“CASTING PRODUCT DEFECT DETECTION”

A Report

Submitted as special assignment

of

2ICOE02 MACHINE VISION

By
Kosh Rai (20BCE131)

Under the Guidance of
Prof. Harsh Kapadia

INSTRUMENTATION AND CONTROL ENGINEERING
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY
Ahmedabad 382 481

NOVEMBER 2023

SOFTWARE SIMULATION

Casting Product Defect Detection

1.1 Introduction

The aim of this project is to detect defects in casting products. The defects are detected using a 2 dimensional convolutional neural network which takes in images of casting products as the input, and classifies them into two classes: Defective and Okay as the output. The trained neural network is integrated into an intuitive graphical user interface for ease of use.

1.2 Dataset

The dataset consisted of top-view images of submersible pump impellers. Each image had spatial dimensions of 300x300 and had 3 (R, G, B) color channels. Some examples of the images can be seen in figure 1.

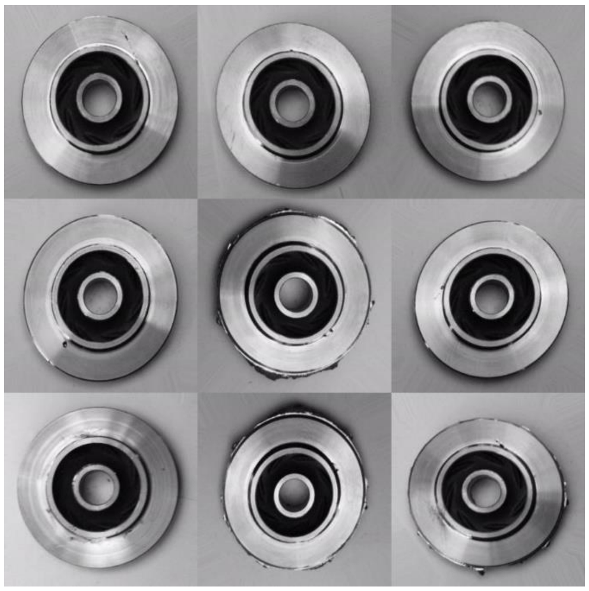


FIGURE 1

1.2 Network Architechture and Training

The CNN itself is the most crucial part of the developed application. The architecture for the CNN is loosely inspired by the VGG16 and VGG19 architechtures, in the sense that it stacks multiple

convolution layers of a low receptive field to effectively achieve a larger receptive field while limiting the number of learnable parameters, and thereby cutting down on the computational cost.
The block diagram of the convolutional neural network can be seen in figure 2.

The network also utilizes two 2D dropout layers (with p=0.2) for regularization.

The network was implemented using the PyTorch framework and trained using a Nvidia Tesla T4 tensor core GPU (available for free on Kaggle). The training process took 2 hours and 42 minutes. The network was trained for 50 epochs with a batch size of 32 and a learning rate of 0.05. The loss function was categorical crossentropy, and was optimized using the stochastic gradient descent optimization algorithm. The loss curves for the training and validation set are shown in figure 3.

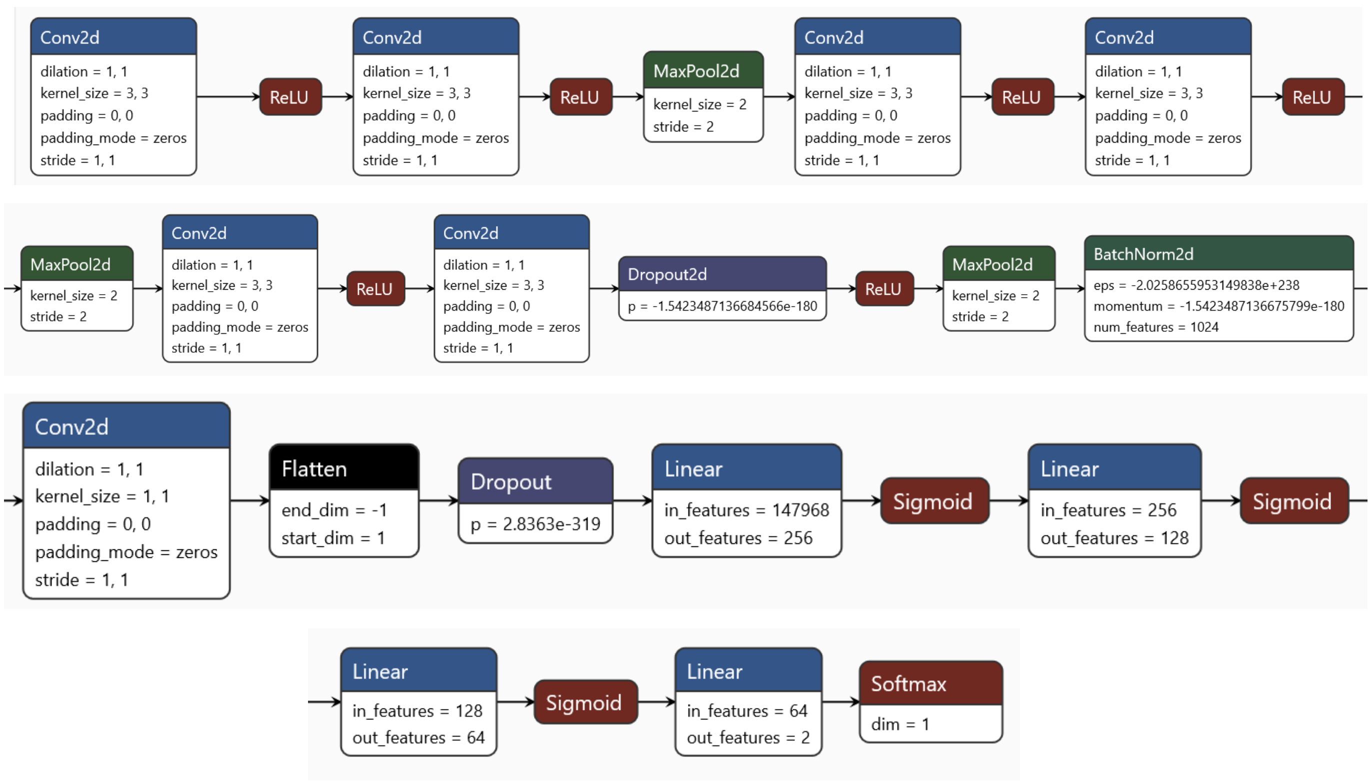


FIGURE 2: MODEL ARCHITECTURE

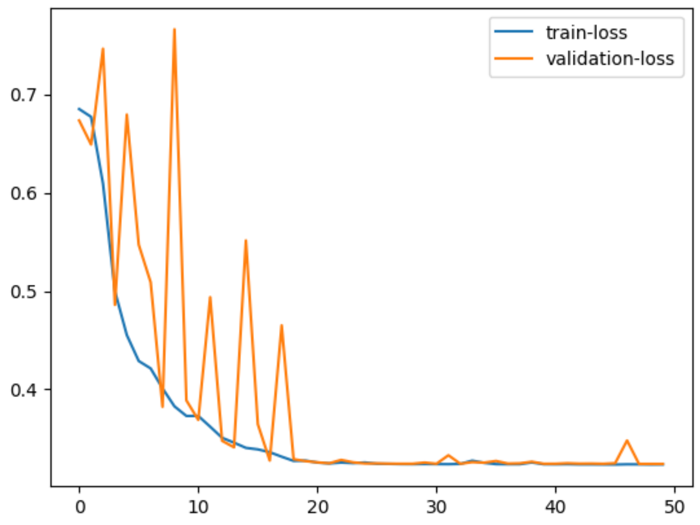
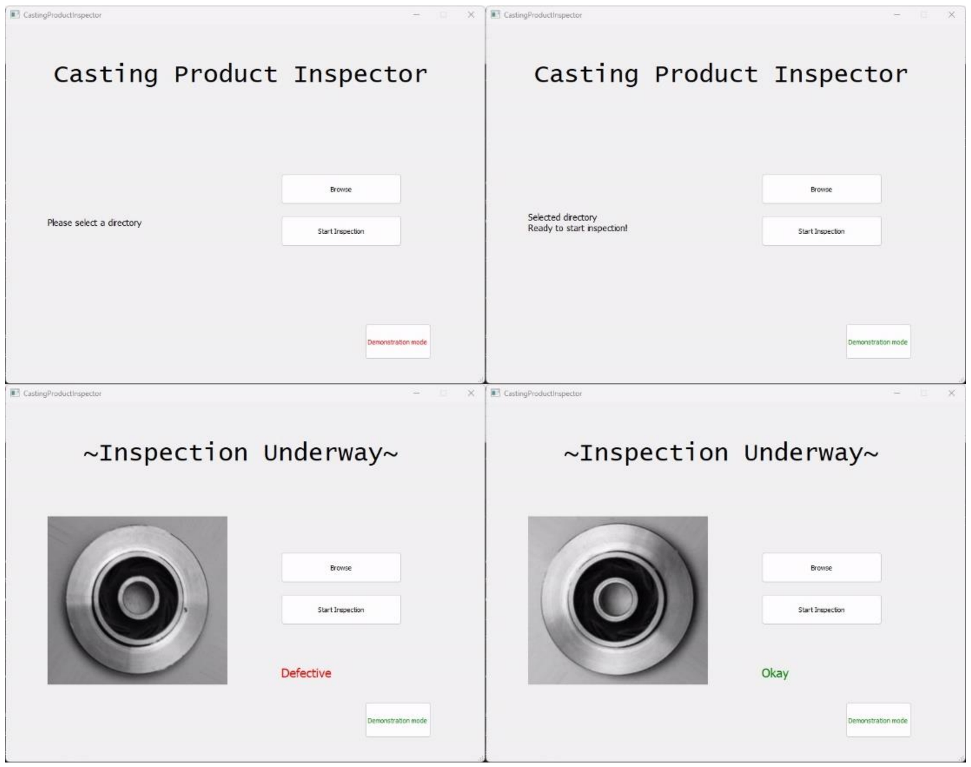


FIGURE 3: LOSS VS EPOCHS

1.3 Graphical user interface

A neural network was integrated into a graphical user interface for ease of usage. The graphical user interface allows the user to run the classification algorithm on a directory containing multiple images with relative ease. The GUI can be seen in action in figure 4.



Casting Product Inspector

Please select a directory

Casting Product Inspector

Selected directory
Ready to start inspection!

~Inspection Underway~

Defective

~Inspection Underway~

Okay

FIGURE 4

The GUI also made the provision of saving the results from a directory scan as a .csv file onto the disc. The csv file contains two columns, one for the filename of the particular image, the second for the classification output.

Results

2.1 Results

The convolutional neural network achieved 98.8% accuracy on the test set, incorrectly classifying only 8 images in a set of 715. The confusion matrix for this classification process can be seen in figure 5.

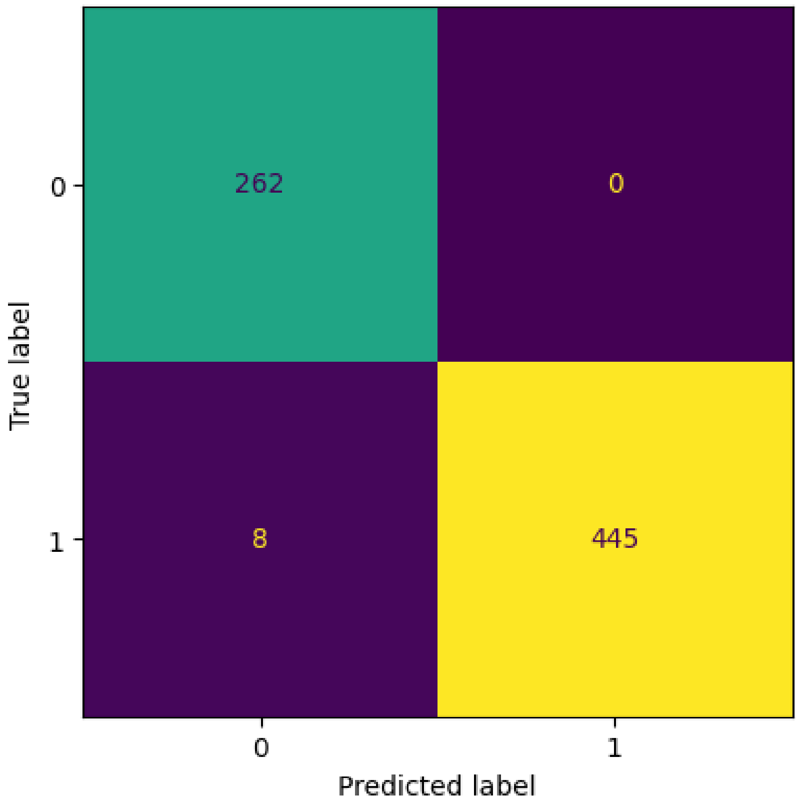


Figure 5, Note that the label 1 signifies defective

2.2 Challenges and Future scope of work

One of the biggest challenges of any deep learning based approach is finding the right dataset. However, the second biggest problem is having, and effectively utilising the computing resources that are required to train a good model. The problem with using the free version of Google Colab is that if the number of concurrent users with a subscription plan increases, your program may stop executing to serve the paying customer. Since the training process in this application took nearly 3 hours, using Google Colab would have been a risky approach. With Kaggle, not only do you get access to two Nvidia Tesla T4 gpus (compared to one on Colab), but you can use them for upto 30 hours every week reliably.

In the future, using unsupervised learning techniques or deterministic image processing algorithms, the dataset could be expanded to include different types of defects. This could extend the usage of the application to not only classify whether the product is defective, but also classify the type of defect. In the run, a company could use this data to try and identify which types of defects are most common, and try to find a way to reduce the rates of these defects.

Appendix I

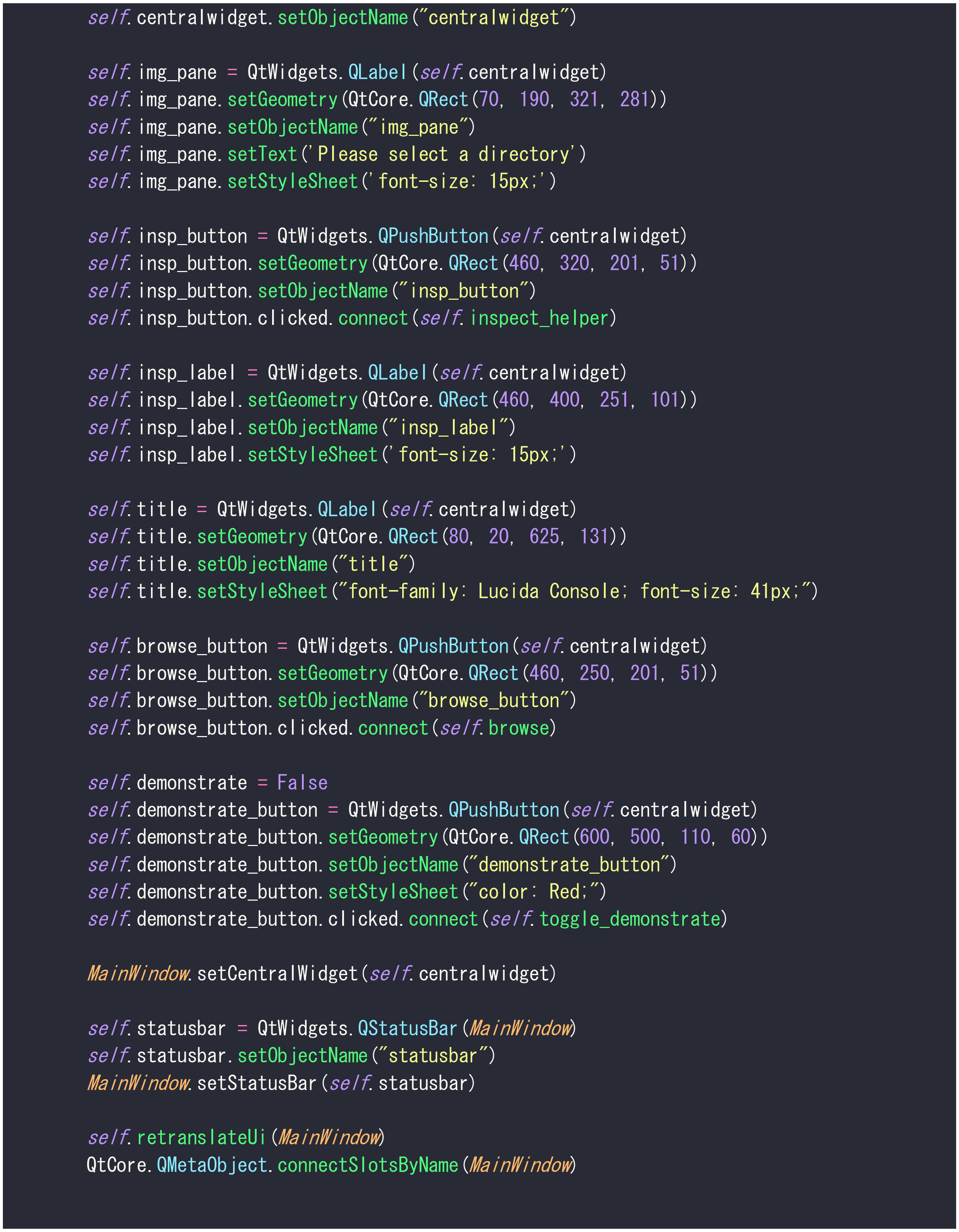
The code is divided into two files, plus two seperate IPython notebooks for training and evaluation of the model.

main.py – The contains driver code for the GUI

```python
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog
import numpy as np
import os
import torch
from Classifier import Classifier
import cv2 as cv
import time
from threading import Thread
import pandas as pd
import time


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.setEnabled(True)
        MainWindow.setFixedSize(800, 600)
        MainWindow.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

        self.centralwidget = QtWidgets.QWidget(MainWindow)
```



```
        self.centralwidget.setObjectName("centralwidget")

        self.img_pane = QtWidgets.QLabel(self.centralwidget)
        self.img_pane.setGeometry(QtCore.QRect(70, 190, 321, 281))
        self.img_pane.setObjectName("img_pane")
        self.img_pane.setText('Please select a directory')
        self.img_pane.setStyleSheet('font-size: 15px;')

        self.insp_button = QtWidgets.QPushButton(self.centralwidget)
        self.insp_button.setGeometry(QtCore.QRect(460, 320, 201, 51))
        self.insp_button.setObjectName("insp_button")
        self.insp_button.clicked.connect(self.inspect_helper)

        self.insp_label = QtWidgets.QLabel(self.centralwidget)
        self.insp_label.setGeometry(QtCore.QRect(460, 400, 251, 101))
        self.insp_label.setObjectName("insp_label")
        self.insp_label.setStyleSheet('font-size: 15px;')

        self.title = QtWidgets.QLabel(self.centralwidget)
        self.title.setGeometry(QtCore.QRect(80, 20, 625, 131))
        self.title.setObjectName("title")
        self.title.setStyleSheet("font-family: Lucida Console; font-size: 41px;")

        self.browse_button = QtWidgets.QPushButton(self.centralwidget)
        self.browse_button.setGeometry(QtCore.QRect(460, 250, 201, 51))
        self.browse_button.setObjectName("browse_button")
        self.browse_button.clicked.connect(self.browse)

        self.demonstrate = False
        self.demonstrate_button = QtWidgets.QPushButton(self.centralwidget)
        self.demonstrate_button.setGeometry(QtCore.QRect(600, 500, 110, 60))
        self.demonstrate_button.setObjectName("demonstrate_button")
        self.demonstrate_button.setStyleSheet("color: Red;")
        self.demonstrate_button.clicked.connect(self.toggle_demonstrate)

        MainWindow.setCentralWidget(self.centralwidget)

        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

```python
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "CastingProductInspector"))
        self.img_pane.setText(_translate("MainWindow", "Please select a directory"))
        self.insp_button.setText(_translate("MainWindow", "Start Inspection"))
        self.insp_label.setText(_translate("MainWindow", ""))
        self.title.setText(_translate("MainWindow", "Casting Product Inspector"))
        self.browse_button.setText(_translate("MainWindow", "Browse"))
        self.demonstrate_button.setText(_translate("MainWindow", "Demonstration mode"))

    def toggle_demonstrate(self):
        self.demonstrate = not self.demonstrate
        if self.demonstrate: self.demonstrate_button.setStyleSheet("color: Green;")
        else: self.demonstrate_button.setStyleSheet("color: Red;")

    def browse(self):
        try:
            self.path = str(QFileDialog.getExistingDirectory(None, "Select Directory"))
            self.imgs = np.array([img for img in os.listdir(self.path)])
            self.img_pane.setText(f'Selected directory¥nReady to start inspection!')
        except:
            return


    def inspect_helper(self):
        t1=Thread(target=self.inspect)
        t1.start()

    def inspect(self):
        d = {'Image':[], 'Inspection result':[]}
        model = torch.load('model.pt', map_location=torch.device('cpu'))
        print('Loaded model')
        self.title.setText('  ~Inspection Underway~')
        try:
            np.random.shuffle(self.imgs)
        except AttributeError:
            msg_box = QtWidgets.QMessageBox()
            msg_box.setIcon(QtWidgets.QMessageBox.Warning)
            msg_box.setText("Please select a directory first!")
            msg_box.setWindowTitle("No directory selected")
            msg_box.exec_()
            return
        defective_count, okay_count = 0, 0
        t = time.process_time()
        for idx, img_path in enumerate(self.imgs):
            try:
                d['Image'].append(img_path)
                img_path = os.path.join(self.path, img_path)
                img = cv.cvtColor(cv.imread(img_path), cv.COLOR_BGR2RGB)
                print(f'{idx}: {img_path}')
                self.img_pane.setPixmap(QtGui.QPixmap(img_path))
                self.insp_label.setText('')
            except:
                d['Image'].pop()
                continue
            img = torch.tensor(img).type(torch.FloatTensor)
            img = torch.permute(img, (2,0,1))
            img = torch.unsqueeze(img, 0)
            img = img/255
            pred = torch.argmax(model(img)).item()
            if pred == 1:
                x = 'Defective'
                defective_count += 1
                d['Inspection result'].append(x)
            else:
                x = 'Okay'
                okay_count += 1
                d['Inspection result'].append(x)

            if self.demonstrate:
                if x == 'Okay': self.insp_label.setStyleSheet('font-size: 20px; color: green')
                else: self.insp_label.setStyleSheet('font-size: 20px; color: red;')
                self.insp_label.setText(x)
                time.sleep(1)
            else:
                self.insp_label.setText('')
        elapsed_time = time.process_time() - t
        self.img_pane.setText(f"Finished Inspection!¥n¥nDefective: {defective_count}¥nOkay: {okay_count}¥n¥nDefect Rate: {defective_count/(defective_count+okay_count+1e-10):.2f}%¥nAvg. Throughput {elapsed_time/len(d['Image']):.2f} items per second")
        self.insp_label.setText('')
        df = pd.DataFrame(d)
        df.to_csv(os.path.join(self.path, 'results.csv'), index=False)
        print('Wrote results.csv')
        self.title.setText('Casting Product Inspector')

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Classifier.py – Contains the CNN architecture, acts as a utility file for main.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import cv2 as cv
import numpy as np
import os

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3),
            nn.ReLU(),
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3),
            nn.ReLU(),
            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3),
            nn.Dropout2d(p=0.2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features = 1024),
            nn.Conv2d(in_channels=1024, out_channels=128, kernel_size=1),
            nn.Flatten(),
            nn.Dropout(p=0.5),
            nn.LazyLinear(out_features=256),
            nn.Sigmoid(),
            nn.Linear(in_features=256, out_features=128),
            nn.Sigmoid(),
            nn.Linear(in_features = 128, out_features=64),
            nn.Sigmoid(),
            nn.Linear(in_features=64, out_features=2),
            nn.Softmax(dim=1)
        )
    def forward(self, x):
        return self.net(x)
```

All the other files can be found on github at:

https://github.com/KoshRai/CastingProductInspector