

# Fatigue Fracture Surfaces

Anthony Lino

2024-10-30

## Abstract

## Introduction

### Fatigue

Fatigue is a common failure mechanism for commercial equipment. Fatigue is a phenomenon where cracks will grow as they solid experiences cycles of high and low stress, gradually weakening the solid and eventually causing failure well below the expected strength of the material. This is shown in figure 1, where the stress experienced near the crack tip is much greater than the stress experienced throughout the rest of the material, and the ratio between these two values is called the stress concentration factor (SCF).

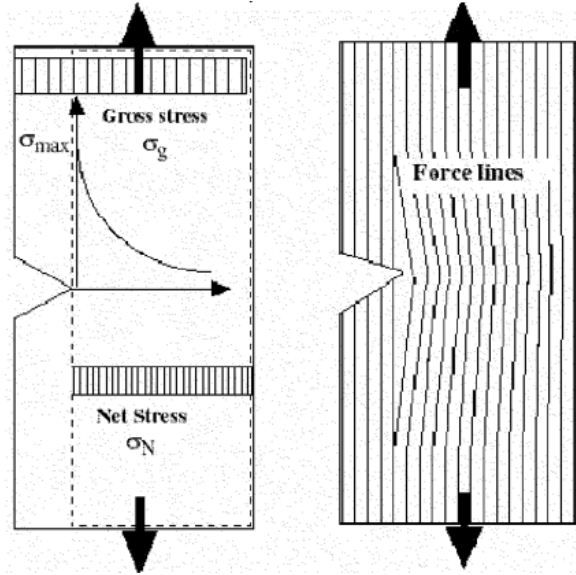


Figure 1: In the left figure the horizontal axis is the distance from the crack in the plane of the crack tip and the vertical axis is the stress as that position. We can see that as we get closer to the crack, the stress experienced at that point will go up to a maximum stress. On the right figure, this is visualized with force lines, which are meant to represent the force passing in a straight line from the bottom to the top of the material. Near the crack tip, several of the force lines are cut off, and are forced to pool near the crack tip, which causes the higher experienced stress.

As the material is stressed, the crack will grow, but eventually reach an equilibrium when the material stops stretching. This growth is also not linear, since the larger the crack the greater the concentration and the faster the growth. This feedback loop causes exponential crack growth, which shows up as linear on a log-log plot, such as figure 2. While this is easiest to explain with cracks, any irregularity in a solid, including sharp edges, surface roughness and internal defects can concentrate stress, which can cause a crack to form, and the crack can grow from there. As a result of this exponential relationship, the majority of the cycles are spent forming an initial crack from a defect, called crack nucleation, and when the crack is small. As a result, the shape, which determines the stress concentration, and the defect size, which determines the starting size of the crack, are key determiners of fatigue performance.

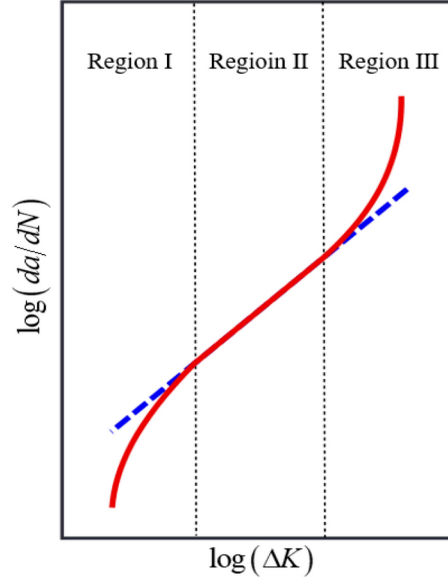


Figure 2:  $a$  is the crack length,  $N$  is the cycle and  $K$  is the stress intensity factor, which estimates the stress experienced near the tip of the crack based on its shape length and the applied stress. This means that  $d(a)/dN$  is the growth in crack length per cycle and  $\Delta K$  is the change in stress near the crack tip at the high and low point of the stress. This linear relationship near the center is called the Paris Regime and is a property of the material.

## Laser Powder Bed Fusion

Laser Powder Bed Fusion is (LPBF) an emerging manufacturing method which allows for straightforward small batch manufacturing of complex designs. The ease of manufacturing small batches and the high design freedom allows for high performance designs. This makes it particularly well suited for adoption in the Aerospace industry. However, fatigue performance is a key metric in the Aerospace industry since cyclic loading is ubiquitous in Aerospace applications. This is problematic because LPBF inherently causes a rough surface and defects, which cause poor fatigue properties. A rough surface can be polished, but defects can be embedded below the surface, making them significantly harder to fix. The number of defects can be reduced by optimizing the settings used for printing, but that can be a very time consuming process. As a result, most studies focusing on process parameter optimization rely on these easier tests, leaving a gap in the literature for studies on fatigue optimization. A study in Professor Lewandowski's lab addressed this by tested the impact of different processing parameters on the fatigue properties. After fracture, the fractured surfaces were imaged under a microscope, revealing the impact of different defects.

## Quantitative Fractography

Fractography is the study of fracture surfaces. This is a largely qualitative field, but there is an emerging field of quantitative fractography which relies on segmenting these images, and then extracting features from these masks. The Lewandowski lab explored the use of quantitative fractography for these images by performing 4 tasks:

1. Segmenting every defect from the fracture surface
2. Segmenting only the initiating defect from the fracture surface
3. Segmenting the region of fatigue crack growth
4. Segmenting the region of overload

While significant efforts were made, this is a time consuming process, it can take upwards of 4 hours for a single sample, making use of the entire dataset unfeasible. The existing manually annotated data can be used to train machine learning models to perform this segmentation task on the remainder of the data, allowing conclusions to be drawn on the remainder of the data. This is the primary task done in the remainder of this data.

## Data Science Methods

Machine learning is used to segment defects and interesting characteristics from the images. Hypothesis testing is used to show the correlation between the extracted defects and the resulting mechanical properties.

## Python Packages

- os - used for reading paths
- sys - used for adding folders to path
- cv2 - the most powerful library for image processing currently available in terms of both performance and capability
- numpy - allows images to be efficiently worked with as arrays. The foundation for cv2
- pandas - the most popular python library for working with dataframes
- math - used for mathematical functions
- random - used to create random variables
- matplotlib.pyplot - used for plotting
- re - python implementation of regex, a syntax for string processing
- datetime - used as a stopwatch to track performance
- ast - used to read in lists
- scipy - used for statistical tests
- seaborn - used to visualize standard statistical tests
- math - provides logarithmic and trigonometric functions

- joblib - package for parallelization of python scripts

organize\_data is a script used to create the dataframe used in the rest of the report. It is imported here because some functions used to organize the data are also helpful in analysis.

```
import os
import sys
import cv2
import numpy as np
import pandas as pd
import math
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.gridspec as gridspec
import re
import ast
import scipy
import seaborn
import math
import joblib
sys.path.append("/mnt/vstor/CSE_MSE_RXF131/cradle-members/mds3/aml334/mds3-advman-2/topics/ar")
import organize_data
```

organize\_data

In addition to these package, the recent release of Meta's foundation models segment anything and segment anything 2 will be explored for use in characterizing the unsegmented initiating defects from high resolution images. [1] [2]

## Exploratory Data Analysis

### Explanation of your data set

The dataset mostly consists of unorganized images in 2 folders, and the numerical data is stored in a handful of spreadsheets. The dataframe was created in two parts: 1) The image dataframe, which contains the path to the images and 2) the numerical dataframe where are spreadsheets were merged. The id used to merge these dataframes is the sample\_id, which was extracted from the image path using a regex and was cleaned from the existing Sample# column in the numerical spreadsheets. For the image dataframes, different categories were made, and each has a corresponding function which takes a file path as input and returns true

for whether the path leads to an image of that category. Data validation is done here to ensure that the categories are well defined.

## Data Cleaning

Data cleaning was an iterative process, with the organize data script being run to create a dataframe, then the results analyzed and verified with a jupyter notebook. The organize data script is in it's own section, and the smaller chunks used to analyze the results are below it. Several sections are not fully cleaned, with the majority of the effort going to the fatigue and overload region, since they are the easiest to verify and should be the simplest task for the model since they are rather large. Some forms of data validation include:

1. Use of a size threshold to validate the separation between stitched low resolution images and individual low resolution images.
2. A lower red pixel threshold to remove pixels in the greyscale images which are identified as colored because of noise and a higher red pixel threshold to identify images where every defect is segmented versus only the initiating defect segmented.

The full data cleaning and formation of the combined\_df can be seen in organize\_data.py

## Dataset Counts

The most important part of this output is the amount of images in the path files. Among the raw images there are 475 high resolution images of the initiating defect and 496 low resolution images of the entire fracture surface. There are far fewer labeled images than unlabeled images.

```
combined_df = pd.read_csv('/mnt/vstor/CSE_MSE_RXF131/lab-staging/mds3/AdvManu/fractography/c
organize_data.print_column_counts(combined_df)
```

Column name	Nulls	Values	dtype	Example
Unnamed: 0	0	7066	int64	0
build_id_x	3632	3434	object	CMU1
build_plate_position_x	3632	3434	object	1
testing_position_x	3632	3434	float64	1.0
sample_id	0	7066	object	CMU1-1-1
cycles	3632	3434	float64	356846.0
test_stress_Mpa	3632	3434	float64	1067.0
scan_power_W	3632	3434	float64	370.0
scan_velocity_mm_s	3632	3434	float64	950.0
energy_density_J_mm3	3632	3434	float64	92.731829573934

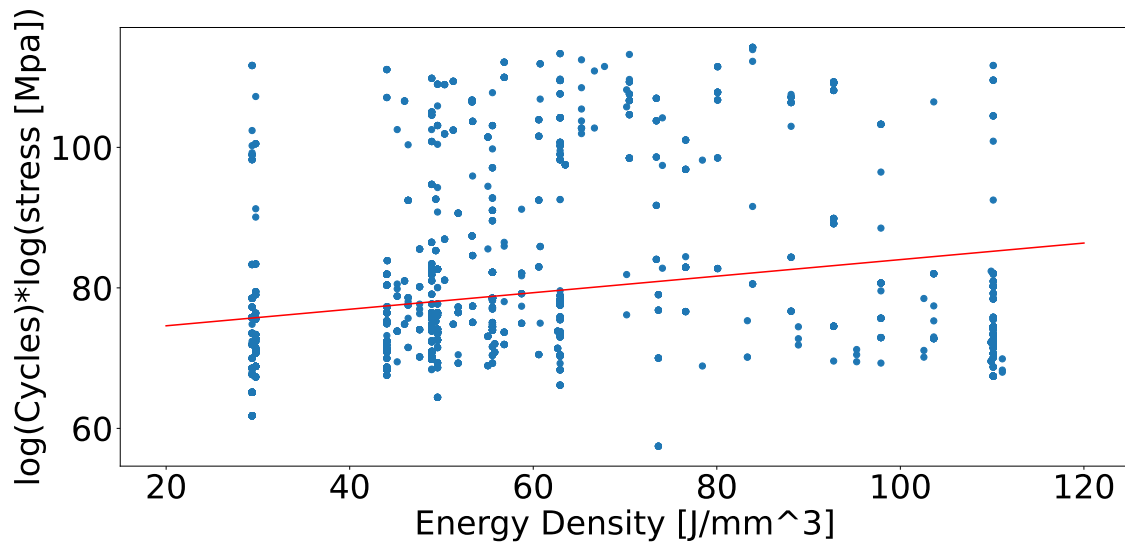
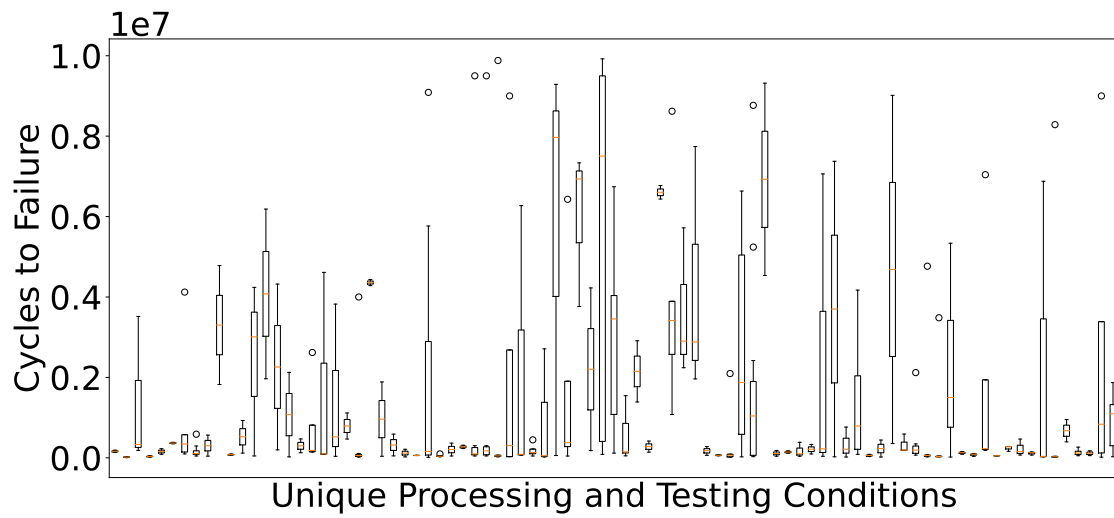
	image_path		110		6956		object		/mnt/vstor/CSE_	
	build_id_y		110		6956		object		CMU1	
	build_plate_position_y		110		6956		object		1	
	testing_position_y		1667		5399		float64		1.0	
	image_basename		110		6956		object		CMU01-1-1 INITI	
	image_class		110		6956		object		valid_image	
	points		5675		1391		object		[[742.0, 181.0]	

```
print(combined_df['image_class'].value_counts())
```

```
image_class
valid_image          4279
stitched             967
full_surface_unmarked 697
initiation           682
overload             112
fatigue              112
full_surface_marked   66
initiation_marked_stitched 41
Name: count, dtype: int64
```

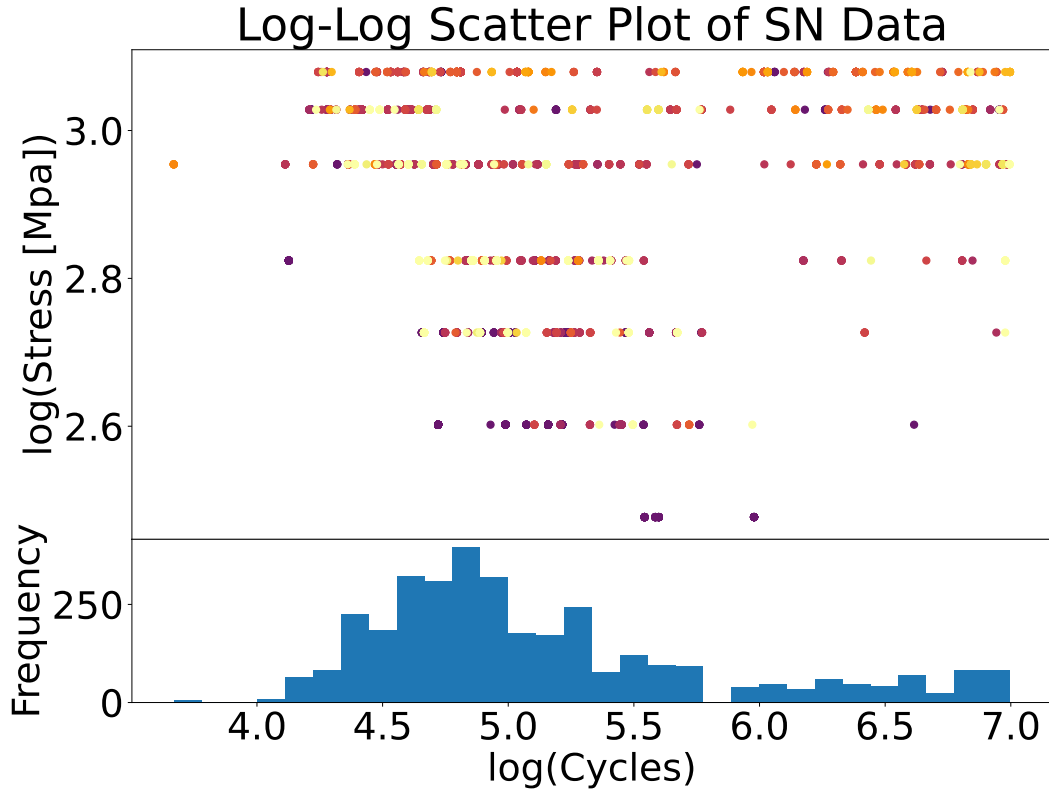
## Difference in Cycles to Failure

Below is a histogram and and scatter plot of the difference of two samples which had identical manufacturing and testing conditions. While this is a



The above figures show that the spreadsheets do not contain all of the necessary information, and from theory, we know that defects may be a leading cause. This is commonly visualized using  $\log(\text{stress})$  vs  $\log(\text{cycle})$  (SN) plots. We can also color the points according to their energy density.





Doing this, we can see that that the purple points, which is the samples with the most energy that have keyhole defects, were generally tested at a lower stress and failed with few cycles, while those with the least energy that have lack of fusion defects failed at higher stress with few cycles. The strongest samples, those tested at the highest stress and lasted the longest tend to be shades of orange. However, we can again see a lot of noise with some purple and yellow points being almost as strong as the other samples.

## Statistical Learning

### Fatigue Region

The fatigue region was selected as the first segmentation task because it is correlated with the fracture toughness, which is determined by the micro structure, which is otherwise unaccounted for. Since a tough micro structure is known to inhibit fatigue crack growth, this is a good candidate for a predictor of Cycles to failure difference. Additionally, the fatigue region is the largest part of the image and is the cleanest of the columns, so is the easiest to work with. The below script was used to train a unet model on segmenting the fatigue region.

Data augmentation was used to make the most of the small dataset. Since all stitched images are of different sizes, the first augmentation must standardize the size. The result was very successful. Initially, a randomized crop of the desired size was used, since that allows for a significantly larger and more diverse dataset than a more traditional resize. However, results in figure 3 show that the model’s training loss did not decrease with epochs, suggesting that the model is incapable of completing this task. All augmentations to the data were investigated, and the result was isolated to the random crop. A training loop with a resize transformation was used instead, shown in figure 4, and the training loss did decrease, though over training was a significant issue. This implies that the relationship of far away pixels is important for performing this task. Self-attention layers are known for their ability to learn global information, so AttentionUnet, a unet architecture which incorporates self-attention layers in skip connections was used, and it significantly outperformed the unet model with the same number of epochs and ultimately converged to a significantly lower training loss. Additionally, the wide interquartile range in both figure 3 and 4 was caused by a single mislabeled image. A well optimized training loop with the mislabeled data point corrected is shown in figure 5 and 6 for u-net and attention unet respectively.

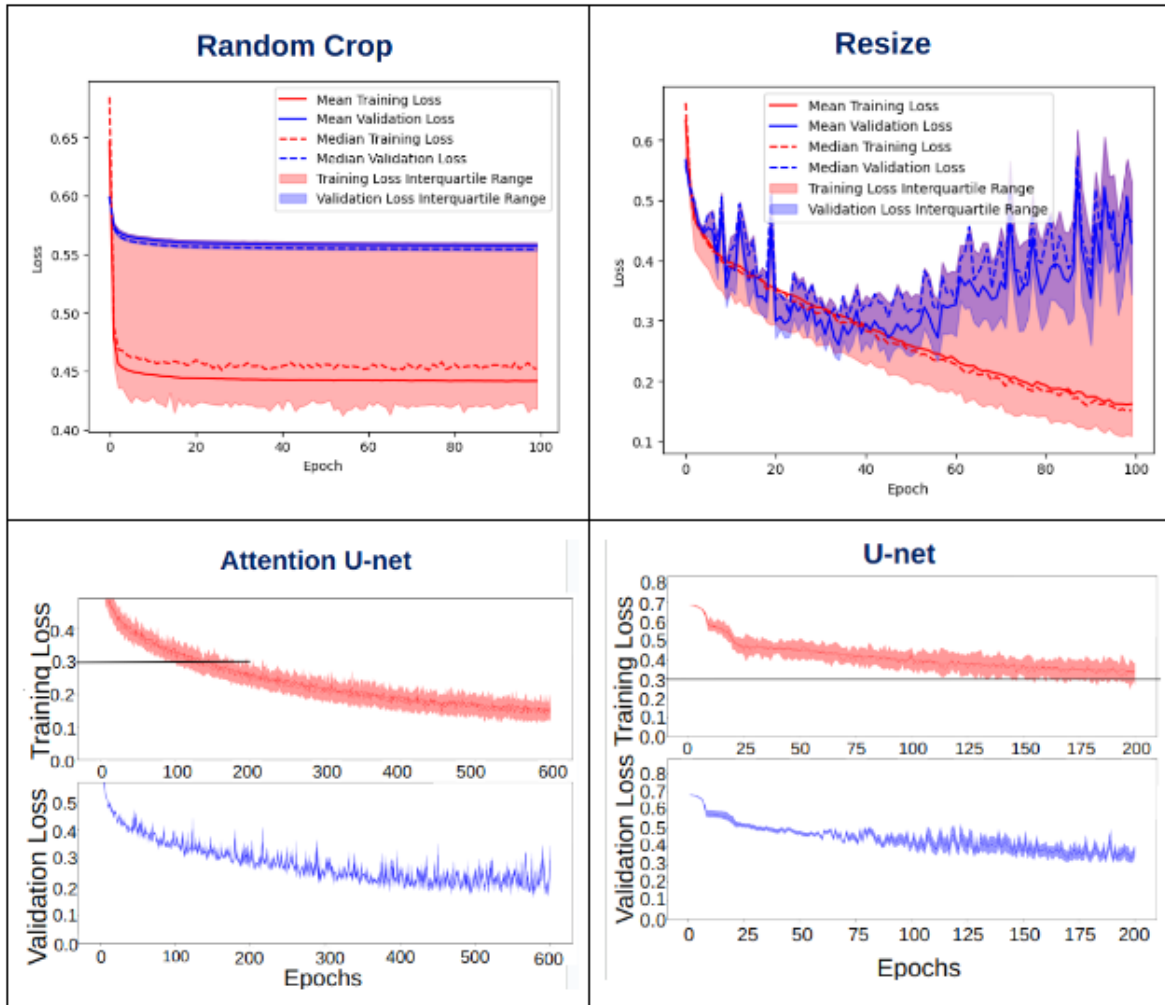


Figure 3: Model Training

## Segment Anything Model on Initiating Defect

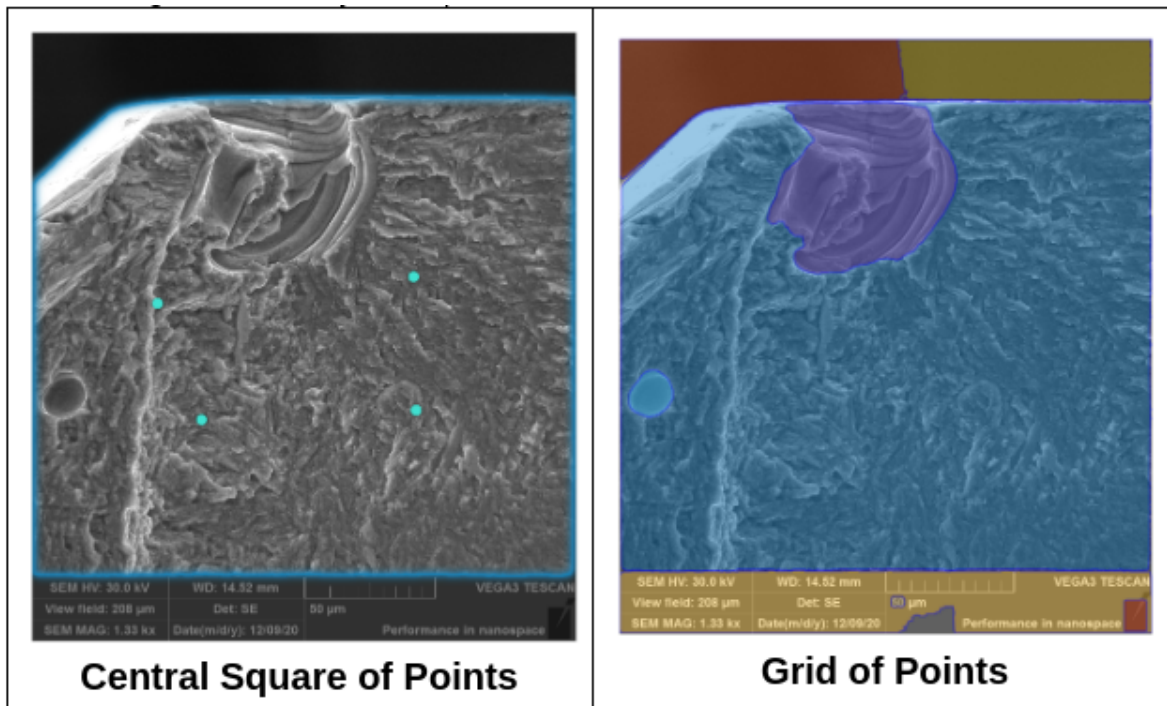


Figure 4: Example Shown for SAM Model

Figure 7 shows the segment anything model being used through it's webpage to segment an example initiating defect image. Based on this result, the surface can be segmented with a square of centralized points and the initiating defect is the largest defect inside of this surface. However, this is a high resolution image with good contrast between the surface, the foreground and the defect. An attempt was made to do this automatically, which was ultimately unsuccessful, so all images were labeled manually. The segmented results are saved as polygons whose points are in the points column of the dataframe.

The columns which were segmented.

```
points_df = combined_df[~combined_df['points'].isna()]
points_df['image_class'].value_counts()
```

```
image_class
valid_image    932
initiation     459
Name: count, dtype: int64
```

```
points_df['sample_id'].value_counts().head(10)
```

```
sample_id
NASA1-83-3    42
EP7-82-2      42
EP7-81-1      22
EP5-71-1      20
EP04-81-1     16
EP4-24-1      15
EP5-84-1      12
NASA1-83-1     12
CMU1-13-1      10
EP5-72-2      10
Name: count, dtype: int64
```

```
for group_string, group in points_df.groupby("sample_id"):
    imgs = group['image_path'].apply(cv2.imread)
    # point_lists = group['image_path'].apply(ast.literal_eval)
```

## Sharpness

From knowledge of fracture mechanics, we know there is a link between the shape of the initiating defect and the resulting fatigue properties, so it should be possible to extract features from these images which predict fatigue performance. The most common feature is the aspect ratio. However, as shown in figure 8, shape 1 and 3 have similar aspect ratio, but shape 1 should create a significantly worse concentrating factor because of the sharp edge in the bottom right corner. Sharpness measures the change in radial distance from the centroid, which better captures these sharp edges, as can be seen by the high value for both shape 1 and 2.




Shape	 1	 2	 3
Maximum Sharpness	0.050	0.057	0.011
Aspect Ratio	1.180	3.990	1.0

Figure 5: Well Optimized Attention Unet Architecture

```
def find_sharpness(numpy_array):
    #Convert array to dataframe
    y,x= np.nonzero(numpy_array)
    df = pd.DataFrame({'x':x,'y':y})
    x0 = df['x'].sum()/df['x'].count()
    y0 = df['y'].sum()/df['y'].count()
    #Calculate polar coordinates
    df['x_rel'] = df['x'] - x0
    df['y_rel'] = df['y'] - y0
    df['angle'] = df.apply(lambda row:math.atan(row['y_rel']/row['x_rel']),axis=1)
    df['distance'] = df.apply(lambda row:math.sqrt(row['y_rel']**2 + row['x_rel']**2),axis=1)
    global_max = df['distance'].max()
    #Find max for each bin
    num_bins = 180
    bin_edges = np.linspace(-math.pi/2, math.pi/2, num_bins + 1)
    bins = pd.IntervalIndex.from_breaks(bin_edges,name='Angle_bin')
    df.index = pd.cut(df['angle'],bins)
    max_df = df.groupby(level=0,observed=False)['distance'].max()
    max_diff = []
    for i in range(0,len(max_df)-2):
        max_diff.append(abs(max_df.iloc[i]-max_df.iloc[i+1])/global_max)

    return max_diff
```

```

def calculate_aspect_ratio(mask):
    # Find the non-zero mask coordinates
    y_coords, x_coords = np.where(mask > 0)

    # Calculate the bounding box dimensions
    height = y_coords.max() - y_coords.min() + 1
    width = x_coords.max() - x_coords.min() + 1

    # Calculate the aspect ratio
    aspect_ratio = width / height
    return aspect_ratio

def invert_colors(image):
    #From claude 3 Haiku
    """
    Invert the colors of the image.
    For 8-bit images, use 255 - image.
    For floating point images (0 to 1), use 1 - image.
    """
    if image.dtype == np.uint8:
        return 255 - image
    else:
        return 1.0 - image

img_1 = invert_colors(cv2.imread('img_1.png', cv2.IMREAD_GRAYSCALE))
img_2 = invert_colors(cv2.imread('img_2.png', cv2.IMREAD_GRAYSCALE))
img_3 = invert_colors(cv2.imread('img_3.png', cv2.IMREAD_GRAYSCALE))

print('Image 1')

```

Image 1

```
print('Max Sharpness: ' + str(max(find_sharpness(img_1))))
```

Max Sharpness: 0.05024144191730347

```
print('Aspect ratio: ' + str(calculate_aspect_ratio(img_1)))
```

Aspect ratio: 1.1806167400881058

```
print('Max Sharpness: '+str(max(find_sharpness(img_2))))
```

Max Sharpness: 0.05715282591919487

```
print('Aspect ratio: '+str(calculate_aspect_ratio(img_2)))
```

Aspect ratio: 3.9904761904761905

```
print('Max Sharpness: '+str(max(find_sharpness(img_3))))
```

Max Sharpness: 0.011759710746858592

```
print('Aspect ratio: '+str(calculate_aspect_ratio(img_3)))
```

Aspect ratio: 1.0

```
save_path = '/mnt/vstor/CSE_MSE_RXF131/lab-staging/mds3/AdvManu/fractography/SAM_whole_surfa
```

```
def load_SAM__segmentation():
    paths_list = []
    basename_lists = []
    for path in os.listdir(save_path):
        if 'seg' in path:
            paths_list.append(save_path+'/'+path)
            basename_lists.append(path.removeprefix("whole_surface_seg_"))
    return pd.concat([pd.Series(paths_list,name='image_path'),pd.Series(basename_lists,name=
segmented_df = load_SAM__segmentation()
```

```
# print(combined_df['image_basename'].drop_duplicates().value_counts())
for group_string, basenames in combined_df.groupby('image_basename'):
    if 'stitched' in sample['image_class'].value_counts().index:
```

```
for group_string, group in combined_df.groupby('sample_id'):
    no_points = group[(group['image_class']=='initiation') & (group['points'].isna())]
    points = group[(group['image_class']=='initiation') & (~group['points'].isna())]
    if not (len(no_points.index)>=1 and len(points.index)>=1):
        if len(no_points.index)>=1:
            cycles =no_points['cycles'].iloc[0]
```



```

        stress =no_points['test_stress_Mpa'].iloc[0]
        x_not_SAM_test.append(math.log(cycles)*math.log(stress))
    elif len(points.index)>=1:
        cycles =points['cycles'].iloc[0]
        stress =points['test_stress_Mpa'].iloc[0]
        x_SAM_test.append(math.log(cycles)*math.log(stress))

```

```

def process_mask(mask):
    # Find connected components
    num_labels, labels = cv2.connectedComponents(mask)

    # Count components (excluding background)
    num_objects = num_labels - 1

    # Find largest object
    largest_object_mask = np.zeros_like(mask)
    if num_objects > 0:
        # Get unique labels, excluding background (0)
        label_counts = [np.sum(labels == i) for i in range(1, num_labels)]
        largest_object_label = np.argmax(label_counts) + 1
        largest_object_mask = (labels == largest_object_label).astype(np.uint8) * 255

    return num_objects, largest_object_mask

def extract_largest_object(img,points_list):
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    cv2.fillPoly(
        mask,
        [np.array(points_list, dtype=np.int32)],
        255
    )
    img_object, largest_object_mask = process_mask(mask)
    return largest_object_mask

def find_sharpness(numpy_array):
    #Convert array to dataframe
    y,x= np.nonzero(numpy_array)
    df = pd.DataFrame({'x':x,'y':y})
    x0 = df['x'].sum()/df['x'].count()
    y0 = df['y'].sum()/df['y'].count()
    #Calculate polar coordinates

```

```

df['x_rel'] = df['x'] - x0
df['y_rel'] = df['y'] - y0
df['angle'] = df.apply(lambda row:math.atan(row['y_rel']/row['x_rel']),axis=1)
df['distance'] = df.apply(lambda row:math.sqrt(row['y_rel']**2 + row['x_rel']**2),axis=1)
global_max = df['distance'].max()
#Find max for each bin
num_bins = 180
bin_edges = np.linspace(-math.pi/2, math.pi/2, num_bins + 1)
bins = pd.IntervalIndex.from_breaks(bin_edges,name='Angle_bin')
df.index = pd.cut(df['angle'],bins)
max_df = df.groupby(level=0,observed=False)['distance'].max()
max_diff = []
for i in range(0,len(max_df)-2):
    max_diff.append(abs(max_df.iloc[i]-max_df.iloc[i+1])/global_max)

    return max(max_diff)
def calculate_aspect_ratio(mask):
    # Find the non-zero mask coordinates
    y_coords, x_coords = np.where(mask > 0)

    # Calculate the bounding box dimensions
    height = y_coords.max() - y_coords.min() + 1
    width = x_coords.max() - x_coords.min() + 1

    # Calculate the aspect ratio
    aspect_ratio = width / height
    return aspect_ratio

def get_perimeter(binary_image):
    # Ensure the image is binary
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # If there are contours, return the perimeter of the largest contour
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        return cv2.arcLength(largest_contour, closed=True)

    return 0

def calculate_aspect_ratio_rotated(binary_image):
    # Find contours
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

```

```

    if not contours:
        return None

    # Get the largest contour
    largest_contour = max(contours, key=cv2.contourArea)

    # Get the rotated rectangle
    rect = cv2.minAreaRect(largest_contour)
    (width, height) = rect[1]

    # Calculate aspect ratio (ensuring width is always the larger dimension)
    aspect_ratio = max(width, height) / min(width, height)

    return aspect_ratio

points_df = combined_df[~combined_df['points'].isna()]
imgs = []
energy = []
stress = []
print(points_df.columns)
for row in points_df.iterrows():
    imgs.append(
        extract_largest_object(
            cv2.imread(row[1]['image_path']),
            ast.literal_eval(row[1]['points'])
        )
    )
    energy.append(row[1]['energy_density_J_mm3'])
    stress.append(row[1]['test_stress_Mpa'])

portion_of_screen = joblib.Parallel(n_jobs=-1)(joblib.delayed(lambda x: x.sum() / (x.size * 1000))(x) for x in imgs)
max_sharpness = joblib.Parallel(n_jobs=-1)(joblib.delayed(find_sharpness)(x) for x in imgs)
aspect_ratio = joblib.Parallel(n_jobs=-1)(joblib.delayed(calculate_aspect_ratio_rotated)(x) for x in imgs)
perimeter = joblib.Parallel(n_jobs=-1)(joblib.delayed(get_perimeter)(x) for x in imgs)
pixels = joblib.Parallel(n_jobs=-1)(joblib.delayed(lambda x: x.sum() / x.max())(x) for x in imgs)
pixel_perimeter_ratio = joblib.Parallel(n_jobs=-1)(joblib.delayed(lambda x: x.sum() / (x.max() * 1000))(x) for x in imgs)
data = {
    "screen_portion": portion_of_screen,
    "max_sharpness": max_sharpness,
    "aspect_ratio": aspect_ratio,
    "perimeter": perimeter,
    "pixels": pixels,
    "pixel_perimeter_ratio": pixel_perimeter_ratio
}

```

```

    "pixel_perimeter_ratio": pixel_perimeter_ratio,
    "energy_density": energy,
    "stress Mpa": stress
}
df = pd.DataFrame(data).dropna()

def annotate_r2(x, y, **kwargs):
    slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(x, y)
    r_squared = r_value ** 2
    ax = plt.gca()
    ax.annotate(f"$R^2$ = {r_squared:.2f}", xy=(0.6, 0.9), xycoords=ax.transAxes, fontsize=20)

# Create pairplot with linear regression
g = seaborn.pairplot(df, kind="reg", plot_kws={"line_kws": {"color": "red"}}, hue="energy_density")

# Adjust the size of axis labels and ticks using matplotlib
for ax in g.axes.flatten():
    ax.set_xlabel(ax.get_xlabel(), fontsize=14) # Set the font size of x-axis labels
    ax.set_ylabel(ax.get_ylabel(), fontsize=14) # Set the font size of y-axis labels
    ax.tick_params(axis='both', labelsize=12) # Set the font size of ticks
# Add R-squared annotations to each plot
g.map(annotate_r2)

```

## Citations

- [1] A. Kirillov *et al.*, “Segment Anything.” arXiv, 2023. doi: [10.48550/ARXIV.2304.02643](https://arxiv.org/abs/2304.02643).
- [2] N. Ravi *et al.*, “SAM 2: Segment anything in images and videos,” *arXiv preprint arXiv:2408.00714*, 2024, Available: <https://arxiv.org/abs/2408.00714>