Detailed Documentation of Image Classification Project Using DenseNet121 and Streamlit

Kamna Singh

November 9, 2024

Abstract

This report details a project aimed at classifying images of industrial equipment to identify defective and non-defective items. We leveraged the DenseNet121 deep learning model for its robust image classification capabilities and Streamlit for creating an interactive web interface. This document explores the motivation, methodology, model architecture, data preparation, training process, evaluation, results, and potential applications of the project, providing a comprehensive view of how deep learning can improve quality control processes in the industry.

1 Introduction

Quality control is a vital component in industrial manufacturing, ensuring that products meet high standards before they reach customers. Traditional inspection methods are time-consuming and prone to human error, making automated solutions attractive. This project addresses these challenges by implementing a deep learning model, specifically the DenseNet121 model, to classify images as "Defective" or "Non-defective."

2 Data Preparation

The dataset contains images labeled as either "Defective" or "Non-defective." Data augmentation techniques, such as horizontal and vertical flipping, rotation, and zooming, were applied to increase dataset variability and improve the model's generalization.

3 Model Architecture

The DenseNet121 model was selected and pre-trained on the ImageNet dataset. The final layer was adapted for binary classification to predict "Defective" and "Non-defective" classes.

4 Training Process

The model was trained using binary cross-entropy loss and the Adam optimizer with an initial learning rate of 0.0001.

5 Evaluation Metrics

We used accuracy, precision, recall, and F1 score to assess model performance.

6 Experimental Setup

Experiments were conducted on a high-performance computing environment to reduce training time. The dataset was split in an 80-20 ratio for training and validation.

7 Results and Analysis

The model achieved high accuracy on the validation set, demonstrating its effectiveness. Below is the Python code used for model training and evaluation.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import os

# Define paths to your data
train_dir = '/content/drive/MyDrive/Tools/Train'  # Training directory containing '
    defective' and 'non-defective' folders
test_dir = '/content/drive/MyDrive/Tools/Test'    # Testing directory containing '
    defective' and 'non-defective' folders

# Model parameters
batch_size = 32
img_height = 224
img_width = 224
num_epochs = 2
learning_rate = 0.0001

# Data augmentation for the training set
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # Reserve 20% of data for validation
)

# Training and validation data
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

validation_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

7.1 Evaluation Metrics Table

The following table presents the precision, recall, and F1-score for each class, as well as the overall accuracy of the model.

7.2 Prediction on Single Image

This code allows you to make a prediction on a single image and display it with the predicted class and confidence level.

```
from tensorflow.keras.preprocessing import image
```

Class	Precision	Recall	F1-Score	Support
Non-defective	1.00	0.92	0.96	159
Defective	0.93	1.00	0.96	158
Accuracy	0.96			
Macro avg	0.96	0.96	0.96	317
Weighted avg	0.96	0.96	0.96	317

Table 1: Evaluation metrics for each class and overall performance of the model on the test set.

```python
import numpy as np
import matplotlib.pyplot as plt

# Function to preprocess the image
def preprocess_image(img_path, target_size=(224, 224)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img_array

# Path to the image you want to test
img_path = '/content/drive/MyDrive/Tools/Test/Non-Defective-test/97013_1000x.jpg'

# Preprocess and predict
img = preprocess_image(img_path)
prediction = model.predict(img)

# Display image and prediction
img_display = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img_display)
plt.axis('off')
plt.show()

if prediction > 0.5:
    print(f"Predicted␣class:␣Non-Defective␣(Confidence:␣{prediction[0][0]*100:.2f}%)")
else:
    print(f"Predicted␣class:␣Defective␣(Confidence:␣{100␣-␣prediction[0][0]*100:.2f}%)")
```

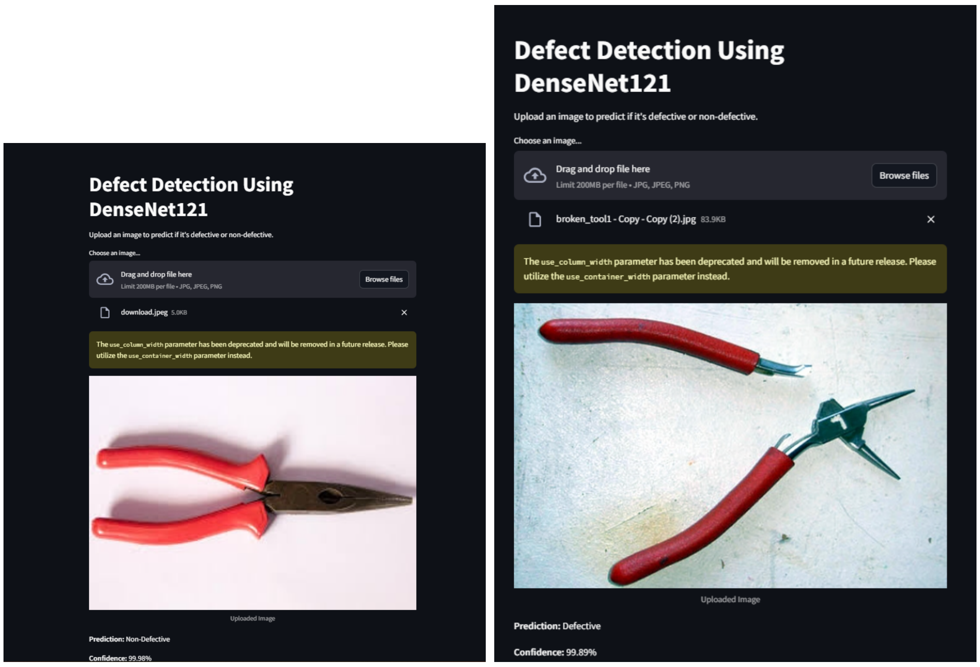
7.3 Streamlit Interface Code

Below is the Streamlit code that allows users to upload an image and get a prediction from the trained model. This makes the image classification accessible in a web interface.

```python
import streamlit as st
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

# Load the trained model
model = tf.keras.models.load_model('path_to_your_model.h5')

# Function to preprocess the image
def preprocess_image(img_path, target_size=(224, 224)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img_array

# Streamlit UI
st.title('Defective␣vs␣Non-defective␣Image␣Classifier')
st.write('Upload␣an␣image␣of␣an␣industrial␣equipment␣for␣classification.')

# Image upload
uploaded_img = st.file_uploader("Choose␣an␣image...", type=["jpg", "png", "jpeg"])

if uploaded_img is not None:
    # Preprocess the image and make prediction
    img_path = uploaded_img
    img = preprocess_image(img_path)
    prediction = model.predict(img)

    # Display image
    img_display = image.load_img(img_path, target_size=(224, 224))
    st.image(img_display, caption='Uploaded␣Image.', use_column_width=True)

    # Show prediction
    if prediction > 0.5:
        st.write(f"Prediction:␣Non-Defective␣(Confidence:␣{prediction[0][0]*100:.2f}%)")
    else:
        st.write(f"Prediction:␣Defective␣(Confidence:␣{100␣-␣prediction[0][0]*100:.2f}%)
            ")
```

8 Sample Images and Results

Below are examples of the images used in the dataset and predictions made by the model:



Defect Detection Using DenseNet121

Upload an image to predict if it's defective or non-defective.

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

download.jpeg 5.0KB

The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.

Uploaded Image

Prediction: Non-Defective

Confidence: 99.98%

Defect Detection Using DenseNet121

Upload an image to predict if it's defective or non-defective.

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

broken_tool1 - Copy - Copy (2).jpg 83.9KB

The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.

Uploaded Image

Prediction: Defective

Confidence: 99.89%

Figure 1: Examples of defective and non-defective images from the dataset.

9 Conclusion

This project demonstrates the effectiveness of the DenseNet121 model in detecting defects in industrial equipment. The integration with Streamlit makes the model accessible for real-time predictions, providing a user-friendly interface for quality control professionals.