Sabancı Üniversitesi

Fall 2021
EE 417 Computer Vision
Term Project

Eylül Bektur
Nur Başak Özer

1. Importance of the Problem and Problem Definition

Anomaly detection (also known as *outlier detection*) is the process of finding data points that behave very differently from the expectation.[1] Such data points are called **anomalies** due to the fact that they show signs of significant deviation from the rest of the data, as if they were generated by a different mechanism. Even though in most cases, the data itself is created by the same generating mechanism, unusual behavior in the generating process might cause the creation of anomalies. For this reason, an anomaly most likely contains useful information about atypical characteristics of the mechanisms that affect the data generating process.[2]

Anomaly detection provides useful insights for many real-life applications. For example, abnormal patterns in data retrieved from medical devices such as MRI scans and ECG time-series, may indicate disease conditions. Moreover, a considerable amount of spatiotemporal data regarding weather patterns or climate changes, is accumulated through a variety of mechanisms such as satellites or remote sensing[2]. Anomalies in such data can yield crucial insights regarding human activities or environmental trends that may be the underlying causes of related natural phenomenons.

Anomaly detection has a very close relation to one of the main fields of computer vision, **feature detection**. Features can essentially be described as local, meaningful, and detectable parts of an arbitrary image.[3] This description clearly aligns with the characteristics of anomalies due to the following arguments:

- Anomalies constitute a neighborhood of some pixels in an image. Hence they are local.
- Furthermore, anomalies also correspond to aspects of objects in the real world because objects on the scene are projected onto the image plane.
- Finally, anomalies are computationally detectable with the assistance of a variety of different, state-of-the-art techniques such as supervised and unsupervised learning.

Precisely for this reason, we will explore some of these anomaly detection methodologies in this project, with the intention of demonstrating the strong connection between computer vision and machine learning. In addition, we will compare each and every one of the models built with respect to selected techniques in terms of accuracy and computational cost.

2. Problem Formulation and Solution Method

Our problem is binary image classification with a mapping algorithm. The algorithm can be described as follows.

Definition 1. Our anomaly detection algorithm by classifying surface cracks is a function $f(i) : X \rightarrow Y$ such that:

$$f(i) = label(i)$$

That correctly identifies the existence of an anomaly, by labeling it to zero if it exists or one if it is vice versa.

In this term project, the task of the trained model C is to learn an approximate mapping function. Our algorithm runs with three feature arrays corresponding to their label sets which are splitted from X.

Let training dataset $Y_{train} = \{0, 1\}^M = \{y_1, y_2, y_3, y_4 ..... y_M\}$ be a set of known labels that consists of a number that points if there is an anomaly or not; for our term project Y_{train} array will keep only two numbers, 0 and 1 respectively. The training set should be given as balanced with two labels to prevent underfitting.

These labels corresponds to features $X_{train} = \{x_1, x_2, x_3......, x_M\}$ that is used for training. To optimize this mapping in a way that it gives correct outcomes, validation dataset $X_{val} = \{x_1, x_2, x_3......, x_k\}$ as k being 1000, is used for increasing the accuracy of the algorithm.

Given with the test dataset, $X_{test} = \{x_1, x_2, x_3......, x_n\}$ while n being 2000, algorithm maps each $x \in X_{test}$ into discrete variable $y \in Y$ which belongs to the label array $Y_{test} = \{y_1, y_2, y_3, y_4 ..... y_n\}$.

There are two bottlenecks in our problem formulation. First one is how to create model C. Second one is how to optimize how well C() approximates f().

The model that is used in this project consists of neural networks. By the help of the Figure 2.1,

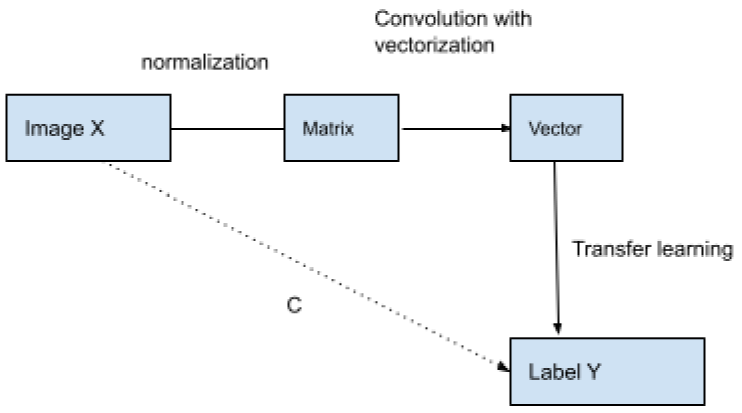


Figure 2.1: Commutative diagram for the process of the problem solution

The model C that will perform X→Y will be implemented with the following steps:

1) Normalization (Image Data preprocessing): Each image will be normalized. This will be done by keras' built-in class ImageDataGenerator. Each image will have 150 pixels as column and row size. Thus, output will be a matrix.
2) Prediction with VGG16 convolutional base: Each matrix will be linearized by passing data through VGG16, our convolutional base. Purpose of convolution is to extract the image's features (like texture of the given surface image). Then these features are vectorized.
3) Transfer learning: Outer layers will be added to VGG16. New layers will be fit into the convolutional base. This model will be used to map each vector into "anomaly" - "not-anomaly".

The model uses 20 layers along with the base model which consists of 16 layers. Several techniques have been applied as a classifier.
First two types of given classifier techniques are based on adding Softmax layers on the convolutional base. The 2 dimensional output which indicates the likelihood for the anomaly is processed through the sigmoid layer, which is equivalent to Softmax in binary classification. Built-in compile function is used to create a Python object that will build the CNN. Binary cross entropy is set as the loss function which suits our problem. Adam optimizer will be used as an optimizer with a learning rate 0.01.

Softmax layer implementation can be given as, where h be the activation of the penultimate layer nodes W is the weight connecting the penultimate layer to the softmax layer and a would be total input to the softmax layer[9].

$$a_i = \sum_k h_k W_{ki}$$

Since we have two classes for our problem, softmax layer will have 2 nodes denoted by p_i, which results in a discrete probability distribution.

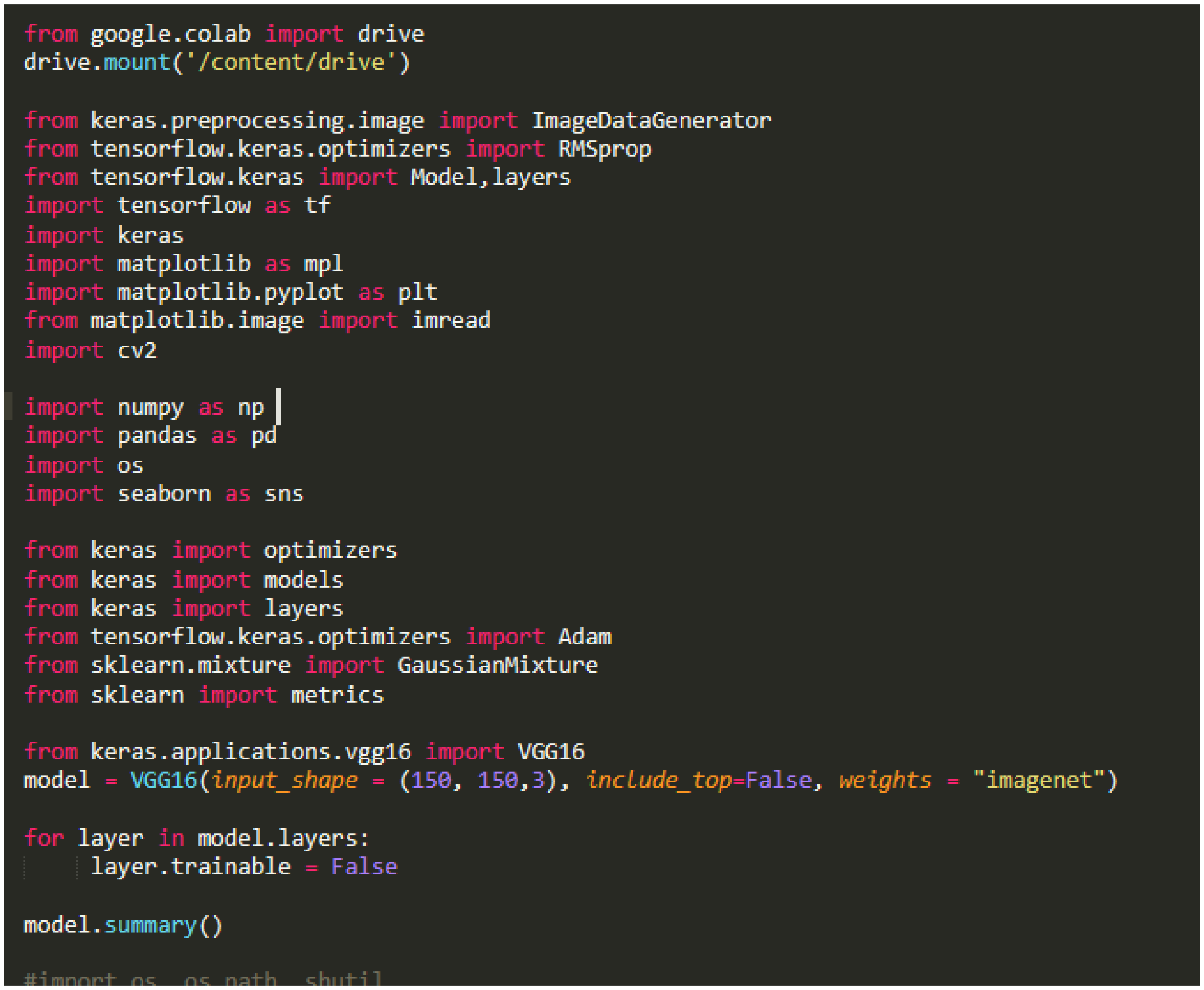
$$p_i = \frac{exp(a_i)}{\sum_j^2 exp(a_i)}$$

For the last classifier implementation, a linear support vector machine (SVM) is built. Apart from the activation function that is used in the first two implementations, the linear combination of two feature inputs is implemented for the training process which acts as the reduction in the dimensionality with the given linear function below[7].

$$f(ax_n, bx_m) = f(ax_n) + f(bx_m)$$

3. Implementation and Results

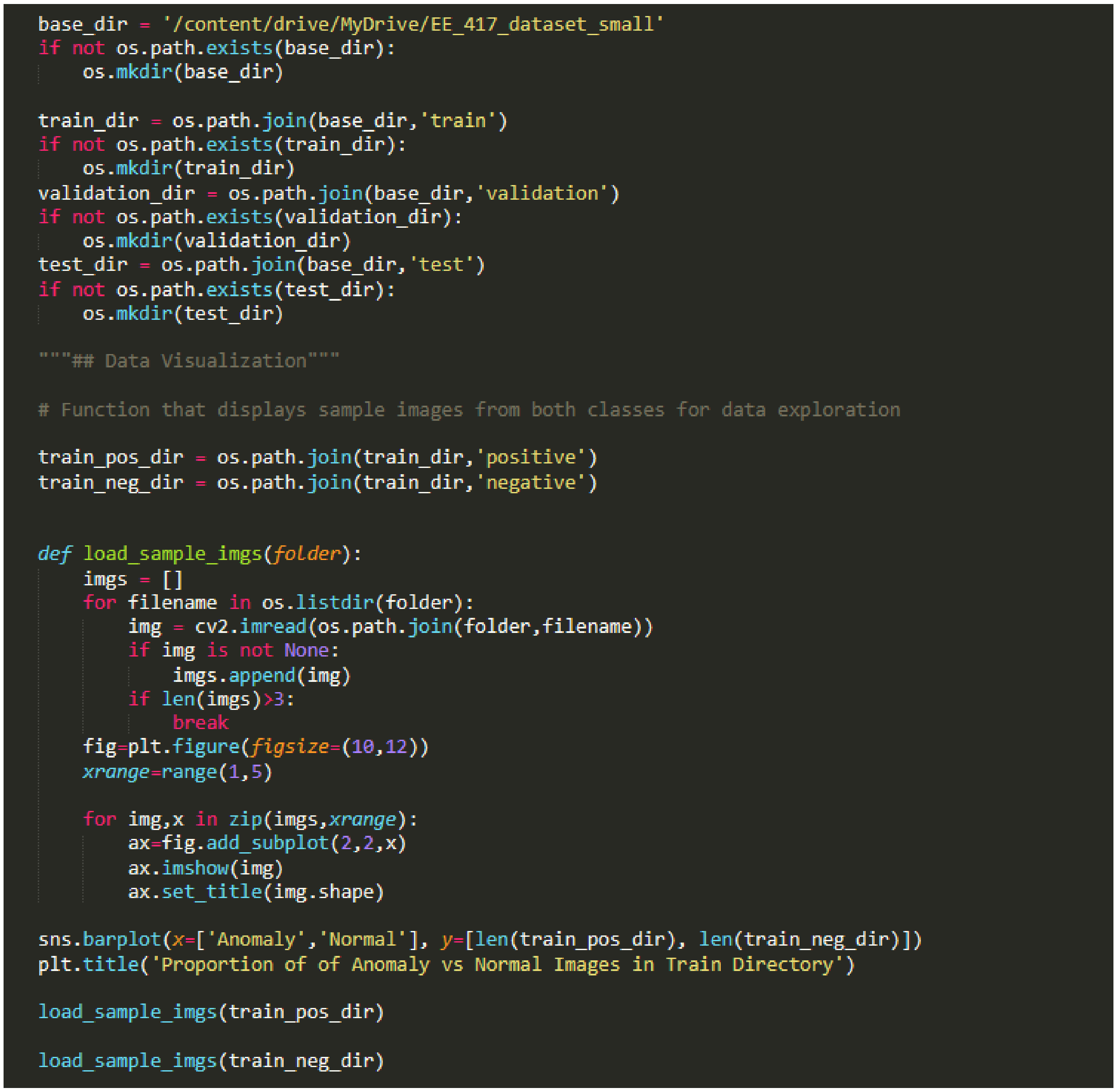
a) Importing libraries and convolutional base VGG16



```
from google.colab import drive
drive.mount('/content/drive')

from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras import Model,layers
import tensorflow as tf
import keras
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2

import numpy as np
import pandas as pd
import os
import seaborn as sns

from keras import optimizers
from keras import models
from keras import layers
from tensorflow.keras.optimizers import Adam
from sklearn.mixture import GaussianMixture
from sklearn import metrics

from keras.applications.vgg16 import VGG16
model = VGG16(input_shape = (150, 150,3), include_top=False, weights = "imagenet")

for layer in model.layers:
    layer.trainable = False

model.summary()

#import os, os.path, shutil
```

In our implementation, we had heavily utilized **Tensorflow** and one of its interfaces **Keras** for most of the machine learning operations that we had implemented. Tensorflow is an open source platform that provides a flexible ecosystem of tools, libraries and community resources for both novices and machine learning experts. Through Tensorflow, we had been able to access Keras, which we had used to be able to import deep learning models and their layers. Here, we had also defined our VGG16 model and set some of its relevant parameters with respect to the characteristics of the surface crack dataset. For instance, the input shape is set to be the same dimension as the dimensions of each and every image from the dataset in order to avoid shape mismatch error.

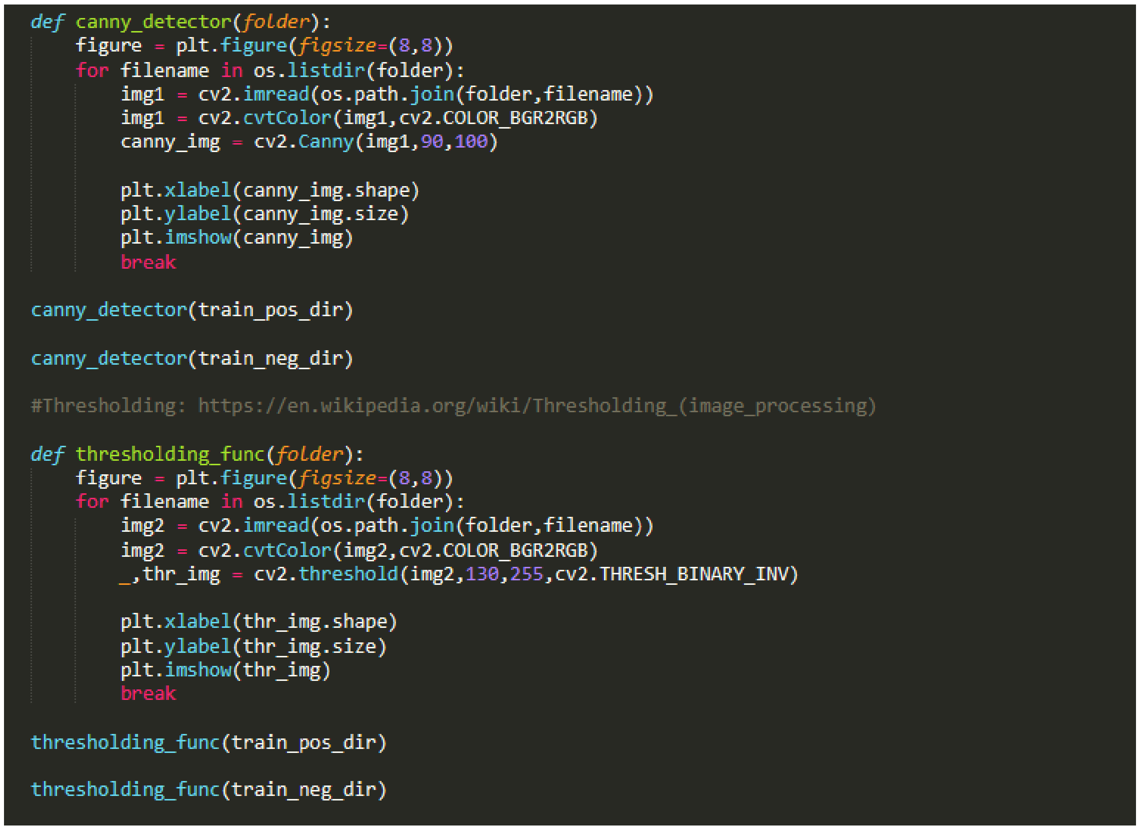
b) Dataset Preprocessing



```python
base_dir = '/content/drive/MyDrive/EE_417_dataset_small'
if not os.path.exists(base_dir):
    os.mkdir(base_dir)

train_dir = os.path.join(base_dir,'train')
if not os.path.exists(train_dir):
    os.mkdir(train_dir)
validation_dir = os.path.join(base_dir,'validation')
if not os.path.exists(validation_dir):
    os.mkdir(validation_dir)
test_dir = os.path.join(base_dir,'test')
if not os.path.exists(test_dir):
    os.mkdir(test_dir)

"""## Data Visualization"""

# Function that displays sample images from both classes for data exploration

train_pos_dir = os.path.join(train_dir,'positive')
train_neg_dir = os.path.join(train_dir,'negative')


def load_sample_imgs(folder):
    imgs = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        if img is not None:
            imgs.append(img)
        if len(imgs)>3:
            break
    fig=plt.figure(figsize=(10,12))
    xrange=range(1,5)

    for img,x in zip(imgs,xrange):
        ax=fig.add_subplot(2,2,x)
        ax.imshow(img)
        ax.set_title(img.shape)

sns.barplot(x=['Anomaly','Normal'], y=[len(train_pos_dir), len(train_neg_dir)])
plt.title('Proportion of of Anomaly vs Normal Images in Train Directory')

load_sample_imgs(train_pos_dir)

load_sample_imgs(train_neg_dir)
```

During this step of our implementation, we had divided the surface crack dataset into 3 directories: Training, validation and test. Particularly, we wanted to make sure that 70% of the images would be used for the training process and 20% of the images would be used when testing the models. We knew that in order to get higher test accuracies for our models, we needed to train them as much as possible. That is why we had to reserve a significant portion of our dataset for the newly created training directory. Moreover, we also had loaded some sample images from the training directory in order to assess if the variation between images belonging to 2 seperate classes is visible to the human eye.

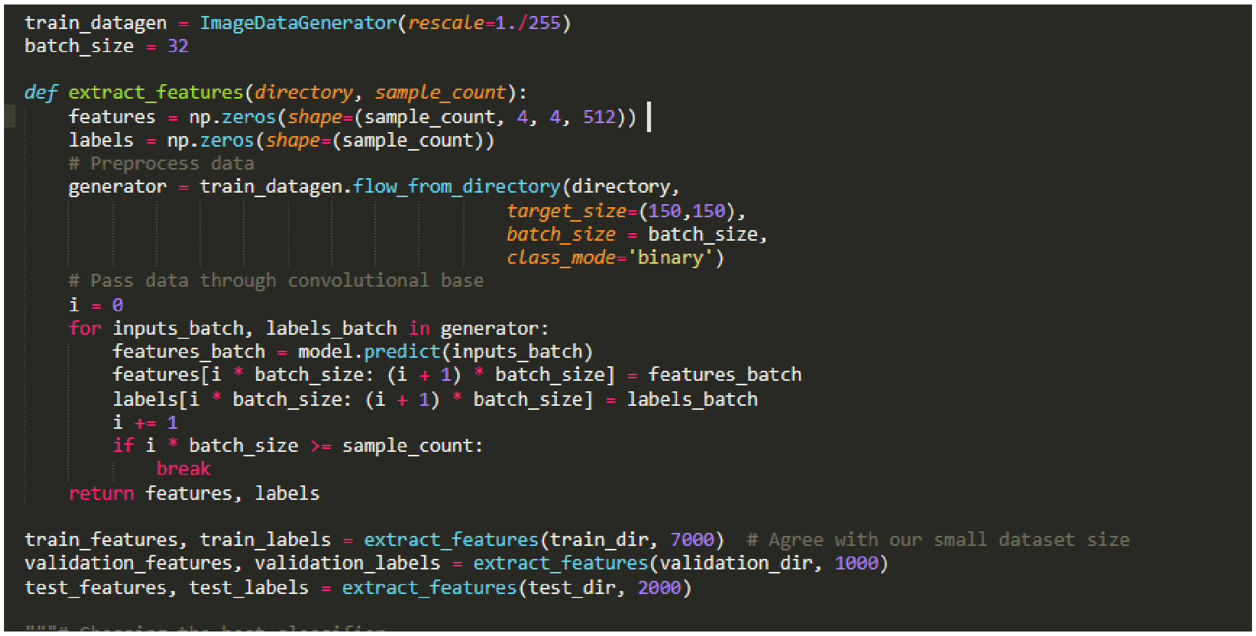
c) Image Visualization



```python
def canny_detector(folder):
    figure = plt.figure(figsize=(8,8))
    for filename in os.listdir(folder):
        img1 = cv2.imread(os.path.join(folder,filename))
        img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
        canny_img = cv2.Canny(img1,90,100)

        plt.xlabel(canny_img.shape)
        plt.ylabel(canny_img.size)
        plt.imshow(canny_img)
        break

canny_detector(train_pos_dir)

canny_detector(train_neg_dir)

#Thresholding: https://en.wikipedia.org/wiki/Thresholding_(image_processing)

def thresholding_func(folder):
    figure = plt.figure(figsize=(8,8))
    for filename in os.listdir(folder):
        img2 = cv2.imread(os.path.join(folder,filename))
        img2 = cv2.cvtColor(img2,cv2.COLOR_BGR2RGB)
        _,thr_img = cv2.threshold(img2,130,255,cv2.THRESH_BINARY_INV)

        plt.xlabel(thr_img.shape)
        plt.ylabel(thr_img.size)
        plt.imshow(thr_img)
        break

thresholding_func(train_pos_dir)

thresholding_func(train_neg_dir)
```

Before moving onto the training process, we had also intended to apply some of the classical computer vision techniques that are heavily utilized in feature detection, over some sample images. Throughout this course, we had inspected a selection of edge and corner detectors and compared their performance on various images (e.g. buildings, geometrical shapes, portraits etc.). Although we knew that the characteristics of surface cracks are far different from the aspects of edges and corners, we still wanted to test if we could see the differences between anomalous and non-anomalous images much better after applying some of these techniques. Results of these operations can be found in the discussion section along with more information regarding each one the used methodologies.

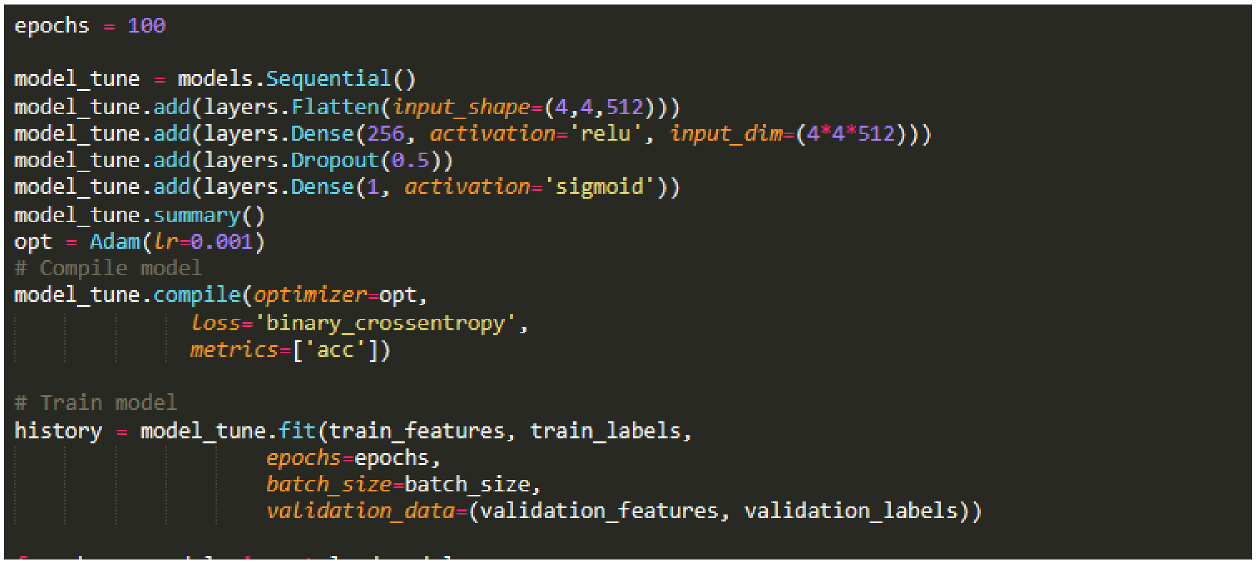
d) Data Preprocessing (Normalization)



```python
train_datagen = ImageDataGenerator(rescale=1./255)
batch_size = 32

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    # Preprocess data
    generator = train_datagen.flow_from_directory(directory,
                                                  target_size=(150,150),
                                                  batch_size = batch_size,
                                                  class_mode='binary')
    # Pass data through convolutional base
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = model.predict(inputs_batch)
        features[i * batch_size: (i + 1) * batch_size] = features_batch
        labels[i * batch_size: (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 7000)  # Agree with our small dataset size
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 2000)
```

We have implemented three classifiers which are built with different approaches. As an output of these classifiers, anomaly existence will be detected. We compiled our classifier with the Adam optimization algorithm which is the extension of the stochastic gradient descent that is performed for the VGG base. For all three of the classifiers, we have set the learning rate 0.01. All classifiers were trained with a hundred epochs.

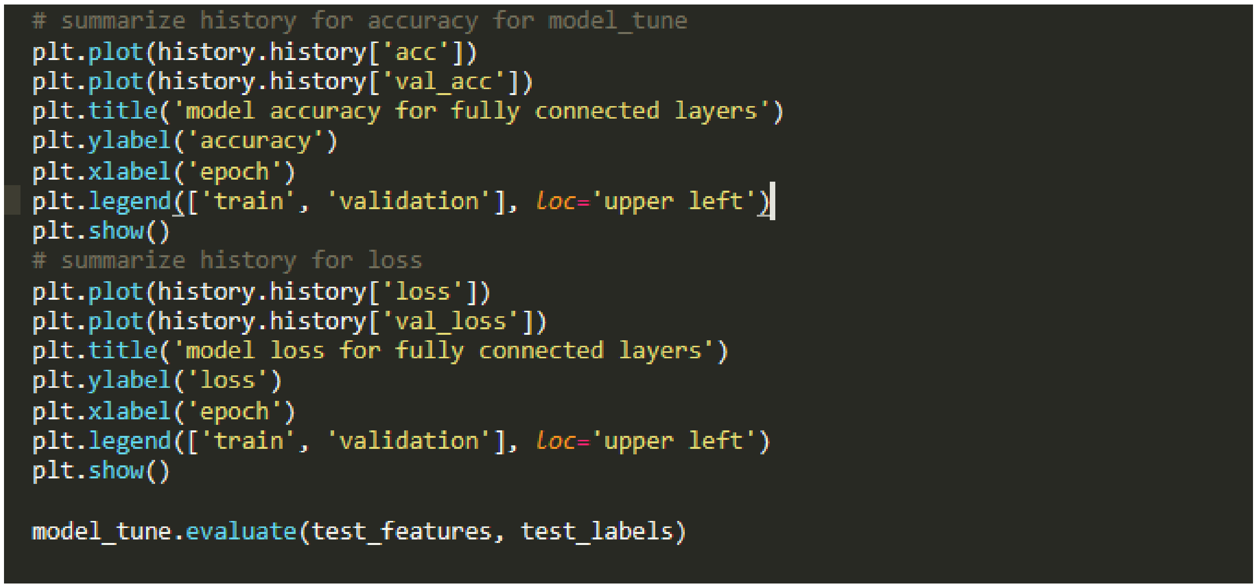
e) Transfer Learning with classifier built by fully connected layers



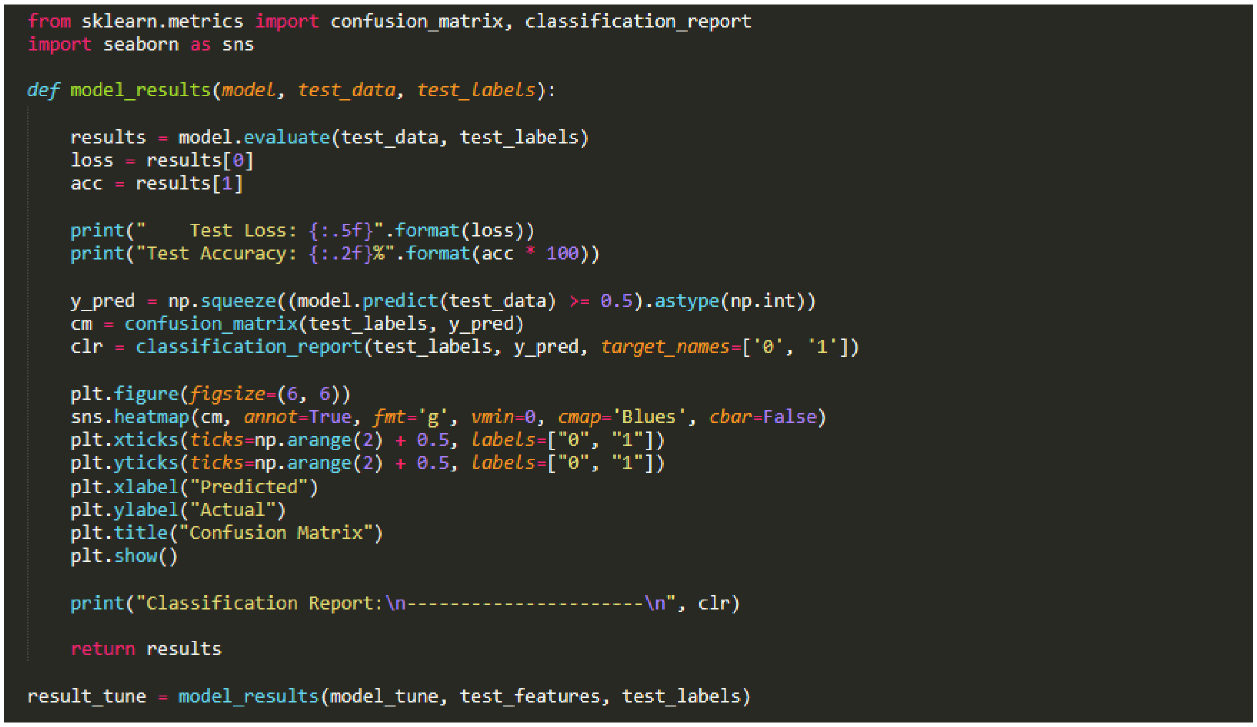
```
epochs = 100

model_tune = models.Sequential()
model_tune.add(layers.Flatten(input_shape=(4,4,512)))
model_tune.add(layers.Dense(256, activation='relu', input_dim=(4*4*512)))
model_tune.add(layers.Dropout(0.5))
model_tune.add(layers.Dense(1, activation='sigmoid'))
model_tune.summary()
opt = Adam(lr=0.001)
# Compile model
model_tune.compile(optimizer=opt,
                   loss='binary_crossentropy',
                   metrics=['acc'])

# Train model
history = model_tune.fit(train_features, train_labels,
                         epochs=epochs,
                         batch_size=batch_size,
                         validation_data=(validation_features, validation_labels))
```

Technique 1, Accuracy: 99.7% Loss: 5%

Visualization for evaluation of performance (will be shown in the discussion section):

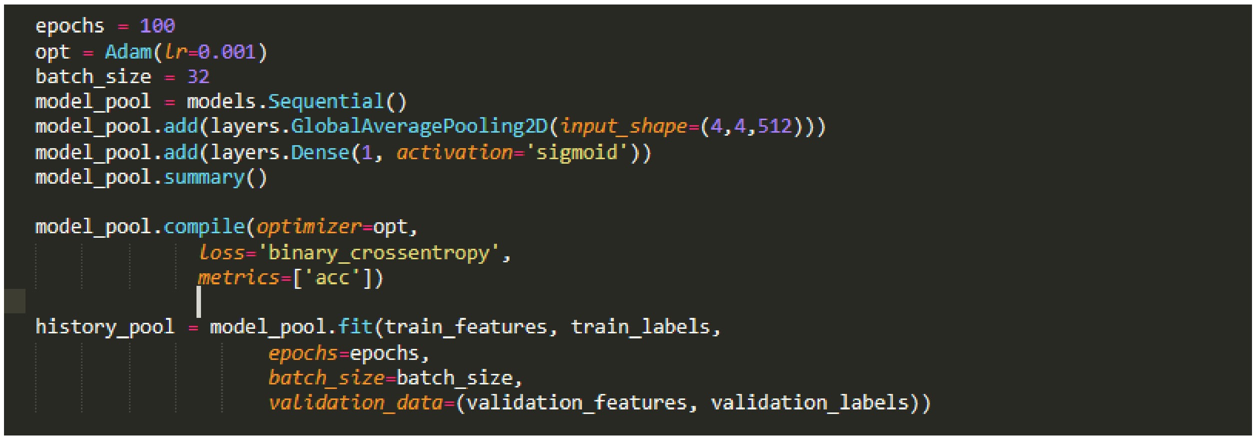


```
# summarize history for accuracy for model_tune
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy for fully connected layers')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss for fully connected layers')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

model_tune.evaluate(test_features, test_labels)
```

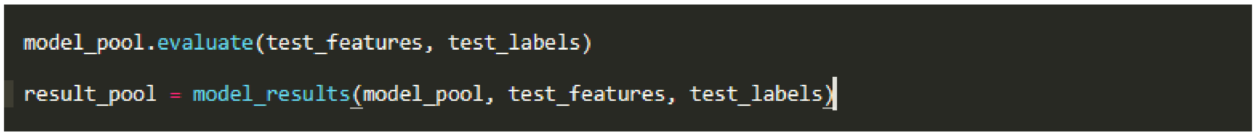


```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

def model_results(model, test_data, test_labels):

    results = model.evaluate(test_data, test_labels)
    loss = results[0]
    acc = results[1]

    print("    Test Loss: {:.5f}".format(loss))
    print("Test Accuracy: {:.2f}%".format(acc * 100))

    y_pred = np.squeeze((model.predict(test_data) >= 0.5).astype(np.int))
    cm = confusion_matrix(test_labels, y_pred)
    clr = classification_report(test_labels, y_pred, target_names=['0', '1'])

    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
    plt.xticks(ticks=np.arange(2) + 0.5, labels=["0", "1"])
    plt.yticks(ticks=np.arange(2) + 0.5, labels=["0", "1"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    print("Classification Report:\n----------------------\n", clr)

    return results

result_tune = model_results(model_tune, test_features, test_labels)
```

f) Transfer Learning with classifier built by global average pooling



```python
epochs = 100
opt = Adam(lr=0.001)
batch_size = 32
model_pool = models.Sequential()
model_pool.add(layers.GlobalAveragePooling2D(input_shape=(4,4,512)))
model_pool.add(layers.Dense(1, activation='sigmoid'))
model_pool.summary()

model_pool.compile(optimizer=opt,
                   loss='binary_crossentropy',
                   metrics=['acc'])

history_pool = model_pool.fit(train_features, train_labels,
                              epochs=epochs,
                              batch_size=batch_size,
                              validation_data=(validation_features, validation_labels))
```



```python
model_pool.evaluate(test_features, test_labels)

result_pool = model_results(model_pool, test_features, test_labels)
```

Technique 2, Accuracy: 99.6% Loss: 1.67%

Visualization for evaluation of performance (will be shown in the discussion section):

```python
# summarize history for accuracy for model_pool
plt.plot(history_pool.history['acc'])
plt.plot(history_pool.history['val_acc'])
plt.title('model accuracy for global average pooling')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history_pool.history['loss'])
plt.plot(history_pool.history['val_loss'])
plt.title('model loss for global average pooling')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

g) Transfer Learning with classifier built by linear support vector machine

```python
import sklearn
from sklearn.svm import LinearSVC

svm_features = np.concatenate((train_features, validation_features))
svm_labels = np.concatenate((train_labels, validation_labels))
X_train, y_train = svm_features.reshape(8000,4*4*512), svm_labels

from sklearn.decomposition import PCA

pca = PCA(n_components = 2).fit(X_train)
pca_2d = pca.transform(X_train)
svmClassifier = LinearSVC().fit(pca_2d, y_train)
```

Technique 3, Accuracy: 96.0%

Visualization for evaluation of performance (will be shown in the discussion section):

```python
import pylab as pl
c1 = 0
c2 = 0
for i in range(0, pca_2d.shape[0]):
  if svm_labels[i] == 0:
    c1 = pl.scatter(pca_2d[i,0], pca_2d[i,1], c='r', marker = '+')
  elif svm_labels[i] == 1:
    c2 = pl.scatter(pca_2d[i,0], pca_2d[i,1], c='g', marker = '*')
x_min, x_max = pca_2d[:,0].min() -1, pca_2d[:,0].max() + 1
y_min, y_max = pca_2d[:,1].min() -1, pca_2d[:,1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .01), np.arange(y_min, y_max, .01))
Z = svmClassifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
pl.contour(xx,yy,Z)
pl.legend([c1, c2], ['Anomaly', 'Not Anomaly'],)
pl.title('Linear SVM Decision Surface')
pl.show()
```

4. Discussion

Transfer learning is a timesaving way by leveraging previous learnings. During our project, VGG is used as a pretrained model trained on a large dataset called ImageNet. Feature extraction is done with the convolutional base of VGG16 and the classifier is used for predicting if there is an anomaly of the given image or not. The first layers of the VGG16 that are closer to the inputs refer to general features whereas higher layers, whereas the layers close to the output refer to specific features. Our classifier aimed to label anomaly and normal images.

Dataset Selection

Crack surface detection dataset in Kaggle contains 40000 images, seperated to halves for positive and negative images for the existence of an anomaly. However, due to computational constraints, the dataset is reduced to 10000. 80% of the images are used for training of the model, while 20% and 10% are used for testing and validation consecutively.

Our dataset is large and similar to the ImageNet dataset that is used for VGG base. According to Shu and Li[4], images' edges, textures, spaces and other feature information can be extracted rich and diverse image spatial feature information with ImageNet being the source data with its more than 1 million images. Although the crack detection dataset contains 8000 images for training the model overcomes the worry of the overfitting, transfer learning becomes a favorable option since ImageNet dataset provides spatial feature information that saves huge training effort. As a result, implying transfer learning, training classifiers and the top layers of the convolutional base would be enough.

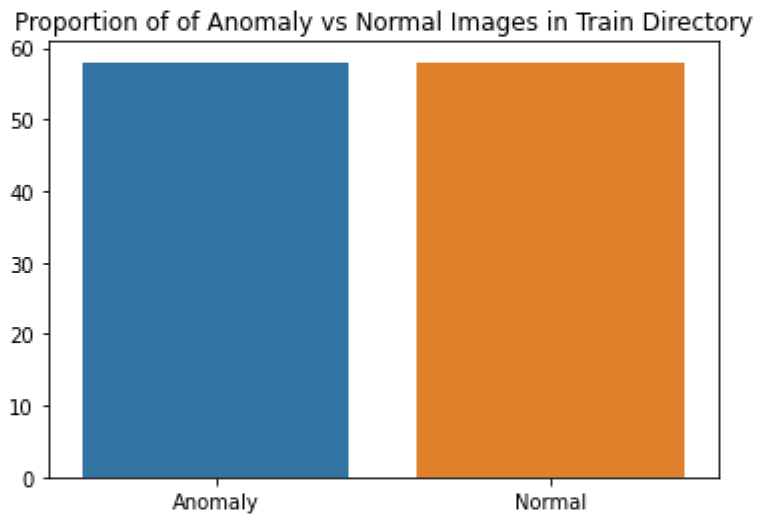


Figure 4.1: Proportion of anomalous images to the normal images inside the training images directory

To become familiar with the data, let us observe some sample images extracted from the training image directory of the dataset before preprocessing. In Figure 4.2, we can see a set of anomalous images as they are depicting various surfaces with cracks. The cracks on the surface are easily noticeable and there is a clear distinction between the intensity levels of pixels of the anomalous surface and normal surface. Thus, it could be a significant feature to help the classification of the image set. On the other hand, we see some sample images of normal surfaces with no visible cracks in Figure 4.3. These images are considered as non-anomalous due to the fact that the intensity values of the pixels are more or less around the same value i.e. it appears there are no cracks on the surface.

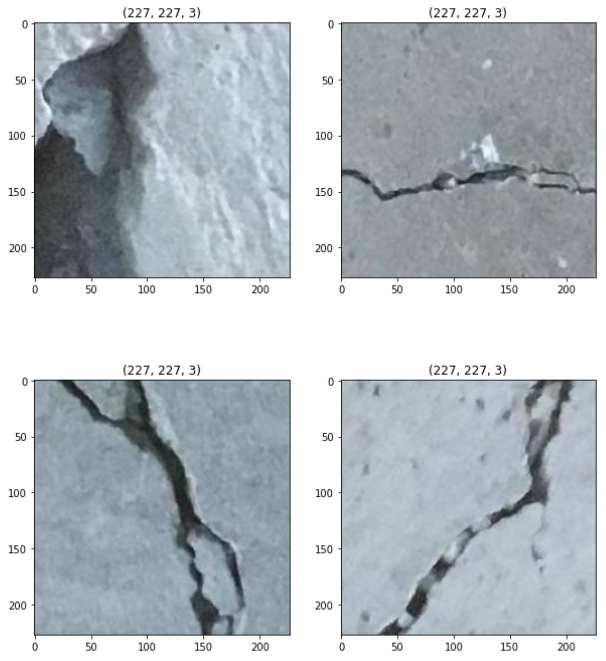


Figure 4.2: Surfaces with the Cracks (with anomaly)

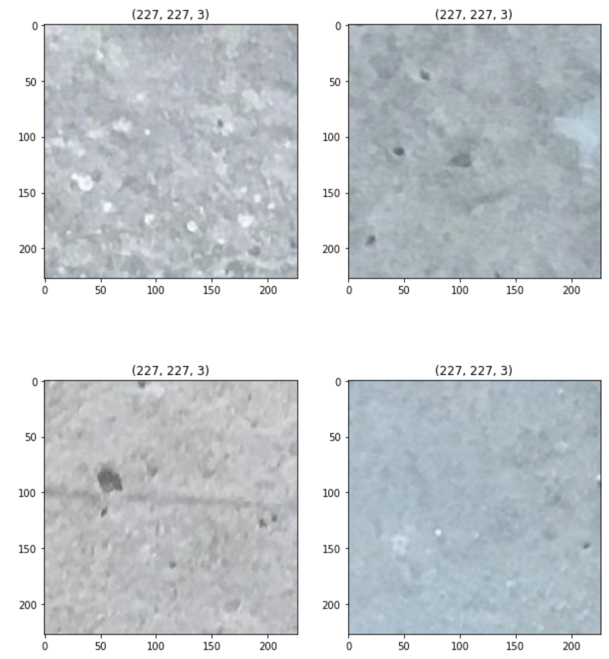


Figure 4.3: Surfaces without the Cracks (without anomaly)

Let us also apply some classical feature detection techniques over a portion of our dataset. Throughout this course, we have learned different kinds of edge detectors which highlighted edge-like features in the images. One of those detectors was called **Canny edge detector**, which provided optimally and most of the time, accurately detected images. Here we basically applied OpenCV's Canny operator named *cv2.canny*[5] over 2 sample images taken from the training dataset. One of these images was anomalous (Figure 4.4) and the one was normal (Figure 4.5). When comparing these 2 sample images, it can be arguably deduced that the anomalous sample contains much more features that appear to form line-like structures (as they bear strong resemblance to the shape of a *crack*) compared to the normal looking sample. Consequently, this could be an important indication for the classification models when labeling each data point.

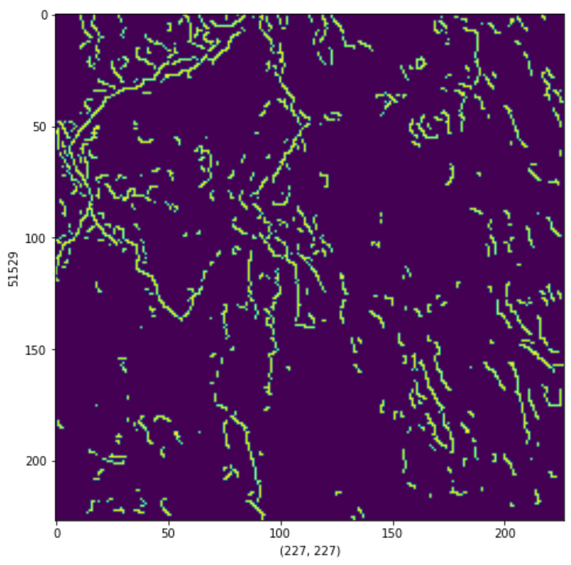


Figure 4.4: Canny Edge Detected Anomalous Image

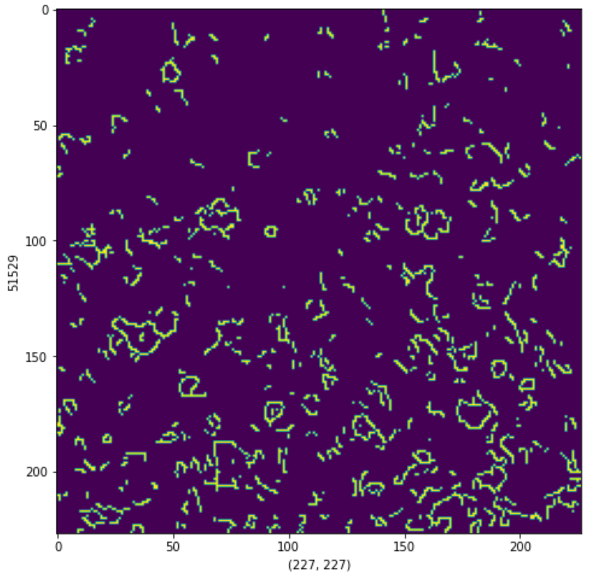


Figure 4.5: Canny Edge Detected Non-Anomalous Image

Alternatively, we can also apply another prominent image processing method called **thresholding**. A thresholding operation essentially chooses some pixels that may be the part of objects of interest as *foreground pixels* and the rest of the pixels as *background pixels*.[6] To put it another way, it isolates separates in the images by transforming them into binary images. Here, we perform a simple thresholding operation using the type **cv.THRESH_BINARY** with OpenCV's built-in *cv2.threshold* function[7], once again over 2 sample images taken from the training directory. The resulting images obtained after performing the thresholding operation are Figure 4.6 and Figure 4.7, where the former image was an anomalous image while the latter image depicted a normal surface. It can be clearly seen from the images that the anomalous sample contains much more pixels that have contrasting intensity values compared to the non-anomalous sample. A significant contrast between the intensity values
of the pixels is something that we had suspected when we were looking at the sets of sample images from the dataset earlier and it might prove to be an important indicator for the classification algorithms when detecting anomalies later on.

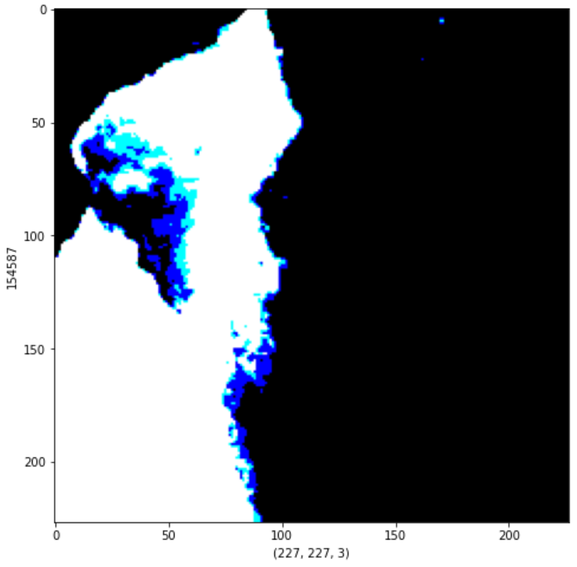


Figure 4.6: Thresholding applied on Anomalous Image

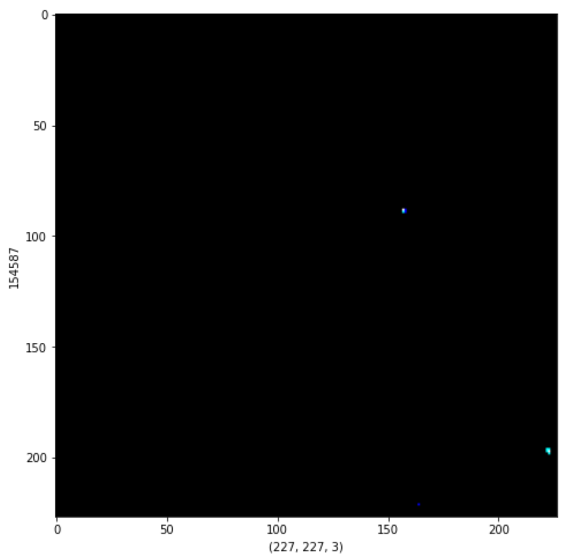


Figure 4.7: Thresholding applied on Non-Anomalous Image

Data Preprocessing including Feature Extraction from VGG16

Normalization and vectorization of each image is performed here. Instead of manually loading image data and obtaining the pixel arrays along with class integers, Keras' ImageDataGenerator class is used. It also allows loading the dataset in small batches, which was beneficial for our project since we have a relatively large dataset compared to our computer's physical constraints. Obtained images in the small batches are used for training of the model. Later, we used flow_from_directory() function, which is a built-in function in the ImageDataGenerator class, to load images of our dataset in small batches. Target size for all images are set to 150x150 to prevent non-uniform training inputs for the model. Class mode is chosen as binary since our problem is a binary classification problem.
Batch size is chosen as 32 to eliminate unnecessary computations of the gradient descent over the large datasets. Our model converges with the given epochs, which hints to correct selection of the batch size. Lastly, batch size should be chosen to be in favor of GPU because it is the power of two.

Each image batch is sent to VGG16 base for feature extraction and stored in the features array of size (size of the dataset, 4,4,512).

Outputs are the features of the images and their labels in an array. Same function is called three times, for training, test and validation set. Each array contains images according to the size of the dataset given. For instance, the shape of the training batch after function is (7000, 4, 4, 512) .

Transfer Learning

1) Classifier Implementation

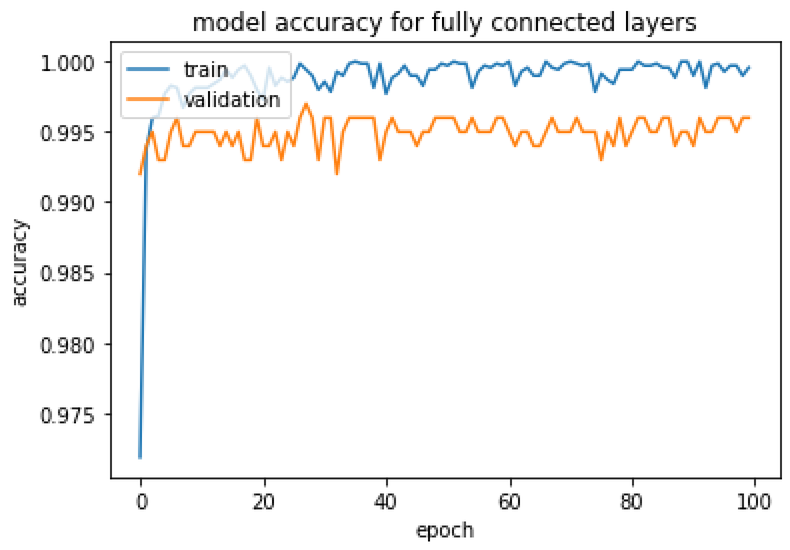
The model is trained with the built-in fit function. Loss function that is calculated determines the tuning of the model parameters and model approximates with the new choice of weights. This illustrates a continuous function. Later, accuracies and losses are recorded for each epoch to see the performance of these two models.
For the last classifier implementation, a linear support vector machine (SVM) is built. All accuracies are compared to pick the most suited classifier to our problem.

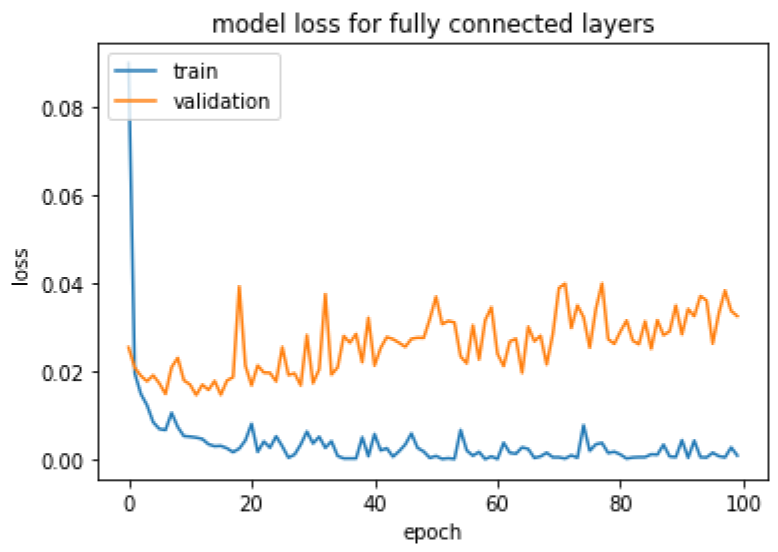
a) Built by fully-connected layers:

Some layers are added on the VGG16 convolutional base for specializing features to make the distinction of the anomaly in the given surface images.
Flatten, dense, dropout and dense layers are used consecutively. Flatten is used to reduce the multidimensional output and pass it to the dense layer. Dense layer is used for “relating features” by relu activation function. Dropout layer randomly sets some of the dimensions zero in the input vector. It is used to eliminate too much association between features and used to tackle overfitting. Lastly, a dense layer is implemented with sigmoid activation function.

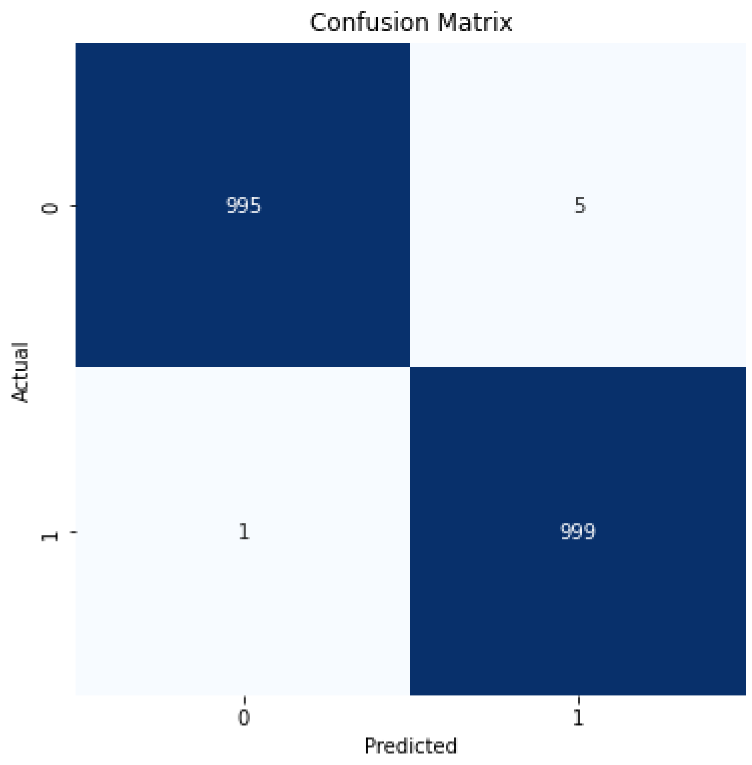
Relu activation function is used in the dense layer since relu helps eliminate negative values in the input to the upcoming nodes and is computationally cheaper than the sigmoid function[8].

Sigmoid activation function is used at the last dense layer in which we need to predict the probability of anomaly as our output, since the value of probability is between 0 and 1.





Test accuracy is around 99.7% and loss is 5%, which is encouraging given the size of the dataset. The model does not overfit. There's a small gap between the training and the validation curves.
Since we already used dropout, we should increase the size of the dataset to improve the results.
The loss function is not decreasing when the model stops training. Probably, it is not possible to improve the model by increasing the number of epochs.

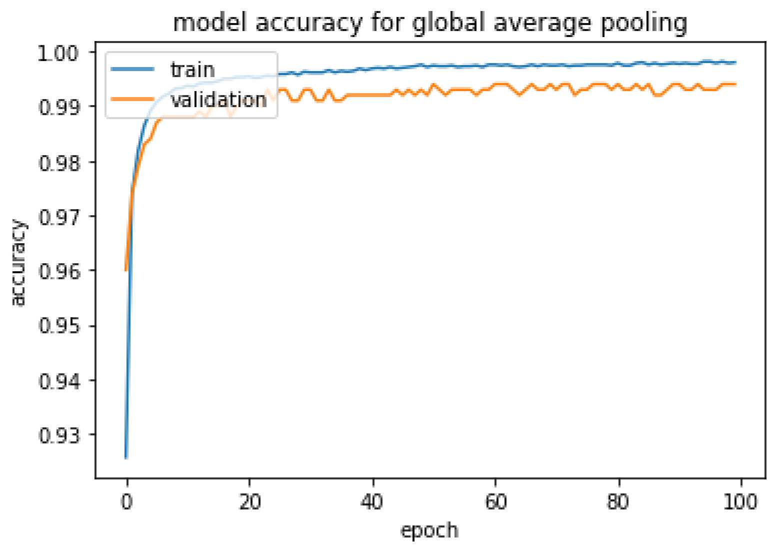


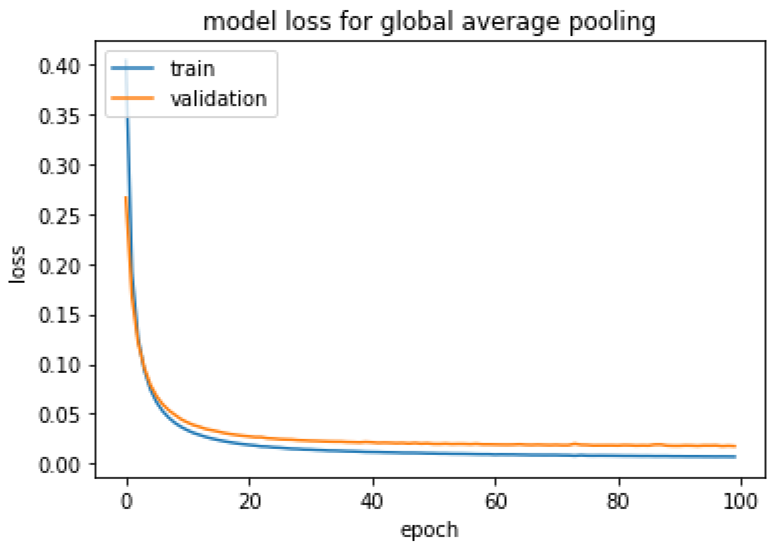
Our model is able to identify the normal images and images with anomalies with 99.5% and 99.9% accuracy consecutively.

b) Built by global average pooling

This time instead of implementing the fully connected layers, pooling operation is designed by keras's built-in GlobalAveragePooling2D function. This layer takes the average of the feature maps given as an output from the VGG16 base.

Output vector from the pooling layer is used as an input in the softmax layer, which was also the case in the fully connected layer. This implementation takes less initiatives from us as parameters which decreases the risk of overfitting.





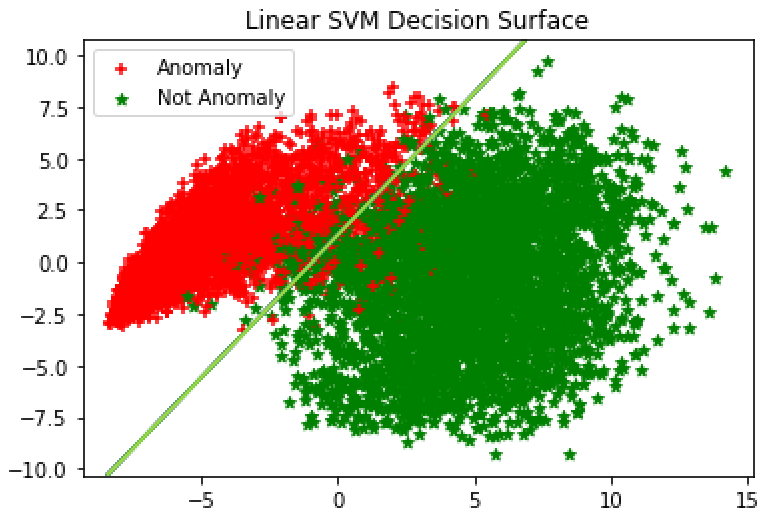
Test accuracy is 99.6% and loss is 1.67% similar to the one resulting from the fully-connected layers solution in terms of accuracy.

This model also proposes a good solution since train and validation loss decrease and stabilize during the same epochs.

c) Built by linear support vector machine

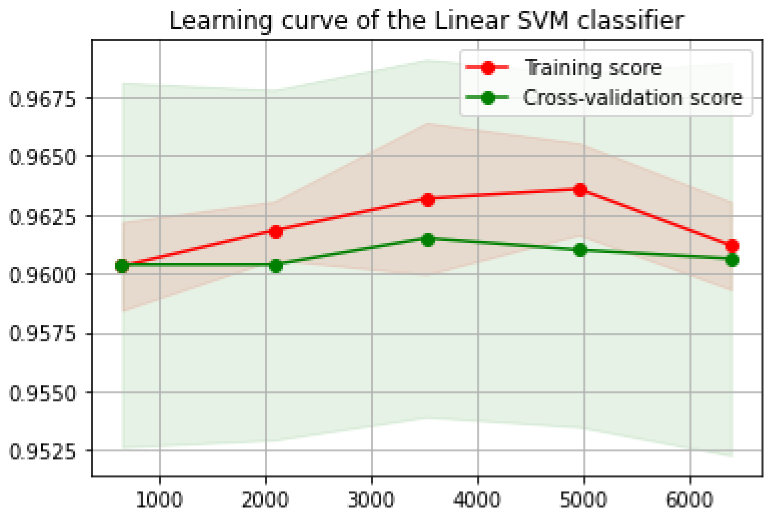
For support vector machine implementation, feature arrays for both validation and training are combined along with the train and validation labels.

Sklearn's built-in PCA function is used for several reasons. Main reason is to obtain how much variance can be attributed to each image to be defined as either with or without anomaly. As a result, it is used as a feature extractor. Moreover, PCA is also used to reduce the dimension to two from four. Later, the SVM classifier is trained with the output of the PCA.



Test accuracy is 96% for this linear SVM solution.

As it is seen from the learning curve, accuracy fluctuates between 96% to 96.5% and shows dramatic increase or decrease. As the train line is always a higher validation line, which also supports our model illustrates a good fit like the previous models.



2) Evaluation on Classifiers

According to our three approaches, we found accuracies of 99.7, 99.6, and 96.0 for fully connected layers, global average pooling and support vector machines respectively. Our problem shows that Softmax layer implementation outperforms SVM. Although it raises the suspicions of overfitting with very high accuracies, we obtained good results due to our success in data preprocessing and our data is very suitable for our problem definition. Our problem is deterministic; which focuses on if there is an anomaly or not. Our dataset contains classes that are similar in such a way that they are all white surfaces. However, they are different in terms of the existence of the cracks on the surfaces.

However, although our data helped us to classify images, our batches where the neural network learns showed success which may be seen from accuracies.
In support of this, we also proposed a balanced dataset where we have an equal number of surfaces with and without cracks. This results in equal share in prediction of "YES" or "NO". Therefore, there is a high base rate in our model's training as well.

Execution times for model fits take 6 minutes 22 seconds, 56 seconds and 6 seconds for fully connected layers, global average pooling and support vector machines consecutively. Although fully connected layers give highest accuracy among them, there is a significant tradeoff between execution times. SVM was very fast to train, almost ten times faster than our faster implementation on sigmoid layers, global average pooling.

5. Appendix

Full code of the implementation can be accessed via the following Google Collab notebook link:
https://colab.research.google.com/drive/1DmgAIm-eYj7SRIAJsrOIa6dyAQm8DabX?usp=sharing

6. References

[1] Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han and Micheline Kamber and Jian Pei, pg. 543-544
[2] Outlier Analysis (2nd Edition) by Charu Aggarwal, pg. 21-22
[3] W4-5 Image Features & Edge Detection, Lecture Slides provided by Mustafa Ünel
[4] Surface Defect Detection and Recognition Method for Multi-Scale Commutator Based on Deep Transfer Learning by Yufeng Shu and Bin Li
[5] OpenCV: Canny Edge Detection
https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
[6] Computer Vision by Linda G. Shapiro and George C. Stockman, pg. 97
[7] OpenCV: Image Thresholding
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
[8] Daily outflow prediction by multi layer perceptron with logistic sigmoid and tangent sigmoid activation functions, Water resources management 24 (2010), no. 11, 2673–2688. Mehdi Rezaeian Zadeh, Seifollah Amin, Davar Khalili, and Vijay P Singh
[9] Deep Learning using Linear Support Vector Machines, Yichuan Tang
[10] SVM vs Neural Network, Gabriele De Luca
https://www.baeldung.com/cs/svm-vs-neural-network