Teknoloji Fakültesi

MRM3048.1 –Mekatronik Sistem Tasarımı

Ödev Raporu

Omar Tarabin 170221992

Görüntü İşleme ile Saç Levha Üzerindeki Yüzey Hataları Tespiti

Bu raporda, saç levha resimler üzerinde görüntü işleme yöntemleri uygulanarak, levhaların üzerindeki bilmem ne sırasında gelebilecek olan bazı hataların otomatik olarak tespit etmesi üzerinde çalışma yapılacaktır. Bu çalışmada önce veri toplama yapıldıktan sonra, Python programlama dili ve TensorFlow makine öğrenimi çerçevesi kullanılarak bir CNN modeli oluşturulacak. Ardından bu modeli toplanan verilerle eğittikten sonra nihai modelin doğrulaması yapılacak ve sonuçlar analiz edilecektir.

1. Sac metalde yüzey kusurlarinin tespitinin önemi

Sac metaldeki yüzey kusurlarını tespit etmenin önemi, ürün kalitesini sağlamak, maliyetleri düşürmek, müşteri memnuniyetini artırmak, güvenliği sağlamak ve düzenlemelere uymaktır. Görüntü İşleme ve özellikle CNN mimarileri gibi araçları kullanan bilgisayar teknolojileri, modern üretim süreçlerinde bu kusurları tespit etmek ve ele almak için verimli ve etkili yöntemler sunar. Bu çalışmanın önemi aşağıdaki maddelerle özetlenebilir:

- Kalite Güvencesi: Kusurların tespit edilmesi, yüksek kaliteli malzemelerin kullanılmasını sağlayarak ürün kalitesini ve güvenilirliğini artırır.
- Maliyet Azaltma: Kusurların erken tespit edilmesi hurda ve yeniden işleme maliyetlerini azaltarak üretim verimliliğini optimize eder.
- Müşteri Memnuniyeti: Hatasız ürünler müşteri memnuniyetini ve sadakatini artırır.
- Güvenlik: Kusurların tespit edilmesi, özellikle otomotiv ve havacılık gibi kritik uygulamalarda kazaları önler.
- Uyumluluk ve Yönetmelikler: Hatasız ürünlerin yasal ve güvenlik gereksinimlerini karşılamasını sağlamak, cezalardan kaçınmak ve pazara erişimi sağlamak.

2. Bazı yüzey kusurları ve örnek görselleri

Bu çalışmada NEU Metal Yüzey Kusurları veri seti kullanılmıştır. Bu veri seti, sıcak haddelenmiş çelik şeritlerin altı tipik yüzey kusurunu içermektedir: haddelenmiş ölçek, yamalar, çentiklenme, çukurlu yüzey, inklüzyon ve çizikler. Veritabanı, her altı tipik yüzey kusurundan 300 fotoğraf olmak üzere toplamda 1800 fotoğraf içerir.

3. Yapay Sinir Ağları ve CNN

Yapay sinir ağları birbirine bağlı yapay nöron denen zımbırtıların birleşiminden oluşan bir yapıdır. Bu ağlar önce toplanan etiketlenmiş verileri kullanarak eğitilir. Eğitilmesi sırasında içindeki

parametreler (ağırlıklar) her gelen veri ile Gradiant Descent ve Backpropogation algoritmalarına bağlı olarak hata eğrisini en aza indirecek şekilde güncellenir. Bu şekilde yapay sinir ağları özellikle direkt olarak modellemesi mümkün olmayan çok karmaşık yapıları modellemek için çok başarılı bir yöntemdir. Eğitim sonunda bu aşamaya ayrılmış özel bir veri grubu ile modelin doğrulaması yapılır ve eğer sonuçlar uygun görülürse model tespit için kullanılabilir. CNN (Evrişimli Sinir Ağı), yaygın olarak görsel görüntüleri analiz etmek için uygulanan bir yapay sinir ağı sınıfıdır. Bir girdi olarak bir görüntü aldıktan sonra onu çeşitli katmanlardan geçirerek görüntüdeki farklı özelliklere değer verir (ağırlık verir) ve sınıflandırma yapar

4. Uygulama

Raporun bu kısmında Python ve TensorFlow kullanarak modelin oluşturulmasını, eğitilmesini ve sonuçlarını analizini içeren kodun her parçası detaylı olarak anlatılacaktır.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
from tensorflow.keras import models, layers

import cv2
import numpy as np
from matplotlib import pyplot as plt

tf.test.gpu_device_name()
```

Bu kısım, görüntü işleme ve görselleştirme için bazı ek kütüphanelerin yanı sıra derin öğrenme için Keras ile TensorFlow'u kullanmak için ortamı ayarlar. `'tf.test.gpu_device_name()'` fonksiyonu TensorFlow'un kullanabileceği bir GPU olup olmadığını kontrol etmek için kullanılmıştır.

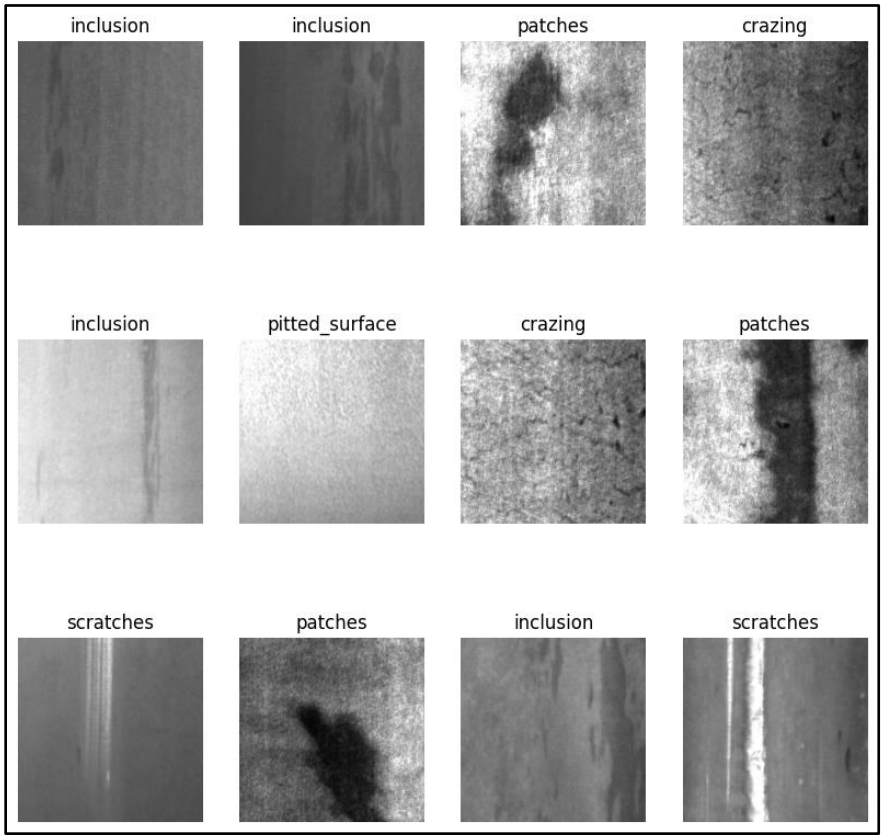
```
data_dir = r'data'
IMAGE_SIZE = 200
BATCH_SIZE = 32
CHANNELS = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory=data_dir,
    image_size= (IMAGE_SIZE, IMAGE_SIZE),
    batch_size= BATCH_SIZE
)

class_names = dataset.class_names
```

And this does this and that. The following snippet will loop through and show some of the images in the dataset along with their labels.

```
plt.figure(figsize=(10, 10))
for image_batch, label_batch in dataset.take(1):
    for i in range(12):
```

```
    ax = plt.subplot(3, 4, i + 1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    plt.title(class_names[label_batch[i]])
    plt.axis("off")
```

Veri içerisinde bulunan bazı görseller aşağıdaki gibidir:




```
train_size = int(len(dataset)*.8)
val_size = int(len(dataset)*.1)
test_size = int(len(dataset)*.1)

train_ds = dataset.take(train_size)
val_ds = dataset.skip(train_size).take(val_size)
test_ds = dataset.skip(train_size+val_size).take(test_size)
# print(len(train_ds) + len(val_ds) + len(test_ds))

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])

train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

This does this and that, don’t forget to write about data augmentation.

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = len(class_names)

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

This is the fun bit. Include a picture that shows the structure of the neural network so it looks cool.

```
training_history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=15,
```

```
    callbacks = [tf.keras.callbacks.TensorBoard(log_dir='logs')]
)
```

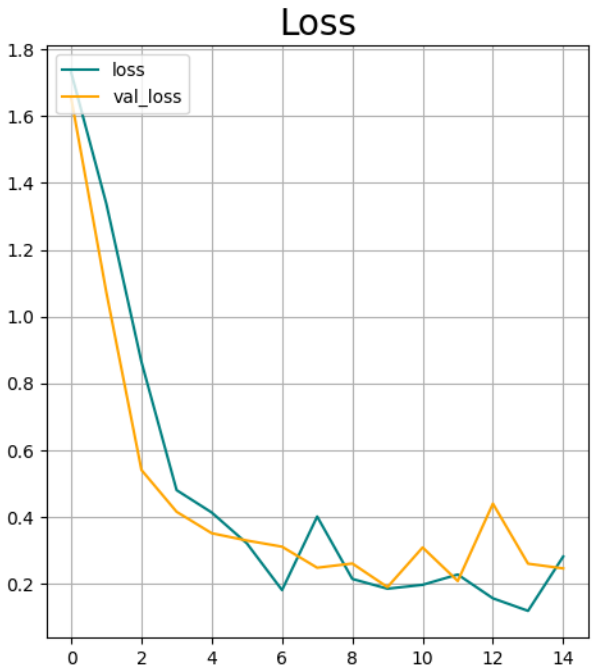
`'model.fit()'` fonkisyonu modeli eğitme sürecini başlatır. Burada 15 döngü boyunda, 45 adet batch'te bulunan tüm verileri işleyerek model kendi parametrelerini loss'u minimum tutacak şekilde günceller ve validation veri grubu ile doğrulama yapar. Model eğitilirken aşağıdaki çıktıyı sürekli verir ve görüldüğü her epoch'ta modelin kaybı yavaşça 0'ya yaklaşmakta, aynı zamanda doğruluk oranı da 1'e yaklaşmaktadır.

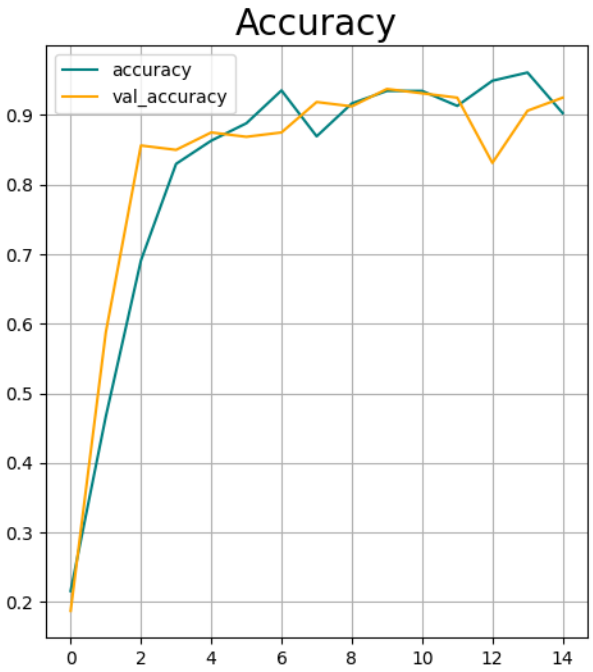
```
Epoch 1/15
45/45 [==============================] - 10s 108ms/step - loss: 1.7303 - accuracy: 0.2160 - val_loss: 1.6540 - val_accuracy: 0.1875
Epoch 2/15
45/45 [==============================] - 4s 87ms/step - loss: 1.3380 - accuracy: 0.4667 - val_loss: 1.0730 - val_accuracy: 0.5875
Epoch 3/15
45/45 [==============================] - 4s 86ms/step - loss: 0.8644 - accuracy: 0.6910 - val_loss: 0.5410 - val_accuracy: 0.8562
…
Epoch 13/15
45/45 [==============================] - 4s 87ms/step - loss: 0.1570 - accuracy: 0.9493 - val_loss: 0.4402 - val_accuracy: 0.8313
Epoch 14/15
45/45 [==============================] - 4s 87ms/step - loss: 0.1194 - accuracy: 0.9611 - val_loss: 0.2605 - val_accuracy: 0.9062
Epoch 15/15
45/45 [==============================] - 4s 87ms/step - loss: 0.2819 - accuracy: 0.9028 - val_loss: 0.2464 - val_accuracy: 0.9250
```

Something something something.

5. Sonuçlar ve Doğrulama

```
fig = plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(training_history.history['loss'], color='teal', label='loss')
plt.plot(training_history.history['val_loss'], color='orange', label='val_loss')
plt.grid(True)
plt.title('Loss', fontsize=20)
plt.legend(loc="upper left")

plt.subplot(1, 2, 2)
plt.plot(training_history.history['accuracy'], color='teal', label='accuracy')
plt.plot(training_history.history['val_accuracy'], color='orange', label='val_accuracy')
plt.grid(True)
plt.title('Accuracy', fontsize=20)
plt.legend(loc="upper left")

plt.show()
```





Şekilde görüldüğü gibi döngüler arttıkça, yani model verileri öğrenmeye devam ettikçe kayıplar 0'a düşerken, doğruluk oranı 1'e doğru yükselerek en sonda 0,92 civarında durmuştur. Bu iki grafik şu şu şu yüzden önemli ve performans metrik olarak kullanılır.

Doğrulama için basit bir bilmem ne yapılmış sonra daha detaylı bir inceleme için bilmem ne fonkisyonu yazılmış.

```
scores = model.evaluate(test_ds)
```

5/5 [==============================] - 1s 18ms/step - loss: 0.2112 - accuracy: 0.9312

Görüldüğü gibi modelimiz test veri grubu üzerinde 0,9312 doğrulama oranı ile tahmin yapabilmektedir. Bu tahmin doğruluğunu arttırmak icin farkli bir model mimarisi denenebilir veya daha fazla veri daha uzun döngüler boyunce eğitilebilir. Son olarak aşağıdaki kısım test verisi üzerinde tek tek fotoğraflar üzerinde model tahmini yapılarak fotoğrafın kendisi gerçek sonuç ve tahmin ile birlikte gösterilmiştir.

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
```

```
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```