Pump Impeller Quality Control Using CNN

Hamed Aarab	9925003
Fateme Khodabande	9825011
Rosha Moshtaghian	9825080

1

The Importance of Quality Control

Image processing in casting manufacturing product

2

Chosen Product

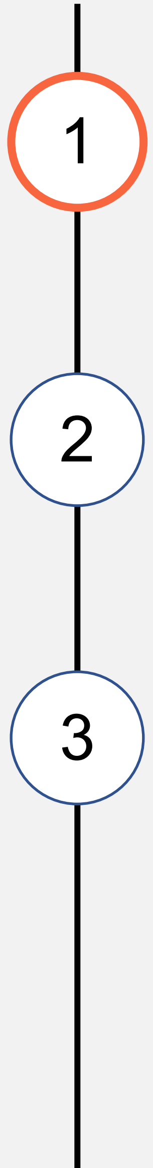
Submersible pump impeller

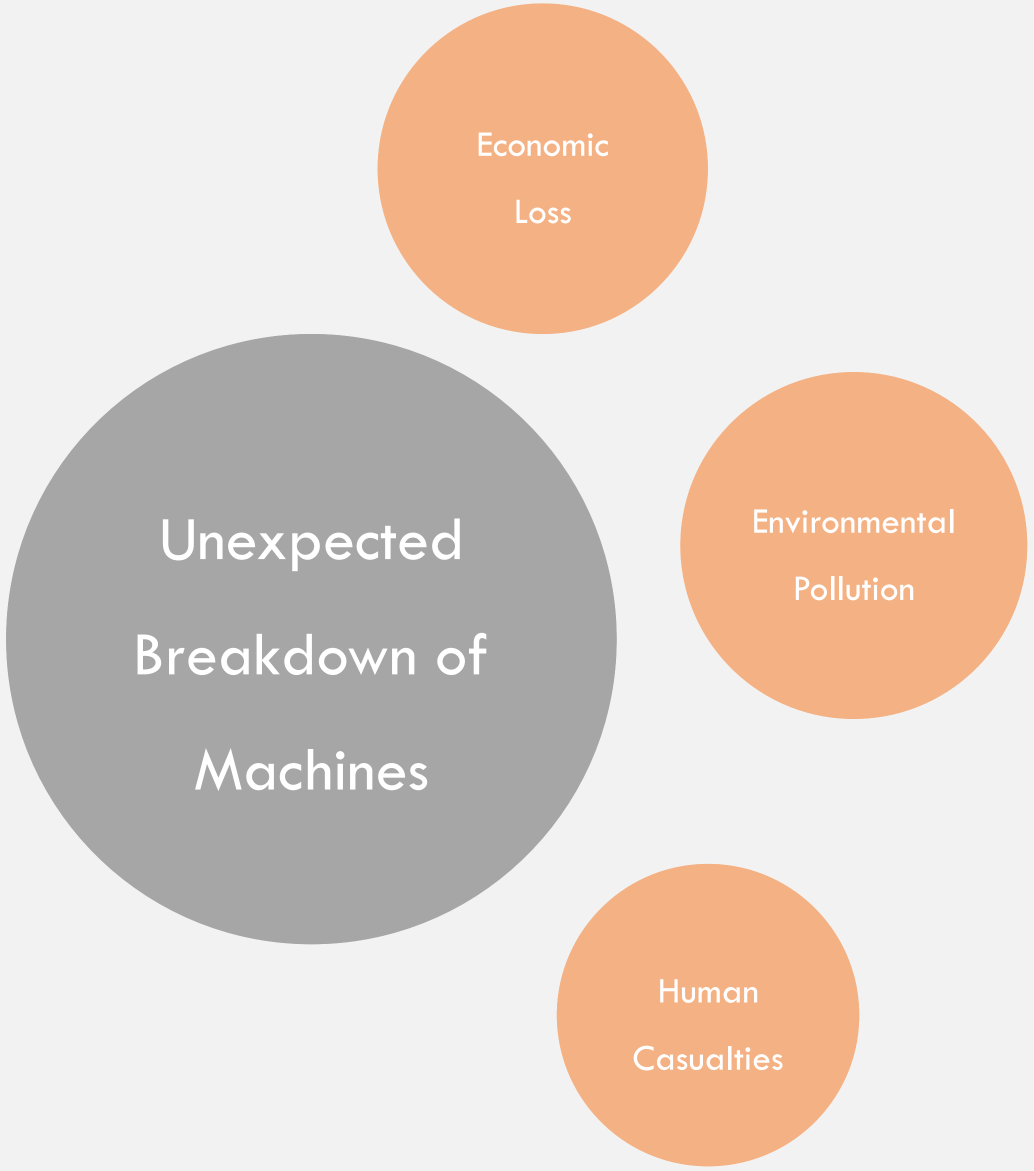
3

The CNN Model

Dataset, prerequisites, data visualization, data preprocessing, model definition, training, and testing

Rotating machines are widely used in manufacturing industry, operating usually for long time under harsh conditions.



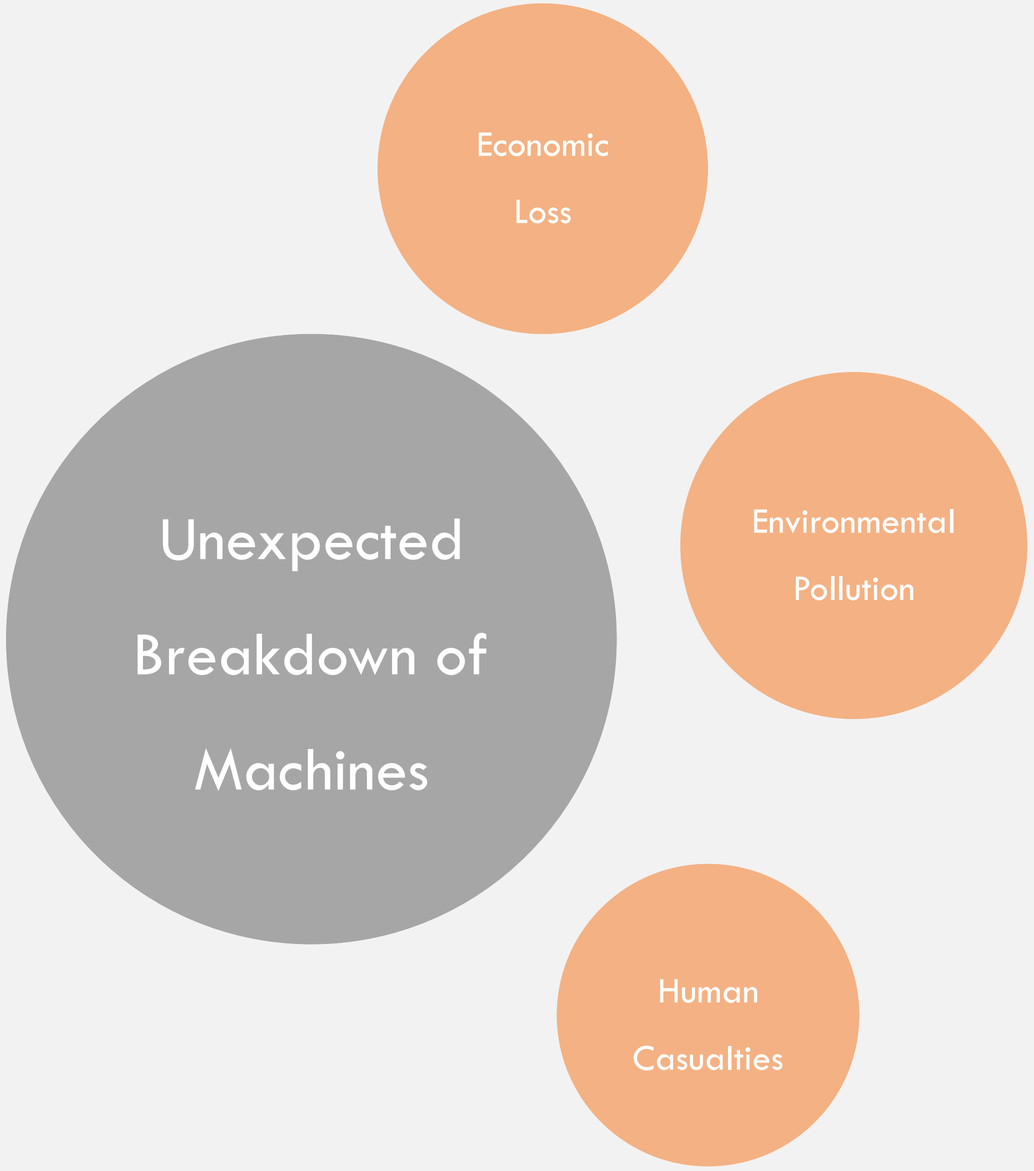


Early and accurate detection of defects and failures of such components of rotating machinery is critical to ensure operational reliability and avoid catastrophic accidents in industrial applications.

1

2

3



1

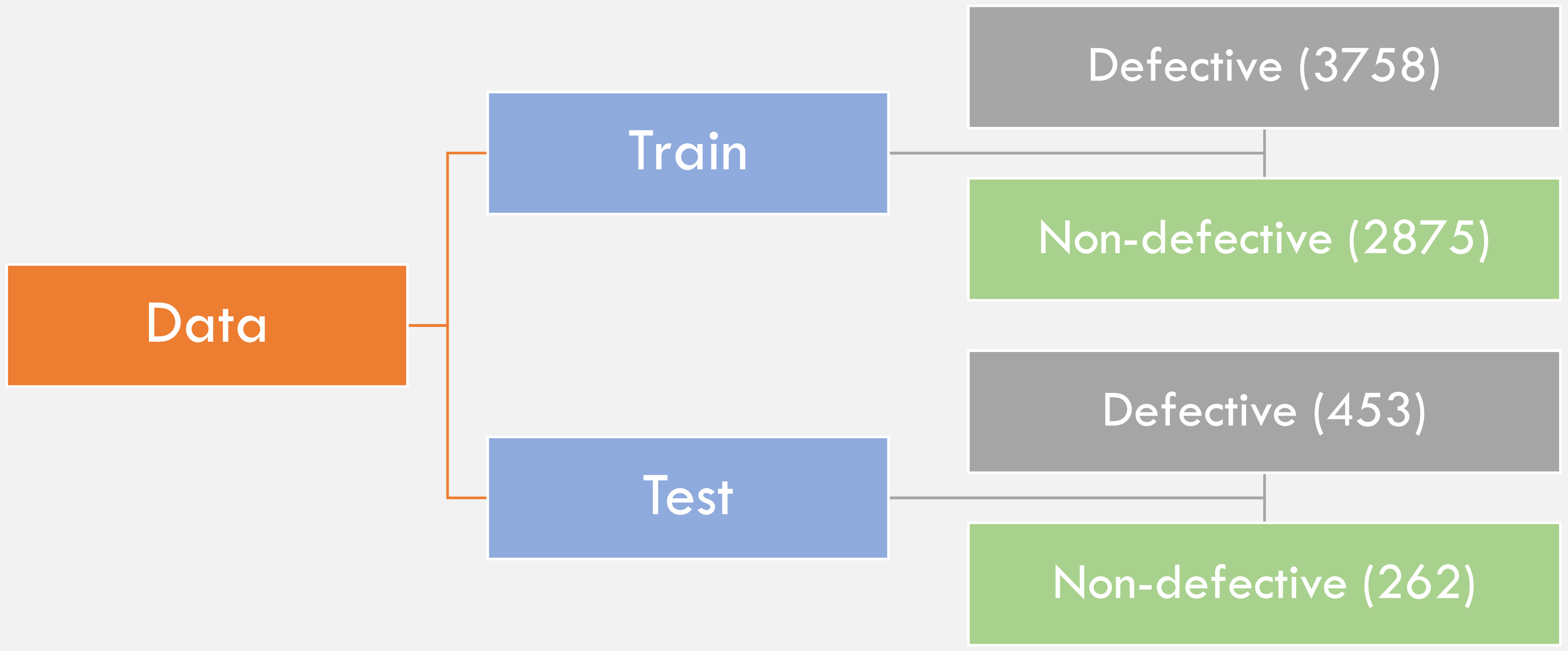
2

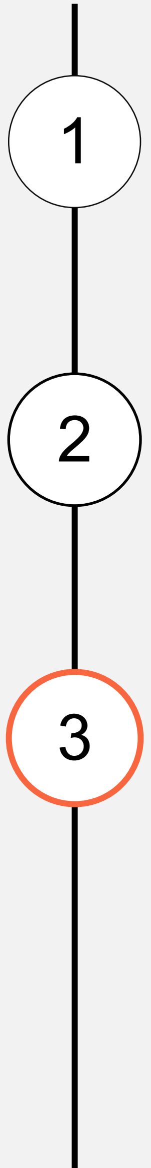
3

Dataset

The data we used are available on Kaggle (https://www.kaggle.com/ravirajsinh45/real-life-industrial-dataset-of-casting-product).

The dataset contains 7348 300x300 grey-scaled images in total. Augmentation is already applied to the images.





About CNN

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

Project Description

This project is basically a classification problem. Since we are dealing with images, we have decided to solve it with a convolutional neural network due to its abilities in image processing.

Prerequisites

The libraries we need are:

- Tensorflow for defining the CNN model.
- Numpy for transforming arrays.
- Matplotlib for visualizing images.
- VisualKeras for visualizing our model.

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import visualkeras as vk
```

1

2

3

Data Visualization

First, we define a function for loading images in the gray-scale mode.

Then, we define another function to get an image and visualize it using Matplotlib. This function transforms the image into an array and rescales its values to [0, 1].

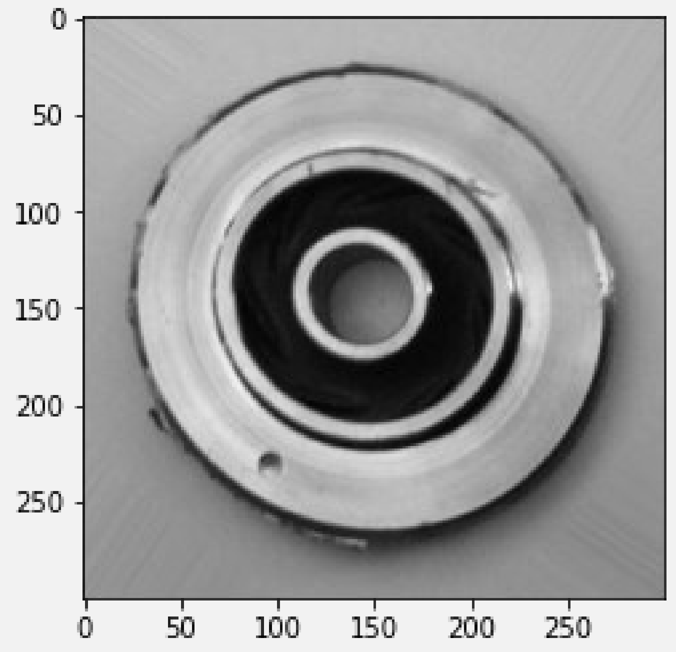
```python
def get_image(path):
    return tf.keras.preprocessing.image.load_img(path, color_mode = 'grayscale')

def visualize_image(image):
    plt.figure()

    plt.imshow(
        tf.keras.preprocessing.image.img_to_array(image) / 255,
        cmap = 'gray',
    )

    plt.show()
```

1

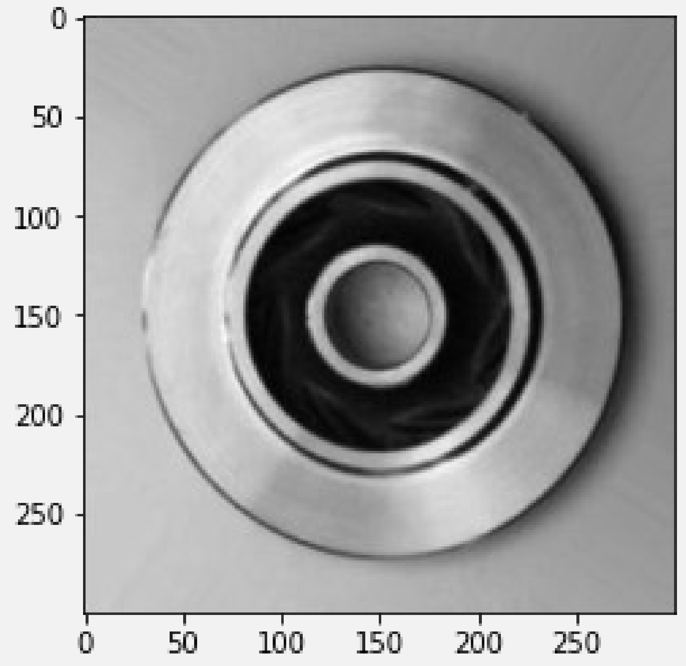
2

3

Defective Image Example



Non-Defective Image Example



1

2

3

Data Preprocessing

Now, we need to pre-process our data by loading and rescaling the train data and the test data separately. To improve our model's processing abilities, we randomly zoom and shear the train data in order to simulate seeing objects from different angles and distances.

```
train_data = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale = 1 / 255,
    zoom_range = 0.2,
    shear_range = 0.2,
).flow_from_directory(
    './data/train',
    class_mode = 'binary',
    batch_size = 8,
    target_size = (64, 64),
    color_mode = 'grayscale',
)

test_data = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale = 1 / 255,
).flow_from_directory(
    './data/test',
    #...
)
```

1

2

3

Defining the CNN Model

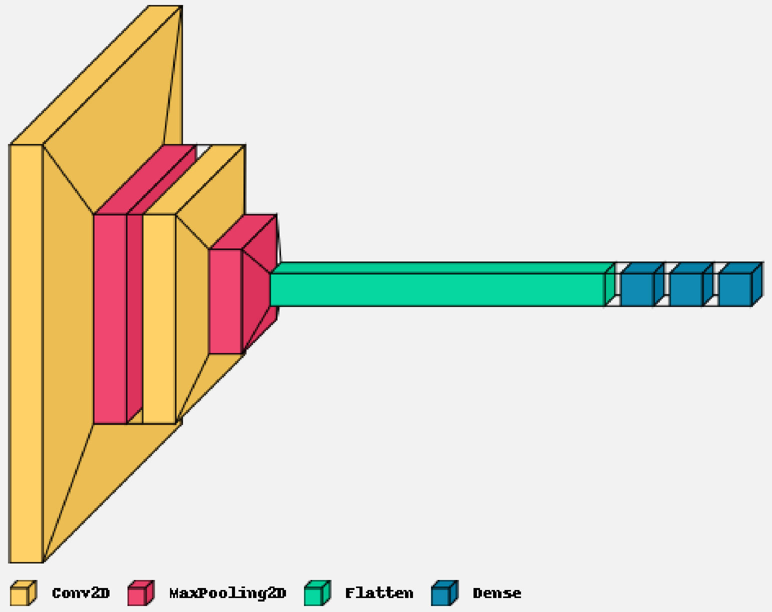
Our CNN model is going to have two series of convolution and max-pooling layers. Then, we flatten the output and pass it to an MLP with two hidden layers.

Finally, we compile our model with the adam optimizer and binary cross entropy loss function.

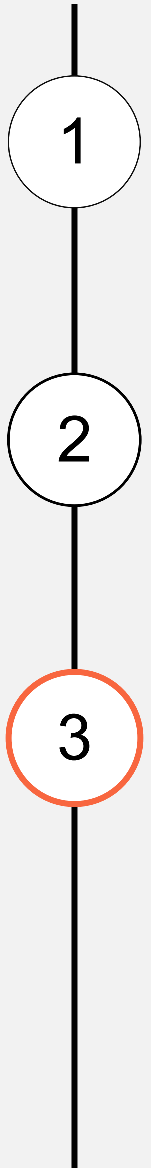
1

2

3



```python
model = tf.keras.models.Sequential()

model.add(
    tf.keras.layers.Conv2D(
        filters = 8,
        kernel_size = 3,
        activation = 'relu',
        padding = 'same',
        input_shape = (64, 64, 1),
    )
)

model.add(tf.keras.layers.MaxPooling2D(pool_size = 2))

model.add(
    tf.keras.layers.Conv2D(
        filters = 8,
        kernel_size = 3,
        activation = 'relu',
        padding = 'same',
    )
)

model.add(tf.keras.layers.MaxPooling2D(pool_size = 2))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(units = 16, activation = 'tanh'))
model.add(tf.keras.layers.Dense(units = 16, activation = 'relu'))

model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

model.compile(
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics = ['binary_accuracy'],
)

vk.layered_view(model, legend = True)
```

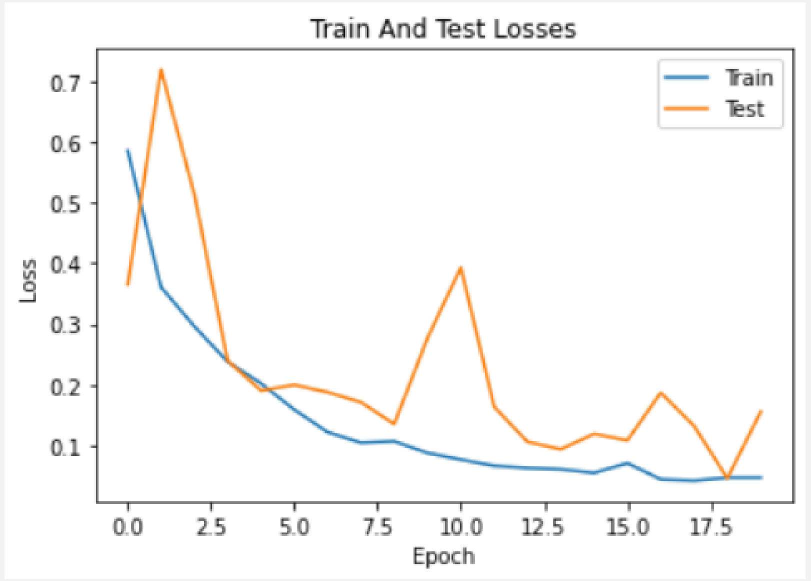


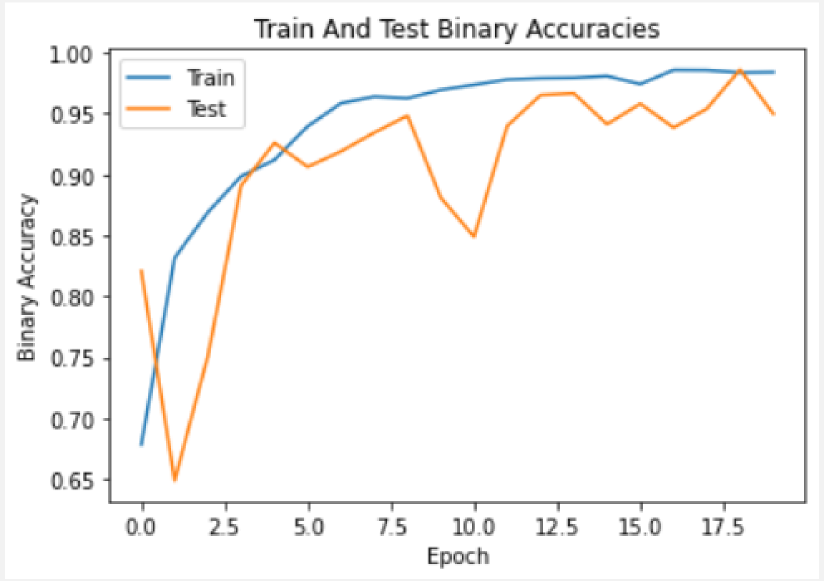
Training and Testing the CNN Model

We train our model with 20 epochs and save the one with the highest validation accuracy.

Here, we plot our model's accuracy over all of the epochs.

```
history = model.fit(train_data, validation_data = test_data, epochs = 20)

model.save('classifier_model.h5')
```





Our CNN model can classify images of pump impeller with an accuracy of ~98%.

1

2

3

Thank You!

Any Questions?