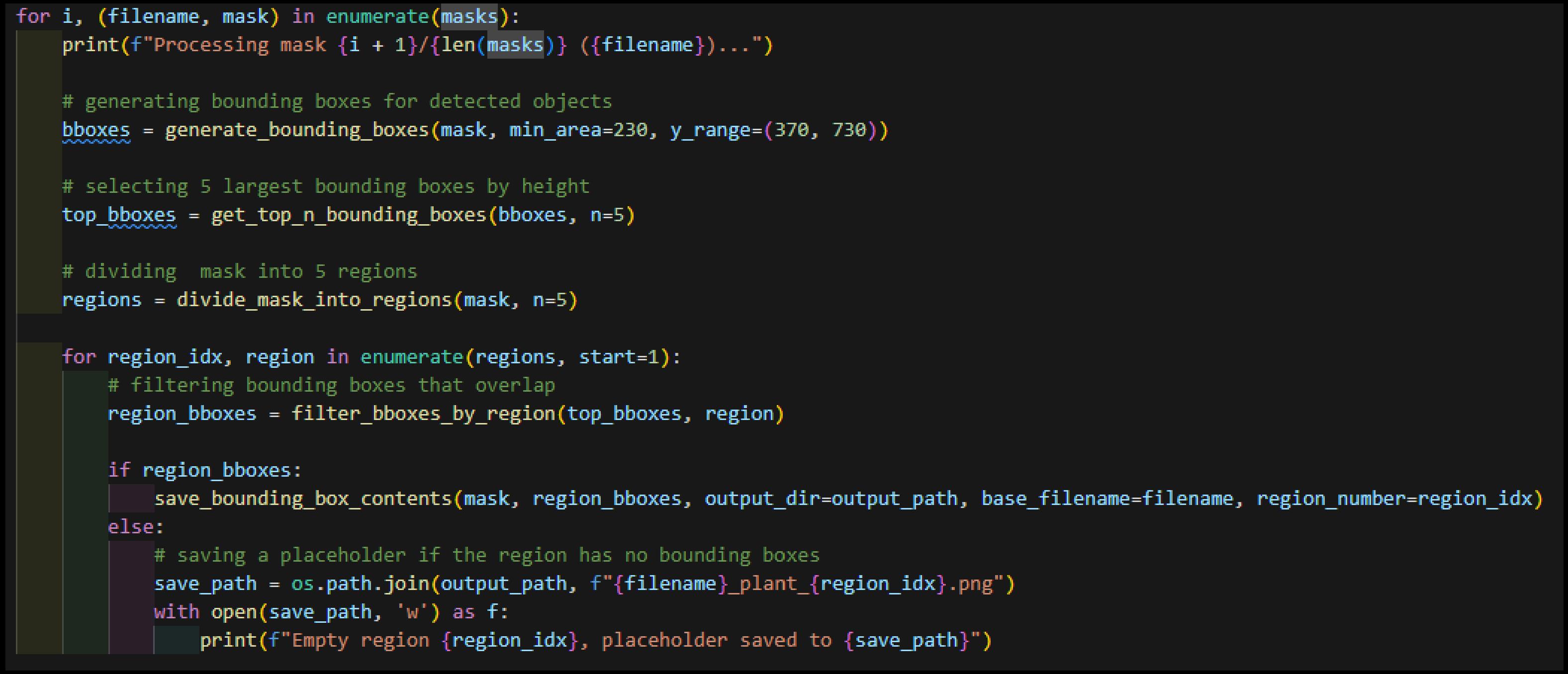


5 plants on every image

My code was designed with assumption that they are going to be 5 plants in every image. However, in some photos this is not true.

If there are fewer than 5 bounding boxes, their images are replaced with placeholders.

I divide the predicted mask into 5 sections to receive consistent output.



```python
for i, (filename, mask) in enumerate(masks):
    print(f"Processing mask {i + 1}/{len(masks)} ({filename})...")

    # generating bounding boxes for detected objects
    bboxes = generate_bounding_boxes(mask, min_area=230, y_range=(370, 730))

    # selecting 5 largest bounding boxes by height
    top_bboxes = get_top_n_bounding_boxes(bboxes, n=5)

    # dividing  mask into 5 regions
    regions = divide_mask_into_regions(mask, n=5)

    for region_idx, region in enumerate(regions, start=1):
        # filtering bounding boxes that overlap
        region_bboxes = filter_bboxes_by_region(top_bboxes, region)

        if region_bboxes:
            save_bounding_box_contents(mask, region_bboxes, output_dir=output_path, base_filename=filename, region_number=region_idx)
        else:
            # saving a placeholder if the region has no bounding boxes
            save_path = os.path.join(output_path, f"{filename}_plant_{region_idx}.png")
            with open(save_path, 'w') as f:
                print(f"Empty region {region_idx}, placeholder saved to {save_path}")
```

Limitations

- I don't have GPU on my computer -> Long training times locally. I wasn't able to test multiple models for long enough.
- Big/heavy dataset -> While trying to train my model on Paperspace, I faced issues with uploading data, difficulty transferring data.
- Not trained enough model -> Gaps between roots. Because of it I wasn't able to detect all plants correctly on some images.

Next Steps

- Val_f1 score >= 10
- Correctly predicted roots
- Training new model on a server
- Using the entire dataset
- Potentially using the model from notebook 7 and addressing the problem

Developing better model

Creating a PID Controller

- Working on the server provided by BUas
- Designed a PID controller to move pipette tip with precision <=1mm
- Implementing three PID controllers responsible for three dimensions (X, Y, Z)

Integrating the Computer Vision Pipeline

- Connecting CV pipeline to PID and RL controllers for precise control
- Converting root tip pixel coordinates into robot-compatible coordinates.
- Testing the integration

Performance Benchmarking

- Quantifying and comparing performance of PID and RL controllers based on accuracy and speed
- Identifying metrics and methods to evaluate both controllers

Summary

Thank you for your attention