هوش مصنوعی

تمرین چهارم

مائده اسماعیل‌زاده

۸۱۰۶۰۲۱۶۱

استاد: دکتر شریعت‌پناهی

دانشکده مهندسی مکانیک

پردیس دانشکده‌های فنی دانشگاه تهران

فهرست مطالب

دادگان UCI HAR

آماده‌سازی داده‌ها

در یادگیری ماشین، نرمال‌سازی داده‌ها نقش بسیار مهمی در عملکرد مدل‌ها دارد. روش‌های رایج نرمال‌سازی شامل موارد زیر هستند. یکی از این روش‌ها استانداردسازی(Z-score Normalization) می‌باشد. استانداردسازی مناسب برای داده‌هایی می‌باشد که توزیع نرمال دارند. همچنین برای الگوریتم‌های حساس به مقیاس مانند MLP یا SVM مناسب است. اما اگر داده توزیع نرمال نداشته باشد، ممکن است عملکرد بهینه نباشد. همچنین نسبت به داده‌های پرت نیز حساس است. فرمول این روش بصورت زیر می‌باشد:

$$z = \frac{x - \mu}{\sigma}$$

روش دوم Min-Max Normalization می‌باشد که داده‌ها را بین [۰, ۱] مقیاس می‌کند. مناسب برای الگوریتم‌هایی مانند شبکه‌های عصبی با فعال‌سازsigmoid . این روش نیز به داده‌های پرت بسیار حساس است و در صورت اضافه شدن داده‌های جدید، نیاز به بازنرمال‌سازی دارد .فرمول این روش بصورت زیر می‌باشد:

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

روش سوم MaxAbs Scaling می‌باشد که مناسب برای داده‌هایی با مقادیر مثبت و منفی می‌باشد. این روش ساختار داده‌ها را حفظ می‌کند. اما اطلاعات آماری مانند میانگین و واریانس نادیده گرفته می‌شود. فرمول این روش بصورت زیر می‌باشد:

$$x_{\text{scaled}} = \frac{x}{|x_{\max}|}$$

روش چهارم Robust Scaling می‌باشد که در برابر داده‌های پرت مقاوم بوده و برای داده‌های غیر نرمال مناسب است. اما ممکن است برای برخی الگوریتم‌ها بهینه نباشد. فرمول این روش بصورت زیر می‌باشد:

$$x_{\text{scaled}} = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

در این تمرین، از روش استانداردسازی (StandardScaler) استفاده شده است که داده‌ها را طوری نرمال می‌کند که میانگین صفر و انحراف معیار یک داشته باشند. این روش به‌ویژه برای مدل‌هایی مانند MLP بسیار مؤثر است، زیرا باعث می‌شود گرادیان‌ها در طی آموزش پایدارتر باشند در ادامه ۱۵٪ داده‌ها برای آزمون و ۸۵٪ برای آموزش در نظر گرفته شده‌اند و با استفاده از train_test_splitتقسیم شده‌اند.

در این بخش از کد زیر استفاده شده است.

```python
from google.colab import files
import zipfile, os, pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

uploaded = files.upload()
zip = "UCI HAR Dataset.zip"
extract = "UCI_HAR_Dataset"
with zipfile.ZipFile(zip, 'r') as zip_ref:
    zip_ref.extractall(extract)

train_path = os.path.join(extract, "UCI HAR Dataset", "train")
test_path  = os.path.join(extract, "UCI HAR Dataset", "test")

X_train = pd.read_csv(os.path.join(train_path, "X_train.txt"), sep='\s+', header=None)
y_train = pd.read_csv(os.path.join(train_path, "y_train.txt"), header=None)
X_test  = pd.read_csv(os.path.join(test_path,  "X_test.txt"),  sep='\s+', header=None)
y_test  = pd.read_csv(os.path.join(test_path,  "y_test.txt"),  header=None)

# Taghsime dadeha be amuzesho azmun
X_all = pd.concat([X_train, X_test], axis=0).reset_index(drop=True)
y_all = pd.concat([y_train, y_test], axis=0).reset_index(drop=True)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_all)

X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(
    X_scaled, y_all.values.ravel(),test_size=0.15, random_state=42, stratify=y_all)
```

```python
print("Abaade amuzesh:", X_train_final.shape, "– Abaade azmun:", X_test_final.shape)
```

```
Choose Files  UCI HAR Dataset.zip
• UCI HAR Dataset.zip(application/x-zip-compressed) - 60999314 bytes, last modified: 5/22/2023 - 100% done
Saving UCI HAR Dataset.zip to UCI HAR Dataset.zip
Abaade amuzesh: (8754, 561) – Abaade azmun: (1545, 561)
```

باتوجه به نتیجه می‌توان گفت تعداد کل نمونه‌ها ۱۰,۲۹۹، تعداد ویژگی‌ها ۵۶۱، تعداد نمونه‌های آموزش۸,۷۵۴ و تعداد نمونه‌های آزمون ۱,۵۴۵ می‌باشد.

طراحی شبکه MLP

در این بخش، شبکه‌ی MLP با ساختار روبرو طراحی شده است. لایه ورودی بصورت نوشته شده در کد زیر، لایه پنهان اول بصورت Dense(128) با تابع فعال‌سازی ReLU، لایه Dropout بصورت Dropout(0,5) ، لایه پنهان دوم بصورت Dense(64) با تابع فعال‌سازی ReLU ، لایه Dropout بصورت Dropout(0,5) و لایه خروجی بصورت Dense(6) با تابع فعال‌سازی Softmax می‌باشد.

همانطور که مشاهده می‌شود برای لایه‌های پنهان از تابع فعالسازی ReLU استفاده شده است. دلایل انتخاب این تابع فعالسازی بدلیل محاسبه ساده و بسیار سریع، جلوگیری از مشکل ناپدید شدن گرادیان در شبکه‌های عمیق و کارایی بالا در یادگیری ویژگی‌های غیرخطی در داده‌های پیچیده مانند سیگنال‌های حسگر می‌باشد.

همچنین برای لایه خروجی از تابع Softmax استفاده شده است. از این تابع در مسائل طبقه‌بندی چندکلاسه استفاده می‌شود. در واقع خروجی را به برداری از احتمال‌ها تبدیل می‌کند که مجموع آن‌ها برابر با ۱ است. از آنجایی که هدف ما پیش‌بینی یکی از ۶ کلاس فعالیت انسانی است، تابع Softmax مناسب‌ترین گزینه برای لایه خروجی است.

همچنین مطابق خواسته صورت سوال از بهینه ساز Adam(learning_rate=0.001) استفاده شده است.

ساختار ارائه‌شده برای MLP ترکیبی از سادگی و قدرت یادگیری مناسب برای داده‌های برداری است. استفاده از توابع ReLU در لایه‌های پنهان باعث تسریع در آموزش و افزایش ظرفیت غیرخطی مدل شده است، و اضافه کردن لایه‌های Dropout احتمال بیش‌برازش را کاهش می‌دهد. تابع Softmax نیز برای خروجی مناسب‌ترین گزینه برای طبقه‌بندی شش کلاسه است. کد مربوط به این بخش بصورت زیر می‌باشد:

```
[2] # MLP
    import tensorflow as tf
    from tensorflow.keras import Sequential, Input
    from tensorflow.keras.layers import Dense, Dropout
    from tensorflow.keras.optimizers import Adam

    n = 6
    # one-hot encoding
    y_train_ohe = tf.keras.utils.to_categorical(y_train_final - 1, n)
    y_test_ohe = tf.keras.utils.to_categorical(y_test_final - 1, n)


    model = Sequential([Input(shape=(561,)),Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(n, activation='softmax'),])

    # Adam ba laerning rate=0.001
    optimizer = Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()

    # Amuzesh
    history = model.fit(X_train_final, y_train_ohe,validation_split=0.1,
        epochs=50,batch_size=32)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	71,936
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 6)	390

Total params: 80,582 (314.77 KB)
Trainable params: 80,582 (314.77 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 11s 22ms/step - accuracy: 0.5640 - loss: 1.2098 - val_accuracy: 0.9361 - val_loss: 0.2354
Epoch 2/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8590 - loss: 0.3633 - val_accuracy: 0.9441 - val_loss: 0.1446
Epoch 3/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9015 - loss: 0.2603 - val_accuracy: 0.9600 - val_loss: 0.1043
Epoch 4/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9242 - loss: 0.2086 - val_accuracy: 0.9509 - val_loss: 0.1122
Epoch 5/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9381 - loss: 0.1827 - val_accuracy: 0.9600 - val_loss: 0.1002
Epoch 6/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9450 - loss: 0.1612 - val_accuracy: 0.9658 - val_loss: 0.0879
Epoch 7/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9507 - loss: 0.1378 - val_accuracy: 0.9635 - val_loss: 0.0907
Epoch 8/50
```

```
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9552 - loss: 0.1271 - val_accuracy: 0.9635 - val_loss: 0.0864
Epoch 9/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9528 - loss: 0.1368 - val_accuracy: 0.9692 - val_loss: 0.0805
Epoch 10/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9600 - loss: 0.1024 - val_accuracy: 0.9555 - val_loss: 0.0939
Epoch 11/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9555 - loss: 0.1165 - val_accuracy: 0.9623 - val_loss: 0.0883
Epoch 12/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9629 - loss: 0.0929 - val_accuracy: 0.9703 - val_loss: 0.0699
Epoch 13/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9663 - loss: 0.0858 - val_accuracy: 0.9715 - val_loss: 0.0714
Epoch 14/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9601 - loss: 0.1017 - val_accuracy: 0.9669 - val_loss: 0.0676
Epoch 15/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9637 - loss: 0.0952 - val_accuracy: 0.9715 - val_loss: 0.0649
Epoch 16/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9704 - loss: 0.0797 - val_accuracy: 0.9715 - val_loss: 0.0628
Epoch 17/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9724 - loss: 0.0753 - val_accuracy: 0.9680 - val_loss: 0.0761
Epoch 18/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9678 - loss: 0.0885 - val_accuracy: 0.9749 - val_loss: 0.0540
Epoch 19/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9711 - loss: 0.0827 - val_accuracy: 0.9726 - val_loss: 0.0676
Epoch 20/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9713 - loss: 0.0732 - val_accuracy: 0.9680 - val_loss: 0.0694
Epoch 21/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9680 - loss: 0.0866 - val_accuracy: 0.9737 - val_loss: 0.0525
Epoch 22/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9730 - loss: 0.0764 - val_accuracy: 0.9726 - val_loss: 0.0571
Epoch 23/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9775 - loss: 0.0662 - val_accuracy: 0.9703 - val_loss: 0.0570
Epoch 24/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9733 - loss: 0.0814 - val_accuracy: 0.9669 - val_loss: 0.0640
Epoch 25/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9738 - loss: 0.0713 - val_accuracy: 0.9715 - val_loss: 0.0615
Epoch 26/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9722 - loss: 0.0775 - val_accuracy: 0.9703 - val_loss: 0.0705
Epoch 27/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9652 - loss: 0.0974 - val_accuracy: 0.9737 - val_loss: 0.0520
Epoch 28/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9764 - loss: 0.0587 - val_accuracy: 0.9783 - val_loss: 0.0532
Epoch 29/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9770 - loss: 0.0621 - val_accuracy: 0.9760 - val_loss: 0.0501
Epoch 30/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9780 - loss: 0.0618 - val_accuracy: 0.9715 - val_loss: 0.0767
Epoch 31/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9747 - loss: 0.0720 - val_accuracy: 0.9760 - val_loss: 0.0555
Epoch 32/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9769 - loss: 0.0619 - val_accuracy: 0.9749 - val_loss: 0.0502
Epoch 33/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9670 - loss: 0.0986 - val_accuracy: 0.9795 - val_loss: 0.0479
Epoch 34/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9805 - loss: 0.0491 - val_accuracy: 0.9772 - val_loss: 0.0522
Epoch 35/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9788 - loss: 0.0503 - val_accuracy: 0.9795 - val_loss: 0.0525
Epoch 36/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9823 - loss: 0.0448 - val_accuracy: 0.9783 - val_loss: 0.0423
Epoch 37/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9844 - loss: 0.0431 - val_accuracy: 0.9760 - val_loss: 0.0621
Epoch 38/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9826 - loss: 0.0523 - val_accuracy: 0.9783 - val_loss: 0.0529
Epoch 39/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9785 - loss: 0.0621 - val_accuracy: 0.9783 - val_loss: 0.0445
Epoch 40/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9833 - loss: 0.0476 - val_accuracy: 0.9783 - val_loss: 0.0458
Epoch 41/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9821 - loss: 0.0490 - val_accuracy: 0.9806 - val_loss: 0.0445
Epoch 42/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9837 - loss: 0.0444 - val_accuracy: 0.9817 - val_loss: 0.0399
Epoch 43/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9845 - loss: 0.0471 - val_accuracy: 0.9829 - val_loss: 0.0416
Epoch 44/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9856 - loss: 0.0452 - val_accuracy: 0.9829 - val_loss: 0.0425
Epoch 45/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9848 - loss: 0.0483 - val_accuracy: 0.9795 - val_loss: 0.0420
Epoch 46/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9802 - loss: 0.0656 - val_accuracy: 0.9795 - val_loss: 0.0450
Epoch 47/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9842 - loss: 0.0436 - val_accuracy: 0.9806 - val_loss: 0.0389
Epoch 48/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9800 - loss: 0.0560 - val_accuracy: 0.9795 - val_loss: 0.0446
Epoch 49/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9806 - loss: 0.0586 - val_accuracy: 0.9806 - val_loss: 0.0468
Epoch 50/50
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9822 - loss: 0.0458 - val_accuracy: 0.9772 - val_loss: 0.0567
```

طراحی شبکه CNN

در این بخش از شبکه CNN بصورت زیر استفاده شده است. از آنجا که ورودی‌ها به صورت ۵۶۱ ویژگی متوالی هستند، از Conv1D برای استخراج الگوهای محلی استفاده شده است. در مرحله بعد فیلترها از ۶۴ تا ۲۵۶ افزایش یافته‌اند تا شبکه بتواند ویژگی‌های پیچیده‌تری را در لایه‌های بالاتر یاد بگیرد. همچنین از آنجاییکه هسته‌های با اندازه‌ی ۳ در Conv1D برای یادگیری الگوهای محلی کوتاه مناسب‌اند، از Kernel size=3 استفاده شده است. BatchNormalization باعث پایدارسازی یادگیری و تسریع همگرایی می‌شود و از overfitting جلوگیری می‌کند. بخش MaxPooling1D نیز برای کاهش ابعاد و افزایش پایداری ویژگی‌ها در برابر نویز استفاده شده است. بخش Dropout نیز با حذف تصادفی نورون‌ها از overfitting جلوگیری کرده است. همچنین از GlobalAveragePooling1D به جایFlatten استفاده شده است؛ زیرا باعث کاهش تعداد پارامترها می‌شود و از overfittingجلوگیری می‌کند. از لایه‌ی Fully Connected (Dense 128)برای ترکیب ویژگی‌ها قبل از طبقه‌بندی نهایی استفاده شده است. و چون خروجی شامل ۶ کلاس است، از softmax برای احتمال‌دهی نهایی استفاده شده است.

کد مربوط به این بخش بصورت زیر نوشته شده است.

```python
# CNN
from tensorflow.keras.layers import Conv1D, MaxPooling1D, BatchNormalization, GlobalAveragePooling1D

# Reshape baraye Conv1D
X_train_cnn = X_train_final.reshape(-1, 561, 1)
X_test_cnn  = X_test_final.reshape(-1, 561, 1)
y_test_ohe  = tf.keras.utils.to_categorical(y_test_final - 1, n)

model_cnn = Sequential([
    Input(shape=(561,1)),
    Conv1D(64, 3, activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling1D(2),
    Dropout(0.3),

    Conv1D(128, 3, activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling1D(2),
    Dropout(0.3),

    Conv1D(256, 3, activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling1D(2),
    Dropout(0.3),

    GlobalAveragePooling1D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(n, activation='softmax'),
    ])
```

```python
# Adam ba learning rate=0.001
optimizer_cnn = Adam(learning_rate=0.001)
model_cnn.compile(optimizer=optimizer_cnn,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
model_cnn.summary()

# Amuzesh
history_cnn = model_cnn.fit(X_train_cnn, y_train_ohe,
    validation_split=0.1,epochs=30,batch_size=64)

# Arzyabie nahayi
test_loss, test_acc = model_cnn.evaluate(X_test_cnn, y_test_ohe)
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 561, 64)	256
batch_normalization (BatchNormalization)	(None, 561, 64)	256
max_pooling1d (MaxPooling1D)	(None, 280, 64)	0
dropout_2 (Dropout)	(None, 280, 64)	0
conv1d_1 (Conv1D)	(None, 280, 128)	24,704
batch_normalization_1 (BatchNormalization)	(None, 280, 128)	512
max_pooling1d_1 (MaxPooling1D)	(None, 140, 128)	0
dropout_3 (Dropout)	(None, 140, 128)	0
conv1d_2 (Conv1D)	(None, 140, 256)	98,560
batch_normalization_2 (BatchNormalization)	(None, 140, 256)	1,024
max_pooling1d_2 (MaxPooling1D)	(None, 70, 256)	0
dropout_4 (Dropout)	(None, 70, 256)	0
global_average_pooling1d (GlobalAveragePooling1D)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32,896
dropout_5 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 6)	774

Total params: 158,982 (621.02 KB)
Trainable params: 158,086 (617.52 KB)
Non-trainable params: 896 (3.50 KB)

```
Epoch 1/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 17s 69ms/step - accuracy: 0.4689 - loss: 1.1542 - val_accuracy: 0.3687 - val_loss: 3.0785
Epoch 2/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6129 - loss: 0.8231 - val_accuracy: 0.1986 - val_loss: 4.2036
Epoch 3/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.6809 - loss: 0.6702 - val_accuracy: 0.3699 - val_loss: 3.9649
Epoch 4/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.7579 - loss: 0.5510 - val_accuracy: 0.3779 - val_loss: 2.4615
Epoch 5/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.7937 - loss: 0.4754 - val_accuracy: 0.6039 - val_loss: 0.8565
Epoch 6/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.8463 - loss: 0.3823 - val_accuracy: 0.8208 - val_loss: 0.4681
Epoch 7/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8718 - loss: 0.3230 - val_accuracy: 0.9018 - val_loss: 0.2559
Epoch 8/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8822 - loss: 0.2970 - val_accuracy: 0.8927 - val_loss: 0.2453
Epoch 9/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9016 - loss: 0.2666 - val_accuracy: 0.9121 - val_loss: 0.2273
Epoch 10/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9137 - loss: 0.2353 - val_accuracy: 0.9224 - val_loss: 0.1717
Epoch 11/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9152 - loss: 0.2229 - val_accuracy: 0.9475 - val_loss: 0.1400
Epoch 12/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9228 - loss: 0.2216 - val_accuracy: 0.9521 - val_loss: 0.1158
Epoch 13/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9361 - loss: 0.1839 - val_accuracy: 0.9463 - val_loss: 0.1288
Epoch 14/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9361 - loss: 0.1718 - val_accuracy: 0.9692 - val_loss: 0.0970
Epoch 15/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9363 - loss: 0.1668 - val_accuracy: 0.9669 - val_loss: 0.0827
Epoch 16/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9408 - loss: 0.1600 - val_accuracy: 0.9578 - val_loss: 0.1138
Epoch 17/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9484 - loss: 0.1395 - val_accuracy: 0.9452 - val_loss: 0.1316
Epoch 18/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9520 - loss: 0.1323 - val_accuracy: 0.9486 - val_loss: 0.1411
Epoch 19/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9601 - loss: 0.1211 - val_accuracy: 0.9726 - val_loss: 0.0867
Epoch 20/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9555 - loss: 0.1234 - val_accuracy: 0.9692 - val_loss: 0.0769
Epoch 21/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9561 - loss: 0.1142 - val_accuracy: 0.9760 - val_loss: 0.0614
Epoch 22/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9483 - loss: 0.1342 - val_accuracy: 0.9715 - val_loss: 0.0680
Epoch 23/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9580 - loss: 0.1237 - val_accuracy: 0.9726 - val_loss: 0.0617
Epoch 24/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9678 - loss: 0.0941 - val_accuracy: 0.9715 - val_loss: 0.0645
Epoch 25/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9647 - loss: 0.1009 - val_accuracy: 0.9612 - val_loss: 0.0983
Epoch 26/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9645 - loss: 0.0922 - val_accuracy: 0.9772 - val_loss: 0.0652
Epoch 27/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9631 - loss: 0.0994 - val_accuracy: 0.9795 - val_loss: 0.0475
Epoch 28/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9734 - loss: 0.0815 - val_accuracy: 0.9703 - val_loss: 0.0764
Epoch 29/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9692 - loss: 0.0844 - val_accuracy: 0.9589 - val_loss: 0.0999
Epoch 30/30
124/124 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9726 - loss: 0.0780 - val_accuracy: 0.9795 - val_loss: 0.0551
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9748 - loss: 0.0656
Test Accuracy: 0.9773, Test Loss: 0.0667
```

با توجه به نتیجه می‌توان گفت دقت ۹۷.۷۳٪ نشان می‌دهد مدل CNN توانسته الگوهای استخراج‌شده از داده‌های حسگر را به‌خوبی یاد بگیرد. همچنین کاهش مقدار loss نشانه‌ی آموزش مؤثر بدون overfitting جدی است.

بنابراین بصورت کلی معماری طراحی‌شده با استفاده از Conv1D + BatchNorm + MaxPooling + Dropout + GAP تعادلی بین دقت بالا و جلوگیری از overfitting ایجاد کرده است. همچنین با وجود اینکه داده‌ها تصویری نیستند، استفاده از CNN بر روی بردارهای ویژگی نتیجه‌ی مطلوبی داشته است. دقت بالاتر از ۹۳٪ بر روی مجموعه آزمون نشان‌دهنده‌ی مناسب بودن این معماری برای مسئله‌ی طبقه‌بندی فعالیت‌های انسانی است.

ارزیابی

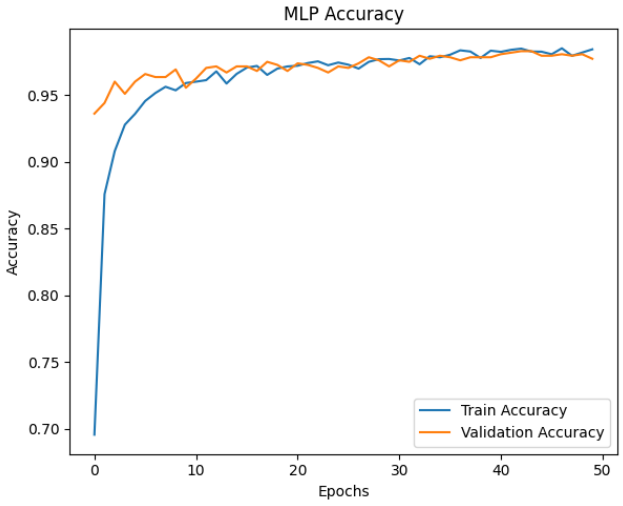
در این بخش از کد زیر استفاده شده است.

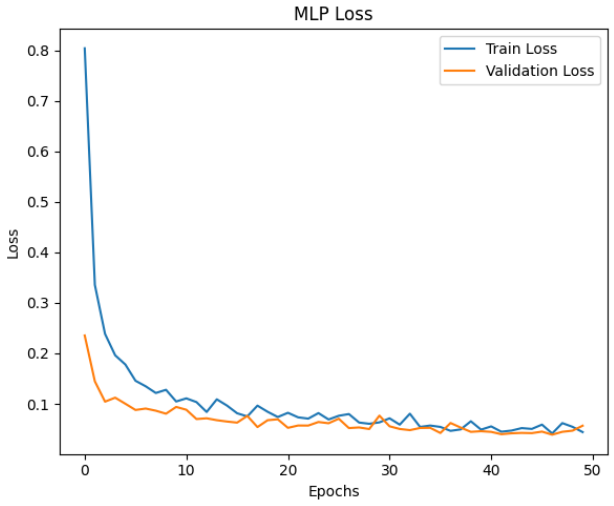
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Nemudare deghat o khata baraye MLP
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('MLP Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('MLP Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

# Arzyabi
mlp_loss, mlp_acc = model.evaluate(X_test_final, y_test_ohe, verbose=0)
print(f"MLP Test Accuracy: {mlp_acc:.4f}")

mlp_preds = np.argmax(model.predict(X_test_final), axis=1) + 1
```

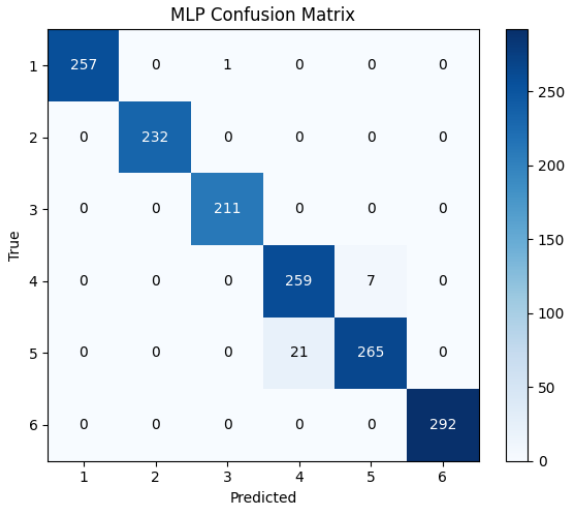
```
mlp_cm = confusion_matrix(y_test_final, mlp_preds)

plt.figure(figsize=(6,5))
plt.imshow(mlp_cm, cmap='Blues')
plt.title('MLP Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.colorbar()
for i in range(6):
    for j in range(6):
        plt.text(j, i, mlp_cm[i, j], ha='center', va='center',
                 color='white' if mlp_cm[i, j] > mlp_cm.max()/2 else 'black')
plt.xticks(np.arange(6), np.arange(1,7))
plt.yticks(np.arange(6), np.arange(1,7))
plt.tight_layout()
plt.show()

print("MLP Classification Report:\n")
print(classification_report(y_test_final, mlp_preds, digits=4))

# Nemudare deghat o khata baraye CNN
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history_cnn.history['accuracy'], label='Train Accuracy')
plt.plot(history_cnn.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history_cnn.history['loss'], label='Train Loss')
plt.plot(history_cnn.history['val_loss'], label='Validation Loss')
plt.title('CNN Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

# Arzyabi
cnn_loss, cnn_acc = model_cnn.evaluate(X_test_cnn, y_test_ohe, verbose=0)
print(f"CNN Test Accuracy: {cnn_acc:.4f}")

cnn_preds = np.argmax(model_cnn.predict(X_test_cnn), axis=1) + 1
cnn_cm = confusion_matrix(y_test_final, cnn_preds)

plt.figure(figsize=(6,5))
plt.imshow(cnn_cm, cmap='Blues')
plt.title('CNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.colorbar()
for i in range(6):
    for j in range(6):
        plt.text(j, i, cnn_cm[i, j], ha='center', va='center',
                 color='white' if cnn_cm[i, j] > cnn_cm.max()/2 else 'black')
plt.xticks(np.arange(6), np.arange(1,7))
plt.yticks(np.arange(6), np.arange(1,7))
```

```
plt.tight_layout()
plt.show()

print("CNN Classification Report:\n")
print(classification_report(y_test_final, cnn_preds, digits=4))
```



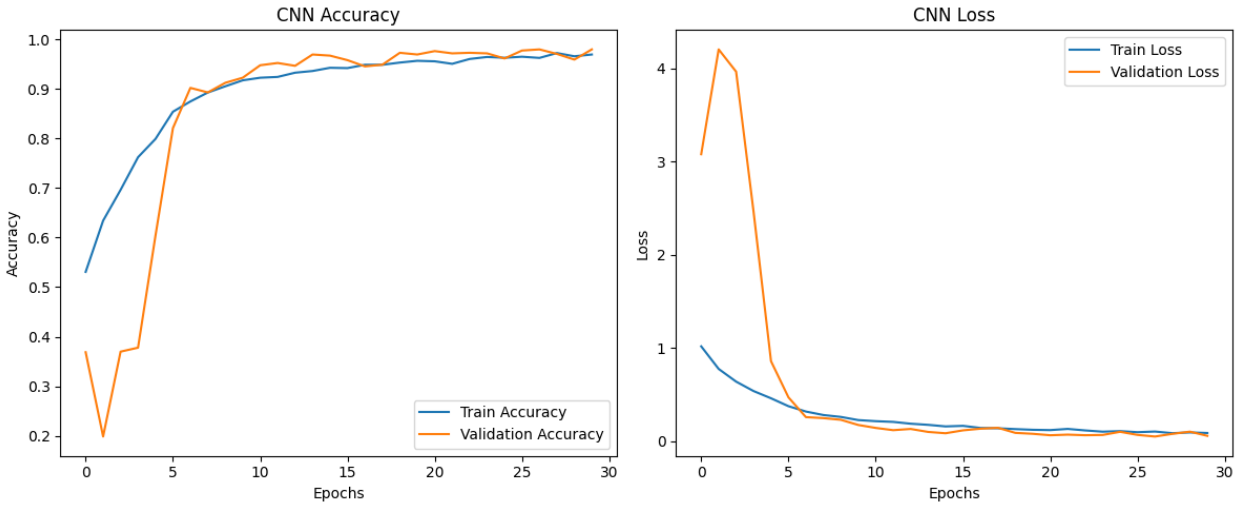


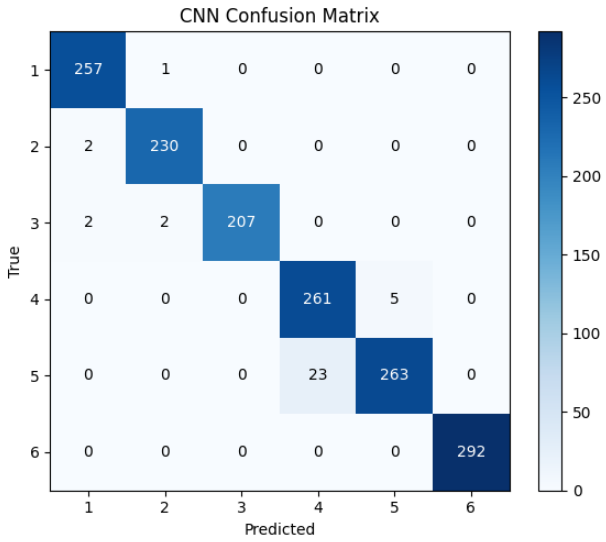



```
MLP Test Accuracy: 0.9812
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step
```

MLP Classification Report:

	precision	recall	f1-score	support
1	1.0000	0.9961	0.9981	258
2	1.0000	1.0000	1.0000	232
3	0.9953	1.0000	0.9976	211
4	0.9250	0.9737	0.9487	266
5	0.9743	0.9266	0.9498	286
6	1.0000	1.0000	1.0000	292
accuracy			0.9812	1545
macro avg	0.9824	0.9827	0.9824	1545
weighted avg	0.9817	0.9812	0.9812	1545





CNN Test Accuracy: 0.9773

49/49 ━━━━━━━━━━━━━━━━━━━━ **2s** 24ms/step

CNN Classification Report:

	precision	recall	f1-score	support
1	0.9847	0.9961	0.9904	258
2	0.9871	0.9914	0.9892	232
3	1.0000	0.9810	0.9904	211
4	0.9190	0.9812	0.9491	266
5	0.9813	0.9196	0.9495	286
6	1.0000	1.0000	1.0000	292
accuracy			0.9773	1545
macro avg	0.9787	0.9782	0.9781	1545
weighted avg	0.9781	0.9773	0.9773	1545

با توجه به نمودار دقت برای MLP می‌توان گفت دقت آموزش و اعتبارسنجی به حدود ۰،۹۸ رسیده‌اند. نوسان کمی در دقت اعتبارسنجی دیده می‌شود که ممکن است به دلیل نرساندن مدل به تعمیم کامل باشد. همچنین از روی نمودار Loss برای این روش نیز می‌توان گفت در چند epoch اول کاهش سریع اتفاق افتاده است. تفاوت بین خطای آموزش و آزمون نسبتاً کم است، که نشان‌دهنده عدم overfitting شدید است.

از روی نمودارهای Loss و دقت CNN نیز می‌توان گفت دقت هر دو مجموعه به حدود ۰،۹۷ تا ۰،۹۹ رسیده‌اند و نسبت به MLP افزایش سریع‌تر در مراحل اولیه مشاهده می‌شود. از طرفی می‌توان گفت خطای آزمون در ابتدا بسیار زیاد است (بیش از ۴)، ولی سپس کاهش شدیدی دارد و بعد از حدود epoch=5 ، مدل بسیار پایدار عمل می‌کند. کاهش loss سریع‌تر از MLP است، که نشان‌دهنده قدرت یادگیری بیشتر CNN در مراحل اولیه است.

همچنین مشاهده می‌شود دقت نهایی روی داده‌های آزمون برای مدل MLP برابر با ۰.۹۸۱۲ و برای CNN برابر با ۰.۹۷۷۳ می‌باشد. می‌توان گفت هر دو مدل عملکرد بسیار خوبی دارند، اما از نظر دقت کلی MLP کمی بهتر عمل کرده است.

از روی ماتریس آشفتگی MLP مشاهده می‌شود اغلب کلاس‌ها به خوبی تشخیص داده شده‌اند و کلاس‌های ۵ و ۴ بیشترین اشتباه را داشته‌اند (مثلاً ۲۱ نمونه از کلاس ۵ به اشتباه در کلاس ۴ افتاده‌اند). از روی ماتریس آشفتگی CNN نیز می‌توان گفت CNN هم در تشخیص کلاس‌های ۴ و ۵ دچار مشکل شده، ولی نسبت به MLP ، کلاس ۳ را کمتر درست پیش‌بینی کرده (۴ نمونه خطا). بنابراین، MLPدر تشخیص کلاس ۳ بهتر است، ولی CNN در کلاس ۵ عملکرد کمی بهتر دارد.

برای تحلیل Classification Report نیز می‌توان گفت برای MLP میانگین دقت، recallو f1 حدود ۰،۹۸ است. کلاس ۵ کمی مشکل‌ساز بوده وکلاس‌های ۲ و ۶ بهترین عملکرد را دارند.(f1=1.00) برای CNN نیز می‌توان گفت عملکرد کلی بسیار خوب است، ولی کلاس ۳ recall=0.981 دارد که نسبت به MLP (۱،۰۰) کمتر است. کلاس ۵ نیز مانند MLP کمی پایین‌تر از دیگر کلاس‌هاست.(recall=0.919)

تحلیل

برای اینکه بتوان گفت کدام مدل بهتر عمل کرده است، معیار ارزیابی مهم است. بعنوان مثال اگر دقت نهایی معیار باشد، MLP بهتر عمل کرده است، اما فاز یادگیری اولیه برای CNN سریعتر بوده و عملکرد بهتری داشته است. در بحث ثبات دقت و خطا CNN کمی نوسانی بوده است و MLP بهتر عمل کرده است. بطور کلی هر دو مدل عملکرد بسیار خوبی دارند، ولی در این آزمایش خاص، MLP عملکرد بهتری در طبقه‌بندی دقیق‌تر دارد و دقت کلی بالاتری کسب کرده است. با این حال، CNN در فاز یادگیری سریع‌تر بوده و انتظار می‌رود که با داده‌های پیچیده‌تر یا تصاویر واقعی‌تر، CNN بهتر تعمیم دهد.

دادگان NEU Surface Defects

آماده‌سازی داده‌ها

در این بخش از کد زیر استفاده شده است. همانطور که مشاهده می‌شود از تابع image_dataset_from_directory برای بارگذاری کل داده‌ها (۱۸۰۰ تصویر) استفاده شده است. هر تصویر در اندازه‌ی 200×200 پیکسل و خاکستری (Grayscale) است. برچسب‌ها به‌صورت int استخراج شده‌اند (یعنی بین ۰ تا ۵، معادل ۶ کلاس عیب مختلف). همچنین برای کاهش تأثیر روشنایی و مقیاس پیکسل‌ها، از روش Min-Max Normalization بین ۰ و ۱ استفاده شد. این روش باعث بهبود همگرایی شبکه در حین آموزش و کاهش حساسیت به شدت روشنایی می‌شود.

```python
# NEU-DET Dataset
from google.colab import files
import zipfile, os

uploaded = files.upload()
zip = list(uploaded.keys())[0]
dir = "NEU_Surface_Defects"
os.makedirs(dir, exist_ok=True)
with zipfile.ZipFile(zip, "r") as zip_ref:
    zip_ref.extractall(dir)

# Masire sahihe pusheha
train_images_dir = os.path.join(dir, "NEU-DET", "train", "images")
val_images_dir   = os.path.join(dir, "NEU-DET", "validation", "images")

# Sehate vujude dadeha
if not (os.path.isdir(train_images_dir) and os.path.isdir(val_images_dir)):
    for root, dirs, files in os.walk(dir):
        print(f" {root} --> {dirs}")
    raise ValueError("Pusheha peyda nashodand")
else:
    print(f"Masire sahih peida shod: \n  Amuzesh: {train_images_dir}\n  Azmoon: {val_images_dir}")

# Bargozarie dadeha ba image_dataset_from_directory
import tensorflow as tf

img_height, img_width = 200, 200
batch_size = 1800  # Kole dadeha yekja
```

```python
train_ds = tf.keras.utils.image_dataset_from_directory(train_images_dir,labels="inferred",
                                                       label_mode="int",color_mode="grayscale",
                                                       batch_size=batch_size,
                                                       image_size=(img_height, img_width),
                                                       shuffle=True,seed=42,)

val_ds = tf.keras.utils.image_dataset_from_directory(val_images_dir,labels="inferred",
                                                     label_mode="int",color_mode="grayscale",
                                                     batch_size=batch_size,
                                                     image_size=(img_height, img_width),
                                                     shuffle=False,)

# Tabdil be NumPy
for images, labels in train_ds.take(1):
    X_train = images.numpy()
    y_train = labels.numpy()

for images, labels in val_ds.take(1):
    X_test = images.numpy()
    y_test = labels.numpy()

# Normalsazi
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0

# Abaade nahayi
print(f"Training set shape: {X_train.shape}, Labels: {y_train.shape}")
print(f"Validation/Test set shape: {X_test.shape}, Labels: {y_test.shape}")
```

```
Saving NEU-DET Dataset.zip to NEU-DET Dataset (1).zip
Masire sahih peida shod:
  Amuzesh: NEU_Surface_Defects/NEU-DET/train/images
  Azmoon: NEU_Surface_Defects/NEU-DET/validation/images
Found 1440 files belonging to 6 classes.
Found 360 files belonging to 6 classes.
Training set shape: (1440, 200, 200, 1), Labels: (1440,)
Validation/Test set shape: (360, 200, 200, 1), Labels: (360,)
```

طراحی شبکه MLP

در این بخش تصاویر دارای ابعاد ۲۰۰×۲۰۰ پیکسلی و خاکستری هستند، که به‌صورت عددی نرمال‌سازی شده‌اند و به عنوان ورودی به مدل داده می‌شوند. با توجه به پیچیدگی پایین‌تر MLP نسبت به شبکه‌های پیچشی، انتظار می‌رود این شبکه عملکرد متوسطی در مقایسه با CNN داشته باشد، اما همچنان به‌عنوان یک مبنای مقایسه‌ای مهم استفاده شده است.

ساختار شبکه‌ی طراحی‌شده به این صورت می‌باشد: لایه ورودی از نوع Input(shape=(200, 200, 1)) ، تصویر خاکستری با اندازه ۲۰۰×۲۰۰، لایه Flatten، لایه پنهان اول از نوع Dense(512, activation='relu')، Dropout(0.3)، لایه پنهان دوم از نوع Dense(128, activation='relu') با نورون کمتر برای یادگیری ویژگی‌های فشرده‌تر و لایه خروجی از نوع Dense(6, activation='softmax') با شش نورون برای دسته‌بندی ۶ کلاس با توزیع احتمالاتی می‌باشد.

تابع فعالسازی ReLU در لایه‌های پنهان استفاده شده است؛ به دلیل اینکه عملکرد محاسباتی سریع‌تری دارد، باعث حل مشکل گرادیان ناپدیدشونده در لایه‌های عمیق می‌شود و در تمرین‌های تجربی نتایج بهتری نسبت به توابع سنتی مانند sigmoid و tanh دارد.

تابع Softmax در لایه خروجی به‌کار رفته است، زیرا برای دسته‌بندی چندکلاسه ضروری است و خروجی هر نورون را به احتمال تعلق تصویر به هر کلاس تبدیل می‌کند.

تابع هزینه انتخاب‌شده در این بخش sparse categorical crossentropy می‌باشد که مناسب برای داده‌هایی که لیبل‌ها به‌صورت عددی هستند، می‌باشد. آموزش مدل با ۲۰ دوره (epoch) و batch_size=32روی داده‌های آموزش انجام شده و عملکرد آن روی داده‌های آزمون ارزیابی شده است.

بطور کلی با توجه به ساده‌بودن ساختار MLP و مسطح‌سازی کامل ورودی تصویر، این شبکه بیشتر ویژگی‌های مکانی (spatial features)تصویر را از بین می‌برد و باعث ناتوانی مدل در یادگیری ویژگی‌های محلی مثل لبه‌ها، الگوها و ساختارهای بافت می‌شود. بنابراین، هرچند قادر به یادگیری الگوهای کلی است، اما در مقایسه با شبکه‌های پیچشی (CNN) انتظار می‌رود دقت پایین‌تری داشته باشد. دقت آموزش در این حالت در ایپاک آخر به ۲۶.۲ درصد و دقت اعتبارسنجی به حدود ۲۳ درصد رسیده است. وضعیت نهایی این مدل این است که عملکرد خوبی نداشته است.

چند روش برای عملکرد بهتر این مدل وجود دارد؛ بعنوان مثال ممکن است تعداد تصاویر در هر کلاس کم باشد، و بدون افزایش داده (Data Augmentation) مدل نتواند ویژگی‌های مهم را بیاموزد. و یا ممکن است داده‌ها در کلاس‌هایی نامتوازن باشند. با افزایش تعداد ایپاک ممکن است عملکرد بهتری مشاهده شود. استفاده از تکنیک هایی مانندBatchNormalization ، EarlyStoppingو ReduceLROnPlateau برای کنترل overfitting و بهبود همگرایی نیز عملکرد را بهتر می‌کند.

```
[ ] from tensorflow.keras import layers, models, optimizers

    mlp_model = models.Sequential([
        layers.Input(shape=(200, 200, 1)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu'),
        layers.Dense(6, activation='softmax')
    ])

    adam_optimizer = optimizers.Adam(learning_rate=0.001)

    mlp_model.compile(
        optimizer=adam_optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    history = mlp_model.fit(
        X_train, y_train_cat,
        validation_data=(X_test, y_test_cat),
        epochs=20,
        batch_size=32
    )
```

```
Epoch 1/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 0.1805 - loss: 17.9118 - val_accuracy: 0.1694 - val_loss: 2.5607
Epoch 2/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.1909 - loss: 3.0125 - val_accuracy: 0.2028 - val_loss: 1.7474
Epoch 3/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.1983 - loss: 1.7829 - val_accuracy: 0.2389 - val_loss: 1.7231
Epoch 4/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.1727 - loss: 1.7620 - val_accuracy: 0.1667 - val_loss: 1.7666
Epoch 5/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2089 - loss: 1.7496 - val_accuracy: 0.1583 - val_loss: 1.7162
Epoch 6/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2045 - loss: 1.7271 - val_accuracy: 0.2472 - val_loss: 1.6940
Epoch 7/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2207 - loss: 1.7049 - val_accuracy: 0.2472 - val_loss: 1.6777
Epoch 8/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2221 - loss: 1.7048 - val_accuracy: 0.1694 - val_loss: 1.7864
Epoch 9/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.1738 - loss: 1.7727 - val_accuracy: 0.2472 - val_loss: 1.7011
Epoch 10/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2505 - loss: 1.6758 - val_accuracy: 0.2750 - val_loss: 1.6507
Epoch 11/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2413 - loss: 1.6815 - val_accuracy: 0.2361 - val_loss: 1.6889
Epoch 12/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2583 - loss: 1.6547 - val_accuracy: 0.1972 - val_loss: 1.6897
Epoch 13/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2155 - loss: 1.7121 - val_accuracy: 0.2250 - val_loss: 1.7178
Epoch 14/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2299 - loss: 1.6987 - val_accuracy: 0.2472 - val_loss: 1.6721
Epoch 15/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2474 - loss: 1.6571 - val_accuracy: 0.2972 - val_loss: 1.6398
Epoch 16/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2696 - loss: 1.6492 - val_accuracy: 0.2167 - val_loss: 1.7190
Epoch 17/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2456 - loss: 1.6614 - val_accuracy: 0.2472 - val_loss: 1.6455
Epoch 18/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2743 - loss: 1.6422 - val_accuracy: 0.2361 - val_loss: 1.6838
Epoch 19/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.2604 - loss: 1.6255 - val_accuracy: 0.2333 - val_loss: 1.6962
Epoch 20/20
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.2622 - loss: 1.6631 - val_accuracy: 0.2278 - val_loss: 1.6676
```

طراحی شبکه CNN

در این بخش از کد زید استفاده شده است. سه بلوک کانولوشن با افزایش تعداد فیلترها استفاده شده است؛ استفاده از فیلترهای ۳۲، ۶۴، و ۱۲۸ امکان یادگیری ویژگی‌های ساده تا پیچیده را فراهم می‌کند. در لایه‌های ابتدایی ویژگی‌های لبه و بافت ساده استخراج می‌شوند، و در لایه‌های عمیق‌تر ویژگی‌های انتزاعی‌تر (مانند الگوهای خرابی سطح) شناسایی می‌شوند. بعد از هر لایه کانولوشن Batch Normalization قرار داده شده تا سرعت آموزش را افزایش دهد، ناپایداری گرادیان را کاهش دهد و از overfitting جلوگیری کند. از MaxPooling برای کاهش ابعاد تصاویر در هر بلوک، کاهش پارامترها و پیچیدگی محاسباتی و افزایش تحمل مدل نسبت به جابجایی اجزای تصویر استفاده شده است. از Dropout نیز برای کاهش overfitting استفاده شده است(۰.۲۵ در بلوکهای میانی، ۰.۵ در لایه Fully Connected). از لایه Fully Connected با ۲۵۶ نورون به‌عنوان طبقه‌بند نهایی استفاده

شده است. همچنین از توابع فعالسازی ReLU(سریع، غیرخطی و بدون مشکل vanishing gradient) و Softmax (مناسب برای دسته‌بندی چندکلاسه) استفاده شده است. تابع هزینه استفاده شده نیز categorical_crossentropy می‌باشد که مناسب برای one-hot encoding می‌باشد.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

# Tabdile y_train , y_test be one-hot
y_train_cat = to_categorical(y_train, num_classes=6)
y_test_cat = to_categorical(y_test, num_classes=6)

model = Sequential()

# Block1
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 1), padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Block2
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Block3
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
# Fully connected
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
کلاسmodel.add(Dense(6, activation='softmax'))  # 6


# Compile model ba categorical_crossentropy
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy',
metrics=['accuracy'])

model.summary()

# Amuzesh
history = model.fit(
X_train, y_train_cat,
validation_data=(X_test, y_test_cat),
epochs=20,
batch_size=32,
verbose=2
)
```

Model: "sequential_3"

Total params: 20,575,366 (78.49 MB)

Trainable params: 20,574,918 (78.49 MB)

Non-trainable params: 448 (1.75 KB)

```
Epoch 1/20
45/45 - 21s - 457ms/step - accuracy: 0.6160 - loss: 26.3987 - val_accuracy: 0.1667 - val_loss: 149.3757
Epoch 2/20
45/45 - 5s - 109ms/step - accuracy: 0.7014 - loss: 9.2118 - val_accuracy: 0.1667 - val_loss: 159.8872
Epoch 3/20
45/45 - 2s - 53ms/step - accuracy: 0.6889 - loss: 2.6539 - val_accuracy: 0.1667 - val_loss: 165.8598
Epoch 4/20
45/45 - 3s - 57ms/step - accuracy: 0.6604 - loss: 1.0407 - val_accuracy: 0.1667 - val_loss: 152.3388
Epoch 5/20
45/45 - 2s - 49ms/step - accuracy: 0.6722 - loss: 0.8327 - val_accuracy: 0.1667 - val_loss: 137.4172
Epoch 6/20
```

45/45 - 3s - 56ms/step - accuracy: 0.6938 - loss: 0.8237 - val_accuracy: 0.1667 - val_loss: 129.9381
Epoch 7/20
45/45 - 2s - 55ms/step - accuracy: 0.6986 - loss: 0.7576 - val_accuracy: 0.1667 - val_loss: 112.7870
Epoch 8/20
45/45 - 2s - 49ms/step - accuracy: 0.7458 - loss: 0.6878 - val_accuracy: 0.1667 - val_loss: 110.5494
Epoch 9/20
45/45 - 3s - 56ms/step - accuracy: 0.7451 - loss: 0.7913 - val_accuracy: 0.1667 - val_loss: 93.5765
Epoch 10/20
45/45 - 3s - 57ms/step - accuracy: 0.7646 - loss: 0.6532 - val_accuracy: 0.1667 - val_loss: 84.0687
Epoch 11/20
45/45 - 3s - 56ms/step - accuracy: 0.7611 - loss: 0.6433 - val_accuracy: 0.1667 - val_loss: 63.6930
Epoch 12/20
45/45 - 3s - 58ms/step - accuracy: 0.7792 - loss: 0.5574 - val_accuracy: 0.1694 - val_loss: 51.6583
Epoch 13/20
45/45 - 2s - 55ms/step - accuracy: 0.7729 - loss: 0.6296 - val_accuracy: 0.2472 - val_loss: 30.5611
Epoch 14/20
45/45 - 2s - 49ms/step - accuracy: 0.7861 - loss: 0.5233 - val_accuracy: 0.4444 - val_loss: 15.5560
Epoch 15/20
45/45 - 3s - 57ms/step - accuracy: 0.8132 - loss: 0.5904 - val_accuracy: 0.6028 - val_loss: 3.3054
Epoch 16/20
45/45 - 2s - 49ms/step - accuracy: 0.7403 - loss: 0.7619 - val_accuracy: 0.4500 - val_loss: 3.3433
Epoch 17/20
45/45 - 2s - 50ms/step - accuracy: 0.7604 - loss: 0.6370 - val_accuracy: 0.5694 - val_loss: 3.3776
Epoch 18/20
45/45 - 2s - 49ms/step - accuracy: 0.7965 - loss: 0.5334 - val_accuracy: 0.5167 - val_loss: 3.2538
Epoch 19/20
45/45 - 3s - 60ms/step - accuracy: 0.8125 - loss: 0.4969 - val_accuracy: 0.6389 - val_loss: 1.6268
Epoch 20/20
45/45 - 2s - 53ms/step - accuracy: 0.8306 - loss: 0.4677 - val_accuracy: 0.5667 - val_loss: 4.5754

با استفاده از نتایج مشاهده می‌شود مدل روی داده‌های آموزش بسیار خوب عمل کرده اما روی داده‌های آزمون () دچار افت شده است؛ یعنی احتمال Overfitting وجود دارد. همچنین مشاهده می‌شود کلاس‌هایی مانند crazing و rolled-in_scale به‌خوبی تشخیص داده شده‌اند ولی کلاس‌هایی مثل scratches و inclusion ضعیف شناسایی شده‌اند که این اتفاق ممکن است به این دلیل باشد که داده کافی نداشته باشند.

راه هایی برای بهبود این مدل وجود دارد. افزایش داده یاData Augmentation مانند چرخش، کشش، تغییر روشنایی و نویز می‌تواند کمک کند مدل تعمیم‌پذیرتر شود. همچنین استفاده از EarlyStopping و کاهش learning_rate درepoch های

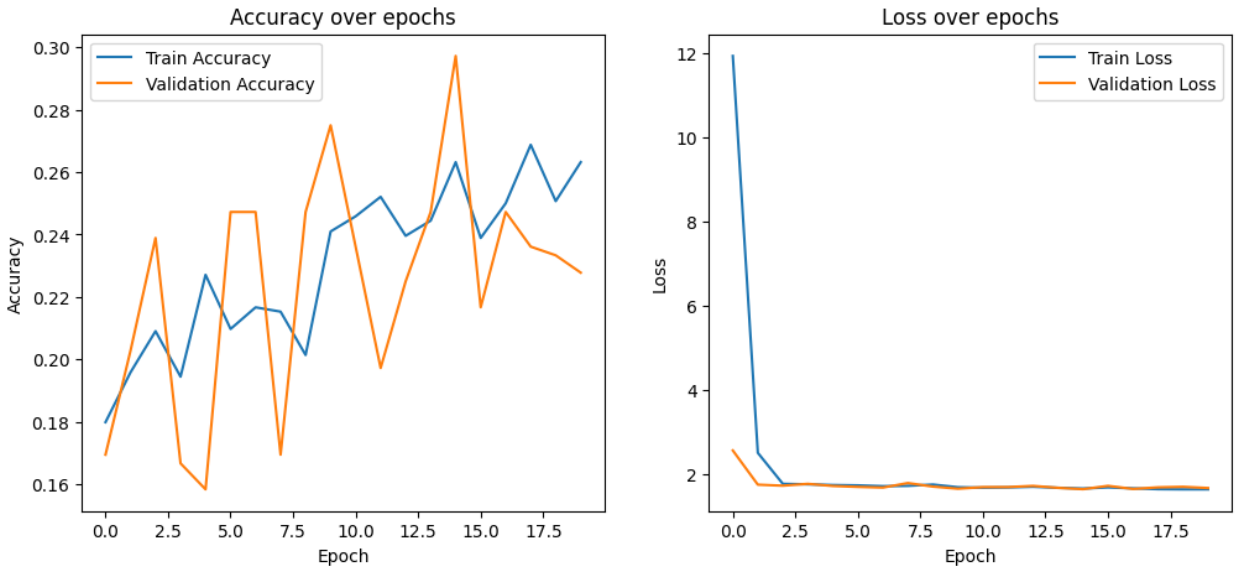
خاص هم می‌تواند کمک کند تا از overfitting جلوگیری شود. کاهش یا تغییر نرخDropout و یا افزایش تعداد فیلترها در لایه‌های عمیق نیز می‌توانند موثر باشند.

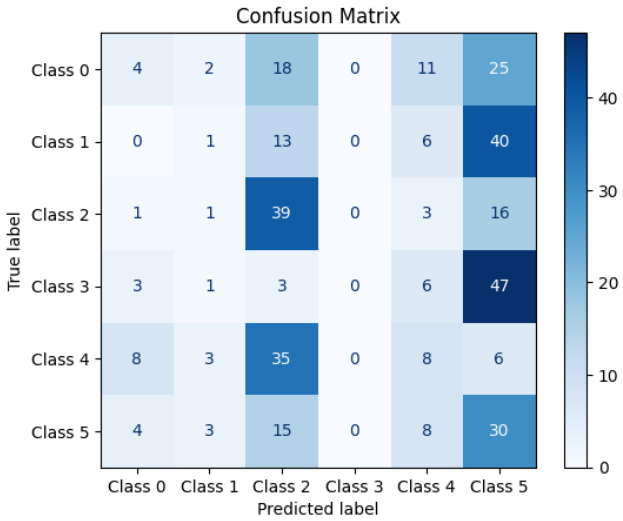
بطور کلی معماری طراحی‌شده برای شروع کار مناسب است؛ زیرا مدل به‌خوبی آموزش دیده اما روی داده‌های آزمون کمی دچار overfitting است و استفاده از تکنیک‌های بهبود تعمیم‌پذیری مثل data augmentation و regularization عملکرد را بهتر می‌کند. توصیه می‌شود.

ارزیابی

در این بخش کدها مشابه بخش قبل می‌باشند. نتایج در زیر قابل مشاهده می‌باشند.

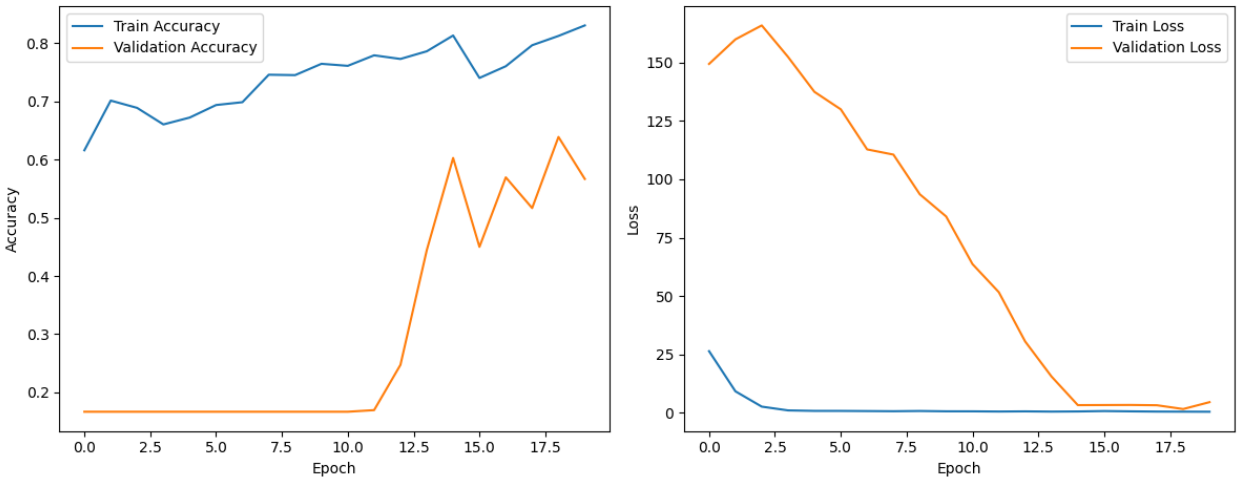
MLP:



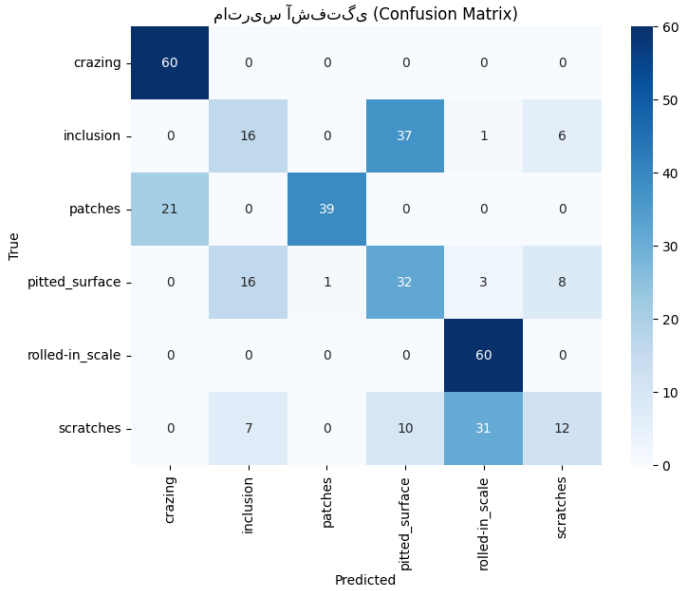



```
Test accuracy: 22.78%
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step
```

CNN:



دقت نهایی مدل روی داده‌های آزمون: ۵۶.۶۷%



Classification Report:

	precision	recall	f1-score	support
crazing	0.74	1.00	0.85	60
inclusion	0.41	0.27	0.32	60
patches	0.97	0.65	0.78	60
pitted_surface	0.41	0.53	0.46	60
rolled-in_scale	0.63	1.00	0.77	60
scratches	0.46	0.20	0.28	60
accuracy			0.61	360
macro avg	0.60	0.61	0.58	360
weighted avg	0.60	0.61	0.58	360

تغییر هایپرپارامترها

همانطور که می‌دانیم در شبکه‌های کانولوشن(CNN) ، لایه‌های Dropout معمولی به صورت تصادفی برخی نورون‌ها را صفر می‌کنند. اما این روش در لایه‌های کانولوشن به خوبی عمل نمی‌کند، زیرا اطلاعات مکانی (spatial information) بین پیکسل‌ها حفظ نمی‌شود. دلیل جایگزینی Dropout با Block Dropout (SpatialDropout2D) به این خاطر است که Block Dropout به جای صفر کردن مقادیر تصادفی در تمام مکان‌ها، کل (feature maps)را به صورت بلوکی و تصادفی صفر می‌کند. این کار باعث می‌شود همبستگی مکانی (spatial correlation) حفظ شود، از وابستگی بیش از حد مدل جلوگیری شود و مدل مقاومت بیشتری نسبت به overfitting داشته باشد. به همین دلیل، استفاده از SpatialDropout2Dدر لایه‌های کانولوشن نسبت به Dropout معمولی مناسب‌تر است.

با مشاهده نتایج می‌توان گفت بهبود چشمگیر در دوره‌های انتهایی یعنی ایپاک ۱۶ تا ۱۹ مشاهده می‌شود که نشان‌دهنده تأثیر مثبت استفاده از SpatialDropout2D است. در صورت استفاده ازمعمولی یا عدم استفاده همگرایی آهسته‌تر ، Overfitting زودتر اتفاق می‌افتد و دقت اعتبارسنجی (Validation Accuracy) کمتر می‌شود؛ همچنین Loss روی داده‌های اعتبارسنجی در اواسط آموزش کاهش نمی‌یابد یا نوسان دارد. اما با روش گفته شده، مدل با داده‌های جدید بهتر سازگار شده، Overfitting کاهش می‌یابد و دقت اعتبارسنجی در Epoch 19 به ۷۱.۱ درصد رسید، که نشان‌دهنده‌ی یادگیری بهتر مدل است.

```
from tensorflow.keras.layers import SpatialDropout2D

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 1), padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.25))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(SpatialDropout2D(0.25))

# Fully connected
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=32,
    verbose=2
)
```

```
Epoch 1/20
45/45 - 10s - 213ms/step - accuracy: 0.5090 - loss: 23.3934 - val_accuracy: 0.2278 - val_loss: 42.3366
Epoch 2/20
45/45 - 5s - 117ms/step - accuracy: 0.5236 - loss: 5.0283 - val_accuracy: 0.1667 - val_loss: 83.8222
Epoch 3/20
45/45 - 2s - 55ms/step - accuracy: 0.4083 - loss: 1.7286 - val_accuracy: 0.1667 - val_loss: 95.7784
Epoch 4/20
45/45 - 2s - 51ms/step - accuracy: 0.4104 - loss: 1.5912 - val_accuracy: 0.1667 - val_loss: 114.9981
Epoch 5/20
45/45 - 2s - 51ms/step - accuracy: 0.4882 - loss: 1.2861 - val_accuracy: 0.1667 - val_loss: 100.6921
Epoch 6/20
45/45 - 2s - 46ms/step - accuracy: 0.5035 - loss: 1.2938 - val_accuracy: 0.1667 - val_loss: 97.4564
Epoch 7/20
45/45 - 3s - 56ms/step - accuracy: 0.5278 - loss: 1.1394 - val_accuracy: 0.1667 - val_loss: 102.8309
Epoch 8/20
45/45 - 2s - 47ms/step - accuracy: 0.5097 - loss: 1.1235 - val_accuracy: 0.1667 - val_loss: 79.4282
Epoch 9/20
45/45 - 3s - 57ms/step - accuracy: 0.5236 - loss: 1.1027 - val_accuracy: 0.1722 - val_loss: 50.7619
```

Epoch 10/20
45/45 - 2s - 47ms/step - accuracy: 0.5389 - loss: 1.1395 - val_accuracy: 0.1833 - val_loss: 51.4639
Epoch 11/20
45/45 - 3s - 57ms/step - accuracy: 0.5597 - loss: 1.0687 - val_accuracy: 0.1694 - val_loss: 27.9791
Epoch 12/20
45/45 - 2s - 46ms/step - accuracy: 0.5521 - loss: 1.0738 - val_accuracy: 0.2000 - val_loss: 13.7715
Epoch 13/20
45/45 - 2s - 46ms/step - accuracy: 0.5833 - loss: 1.0068 - val_accuracy: 0.2861 - val_loss: 4.6582
Epoch 14/20
45/45 - 2s - 51ms/step - accuracy: 0.5972 - loss: 0.9735 - val_accuracy: 0.4167 - val_loss: 1.6679
Epoch 15/20
45/45 - 2s - 47ms/step - accuracy: 0.5604 - loss: 1.0579 - val_accuracy: 0.2972 - val_loss: 1.5276
Epoch 16/20
45/45 - 3s - 56ms/step - accuracy: 0.5757 - loss: 1.0202 - val_accuracy: 0.6500 - val_loss: 0.9521
Epoch 17/20
45/45 - 3s - 56ms/step - accuracy: 0.5910 - loss: 0.9524 - val_accuracy: 0.5500 - val_loss: 1.0015
Epoch 18/20
45/45 - 2s - 47ms/step - accuracy: 0.6139 - loss: 0.9503 - val_accuracy: 0.6417 - val_loss: 0.8419
Epoch 19/20
45/45 - 3s - 57ms/step - accuracy: 0.6056 - loss: 0.9445 - val_accuracy: 0.7111 - val_loss: 0.7641
Epoch 20/20
45/45 - 2s - 47ms/step - accuracy: 0.6028 - loss: 0.9881 - val_accuracy: 0.6417 - val_loss: 0.8274

در شبکه‌های عصبی کانولوشن(CNN) ، لایه‌های کانولوشنی معمولاً از کرنل‌های دوبعدی مربعی استفاده می‌کنند. اما می‌توان این کرنل‌ها را به صورت حاصل‌ضرب دو فیلتر یک‌بعدی تجزیه کرد. به این تکنیک تجزیه فیلتر یا Kernel Factorization گفته می‌شود. از مزایای این روش می‌توان به کاهش تعداد پارامترها، کاهش پیچیدگی محاسباتی، افزایش سرعت یادگیری و جلوگیری از overfitting اشاره نمود. کد مربوط به این بخش در زیر آمده است.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
SpatialDropout2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

y_train_cat = to_categorical(y_train, num_classes=6)
y_test_cat = to_categorical(y_test, num_classes=6)
```

```
model = Sequential()

model.add(Conv2D(32, (3, 1), activation='relu', input_shape=(200, 200, 1), padding='same'))
model.add(Conv2D(32, (1, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.25))

model.add(Conv2D(64, (3, 1), activation='relu', padding='same'))
model.add(Conv2D(64, (1, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.25))

model.add(Conv2D(128, (3, 1), activation='relu', padding='same'))
model.add(Conv2D(128, (1, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(SpatialDropout2D(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(6, activation='softmax'))

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

history = model.fit(
```

```
    X_train, y_train_cat,
    validation_data=(X_test, y_test_cat),
    epochs=20,
    batch_size=32,
    verbose=2
)
```

Total params: 20,578,470 (78.50 MB)

Trainable params: 20,578,022 (78.50 MB)

Non-trainable params: 448 (1.75 KB)

```
Epoch 1/20
45/45 - 23s - 515ms/step - accuracy: 0.4764 - loss: 15.3413 - val_accuracy: 0.1667 - val_loss: 38.9404
Epoch 2/20
45/45 - 9s - 200ms/step - accuracy: 0.4542 - loss: 1.9560 - val_accuracy: 0.1528 - val_loss: 56.0181
Epoch 3/20
45/45 - 5s - 113ms/step - accuracy: 0.5069 - loss: 1.3927 - val_accuracy: 0.1667 - val_loss: 79.4787
Epoch 4/20
45/45 - 5s - 110ms/step - accuracy: 0.5472 - loss: 1.1808 - val_accuracy: 0.1667 - val_loss: 89.6115
Epoch 5/20
45/45 - 5s - 114ms/step - accuracy: 0.5479 - loss: 1.1420 - val_accuracy: 0.1667 - val_loss: 96.5054
Epoch 6/20
45/45 - 5s - 115ms/step - accuracy: 0.5035 - loss: 1.1881 - val_accuracy: 0.1667 - val_loss: 95.5611
Epoch 7/20
45/45 - 3s - 74ms/step - accuracy: 0.4965 - loss: 1.2017 - val_accuracy: 0.1667 - val_loss: 93.3838
Epoch 8/20
45/45 - 3s - 77ms/step - accuracy: 0.4917 - loss: 1.0839 - val_accuracy: 0.1667 - val_loss: 74.2160
Epoch 9/20
45/45 - 3s - 75ms/step - accuracy: 0.5514 - loss: 1.1777 - val_accuracy: 0.1389 - val_loss: 57.0526
Epoch 10/20
45/45 - 5s - 113ms/step - accuracy: 0.4965 - loss: 1.2665 - val_accuracy: 0.2472 - val_loss: 31.1885
Epoch 11/20
45/45 - 5s - 116ms/step - accuracy: 0.5208 - loss: 1.1501 - val_accuracy: 0.2500 - val_loss: 32.4319
Epoch 12/20
45/45 - 4s - 78ms/step - accuracy: 0.5375 - loss: 1.1708 - val_accuracy: 0.2389 - val_loss: 15.7887
Epoch 13/20
45/45 - 3s - 75ms/step - accuracy: 0.5111 - loss: 1.1412 - val_accuracy: 0.3000 - val_loss: 6.8975
Epoch 14/20
45/45 - 5s - 118ms/step - accuracy: 0.5618 - loss: 1.1268 - val_accuracy: 0.3361 - val_loss: 5.6332
Epoch 15/20
45/45 - 4s - 78ms/step - accuracy: 0.5472 - loss: 1.1924 - val_accuracy: 0.4361 - val_loss: 2.7339
```

Epoch 16/20
45/45 - 4s - 79ms/step - accuracy: 0.4951 - loss: 1.2390 - val_accuracy: 0.2611 - val_loss: 8.6130
Epoch 17/20
45/45 - 5s - 110ms/step - accuracy: 0.5083 - loss: 1.2143 - val_accuracy: 0.4250 - val_loss: 1.0658
Epoch 18/20
45/45 - 4s - 78ms/step - accuracy: 0.5146 - loss: 1.1933 - val_accuracy: 0.4528 - val_loss: 1.9567
Epoch 19/20
45/45 - 3s - 76ms/step - accuracy: 0.5597 - loss: 1.0246 - val_accuracy: 0.5667 - val_loss: 1.4523
Epoch 20/20
45/45 - 4s - 79ms/step - accuracy: 0.5944 - loss: 1.0105 - val_accuracy: 0.5111 - val_loss: 1.1562

با مشاهده نتایج می‌توان گفتتعداد کل پارامترها20,578,470 ، بیشینه دقت اعتبارسنجی۵۶.۶۷ درصد و مقدارloss اعتبارسنجی تا ۱.۰۶۵۸ کاهش یافته و دقت نهایی آموزش ۵۹.۴۴ درصد می‌باشد.

تحلیل

با نتایج بالا می‌توان گفت مدل اصلی بخش قبل (بدون تجزیه فیلتر) عملکرد بهتری داشته است. علت این اتفاق را می‌توان آن دانست که فیلترهای مربعی کلاسیک (مثلاً ۳×۳) به طور کامل‌تر ویژگی‌های دوبعدی تصویر را استخراج می‌کنند؛ این مهم است چون عیوب سطحی معمولاً الگوهایی پیچیده و دوبعدی دارند. همچنین مدل اصلی به دقت اعتبارسنجی بالاتری رسیده که نشان می‌دهد توانایی تعمیم بهتری به داده‌های جدید دارد. در مدل با تجزیه فیلتر، به‌رغم کاهش پارامترها و سرعت بیشتر، دقت مدل از حدی بالاتر نرفته (اشباع در ۵۶–۵۷٪) و مدل با نوسانات زیاد، دچار تأخیر در همگرایی شده است.

بطور کلی می‌توان گفت مدل بدون تجزیه فیلتر انتخاب بهتری است. اما اگر هدف سبکتر و سریعتر بودن مدل باشد، مدل Kernel Factorization گزینه خوبی خواهد بود.

در این بخش بهبودهای گفته شده انجام گرفت. مشاهده میشود مدل عملکرد بهتری دارد.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.utils import to_categorical

y_train_cat = to_categorical(y_train, num_classes=6)
y_test_cat = to_categorical(y_test, num_classes=6)
```

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 1), padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(X_train)

model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
```

```
    datagen.flow(X_train, y_train_cat, batch_size=32),
    validation_data=(X_test, y_test_cat),
    epochs=40,
    verbose=2
)
```

```
Epoch 1/40
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
45/45 - 131s - 3s/step - accuracy: 0.4875 - loss: 11.0918 - val_accuracy: 0.1667 - val_loss: 58.7910
Epoch 2/40
45/45 - 127s - 3s/step - accuracy: 0.6389 - loss: 1.1957 - val_accuracy: 0.1667 - val_loss: 84.8249
Epoch 3/40
45/45 - 132s - 3s/step - accuracy: 0.6451 - loss: 1.0344 - val_accuracy: 0.1667 - val_loss: 77.3960
Epoch 4/40
45/45 - 130s - 3s/step - accuracy: 0.6542 - loss: 0.9625 - val_accuracy: 0.1667 - val_loss: 70.1174
Epoch 5/40
45/45 - 140s - 3s/step - accuracy: 0.6771 - loss: 0.8645 - val_accuracy: 0.1667 - val_loss: 60.4545
Epoch 6/40
45/45 - 132s - 3s/step - accuracy: 0.7146 - loss: 0.7562 - val_accuracy: 0.3278 - val_loss: 41.8444
Epoch 7/40
45/45 - 136s - 3s/step - accuracy: 0.7243 - loss: 0.7625 - val_accuracy: 0.1667 - val_loss: 40.6388
Epoch 8/40
45/45 - 141s - 3s/step - accuracy: 0.7167 - loss: 0.7665 - val_accuracy: 0.2889 - val_loss: 47.5310
Epoch 9/40
45/45 - 149s - 3s/step - accuracy: 0.6861 - loss: 0.7629 - val_accuracy: 0.3028 - val_loss: 51.2314
Epoch 10/40
45/45 - 131s - 3s/step - accuracy: 0.7382 - loss: 0.6848 - val_accuracy: 0.3056 - val_loss: 38.9176
Epoch 11/40
45/45 - 134s - 3s/step - accuracy: 0.7458 - loss: 0.7073 - val_accuracy: 0.3111 - val_loss: 30.2662
Epoch 12/40
45/45 - 143s - 3s/step - accuracy: 0.7389 - loss: 0.7663 - val_accuracy: 0.4833 - val_loss: 24.2261
Epoch 13/40
45/45 - 144s - 3s/step - accuracy: 0.7410 - loss: 0.7379 - val_accuracy: 0.3639 - val_loss: 14.9825
```

Epoch 14/40

45/45 - 194s - 4s/step - accuracy: 0.7667 - loss: 0.6397 - val_accuracy: 0.3333 - val_loss: 31.9696

Epoch 15/40

45/45 - 135s - 3s/step - accuracy: 0.7563 - loss: 0.6251 - val_accuracy: 0.3167 - val_loss: 42.3301

Epoch 16/40

45/45 - 132s - 3s/step - accuracy: 0.7819 - loss: 0.5824 - val_accuracy: 0.4778 - val_loss: 15.1470

Epoch 17/40

45/45 - 143s - 3s/step - accuracy: 0.7840 - loss: 0.5754 - val_accuracy: 0.6444 - val_loss: 2.1219

Epoch 18/40

45/45 - 195s - 4s/step - accuracy: 0.7806 - loss: 0.5732 - val_accuracy: 0.5472 - val_loss: 6.4037

Epoch 19/40

45/45 - 136s - 3s/step - accuracy: 0.7924 - loss: 0.5452 - val_accuracy: 0.4194 - val_loss: 16.3622

Epoch 20/40

45/45 - 141s - 3s/step - accuracy: 0.7986 - loss: 0.5524 - val_accuracy: 0.5417 - val_loss: 4.7034

Epoch 21/40

45/45 - 146s - 3s/step - accuracy: 0.8285 - loss: 0.4441 - val_accuracy: 0.5028 - val_loss: 9.1358

Epoch 22/40

45/45 - 136s - 3s/step - accuracy: 0.8083 - loss: 0.4954 - val_accuracy: 0.5861 - val_loss: 7.0024

Epoch 23/40

45/45 - 137s - 3s/step - accuracy: 0.8049 - loss: 0.5213 - val_accuracy: 0.5389 - val_loss: 5.3208

Epoch 24/40

45/45 - 136s - 3s/step - accuracy: 0.8188 - loss: 0.4588 - val_accuracy: 0.4250 - val_loss: 10.3717

Epoch 25/40

45/45 - 142s - 3s/step - accuracy: 0.8208 - loss: 0.4915 - val_accuracy: 0.6028 - val_loss: 1.0969

Epoch 26/40

یادگیری انتقالی

آماده‌سازی داده‌ها

کد مربوط به این بخش در زیر آمده است. همانطور که مشاهده می‌شود در ابتدا مسیر train_dir و val_dir شامل تصاویر آموزش و اعتبارسنجی دسته‌بندی‌شده در زیرپوشه‌ها تنظیم شده‌اند. در مرحله بعد تصاویر به اندازه 224×224 تغییر داده شده‌اند که ورودی استاندارد ResNet-50 است. سپس Data Augmentation انجام گرفته است؛ به این صورت که تغییرات تصادفی از جمله چرخش، انتقال افقی و عمودی، زوم و برش و آینه‌ای کردن تصویر برای افزایش تعمیم‌پذیری و کاهشoverfitting صورت می‌گیرد. سپس از تابع preprocess_input برای مقیاس‌بندی مناسب با ResNet استفاده شده است(پیش پردازش). پس از آن از ImageDataGenerator.flow_from_directory برای بارگذاری داده‌ها همراه با برچسب‌گذاری خودکار از پوشه‌ها استفاده شد. و چون کلاس‌ها چندگانه‌اند کلاس‌بندی با categorical صورت گرفت.در انتهای کد، به‌منظور بررسی داده‌ها، از هر کلاس به‌صورت تصادفی یک تصویر به همراه برچسب آن نمایش داده شده است. با توجه به کد و نتایج می‌توان مشاهده نمود که داده‌ها کاملاً قابل تفکیک بوده و برای آموزش شبکه مناسب‌اند. همچنین طبق گزارش train_generator.class_indicesو val_generator.classes، از هر کلاس دقیقاً ۲۴۰ نمونه در آموزش و ۶۰ نمونه در اعتبارسنجی وجود دارد.

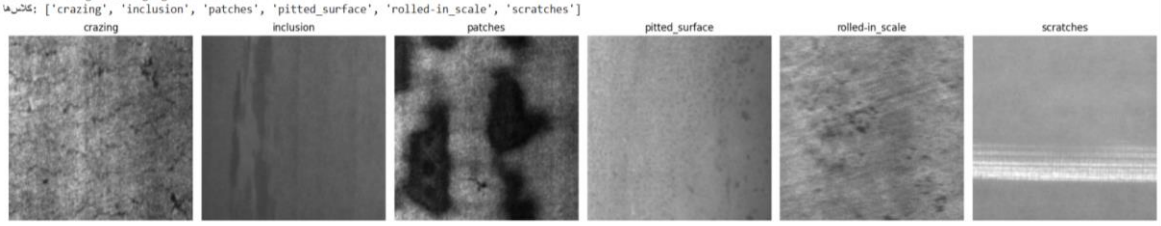
لازم به ذکر است در کد از تکنیک‌های افزایش داده مانند چرخش، انتقال، زوم، shearو افقی‌سازی برای افزایش تنوع ظاهری تصاویر، بهبود تعمیم‌پذیری مدل و کاهش overfitting به داده‌های آموزشی استفاده شده است.

```python
import os
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import preprocess_input
import random

train_dir = '/content/NEU_Surface_Defects/NEU-DET/train/images'
val_dir = '/content/NEU_Surface_Defects/NEU-DET/validation/images'

# Andaze standard vurudie ResNet-50
Image_Size = (224, 224)
Batch_Size = 32
SEED = 42

# Tarife ImageDataGenerator ba Augmentation
train_data = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_data = ImageDataGenerator(
```

```
    preprocessing_function=preprocess_input  # Normalsazi
)

# Datagenerator baraye pusheha
train_generator = train_data.flow_from_directory(
    train_dir,
    target_size=Image_Size,
    color_mode='rgb',
    batch_size=Batch_Size,
    class_mode='categorical',
    shuffle=True,
    seed=SEED
)

val_generator = val_data.flow_from_directory(
    val_dir,
    target_size=Image_Size,
    color_mode='rgb',
    batch_size=Batch_Size,
    class_mode='categorical',
    shuffle=False,
    seed=SEED
)

# Nemuneye tasadofi az har class
class_names = list(train_generator.class_indices.keys())
print("classes:", class_names)

fig, axes = plt.subplots(1, len(class_names), figsize=(20, 5))
```

```
for i, class_name in enumerate(class_names):
    class_path = os.path.join(train_dir, class_name)
    img_name = random.choice(os.listdir(class_path))
    img_path = os.path.join(class_path, img_name)

    img = plt.imread(img_path)
    axes[i].imshow(img, cmap='gray')
    axes[i].set_title(class_name)
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```

```
Found 1440 images belonging to 6 classes.
Found 360 images belonging to 6 classes.
کلاس‌ها: ['crazing', 'inclusion', 'patches', 'pitted_surface', 'rolled-in_scale', 'scratches']
```



آماده‌سازی مدل

در این بخش داده‌ها از مسیر .../content/NEU_Surface_Defects/NEU-DET/ خوانده شده‌اند. سپس از ImageDataGenerator با توابع افزایش داده استفاده شده است(چرخش تا ۲۰ درجه، بزرگ‌نمایی تا ٪۲۰ و وارونگی افقی). سپس تصاویر به اندازه‌ی استاندارد ۲۲۴ در ۲۲۴ و در سه کانال رنگی (RGB) تبدیل شده‌اند. و همچنین کلاس‌ها به‌صورت categorical (one-hot) پردازش شده‌اند(تعداد کلاس‌ها۶ می‌باشد). با توجه به کد واضح است از شبکه‌ی پیش‌آموزش‌دیده‌ی ResNet-50 با وزن‌های ImageNet استفاده شده است. لایه‌های بالایی Fully Connected و Softmax اصلی حذف شده‌اند .(include_top=False)، همچنین وزن‌های مدل پایه freeze شده‌اند (trainable=False)تا فقط لایه‌های جدید آموزش ببینند. پس از ResNet-50 ، لایه‌های GlobalAveragePooling2D(کاهش ابعاد ویژگی‌های مکانی)، Dropout(0.5)، Dense(128, relu)، Dropout(0.5) و Dense(6, softmax) استفاده شده‌اند.

با استفاده از نتایج می‌توان گفت مدلی قوی و بهینه برای شروع یادگیری انتقالی (Transfer Learning) پیاده‌سازی شده که از قدرت استخراج ویژگی‌های عمیق ResNet-50 استفاده می‌کند، با فریز کردن لایه‌های پایه، از یادگیری مجدد غیرضروری جلوگیری

می‌شود، با افزودن لایه‌های Fully Connected وDropout ، ظرفیت تمایز برای ۶ کلاس خاص مسئله فراهم شده است.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Abaade vurudi baraye ResNet50
IMG_SIZE = 224
BATCH_SIZE = 32
NUM_CLASSES = 6

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator = train_datagen.flow_from_directory(
    '/content/NEU_Surface_Defects/NEU-DET/train/images',
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='rgb'
)

val_generator = val_datagen.flow_from_directory(
    '/content/NEU_Surface_Defects/NEU-DET/validation/images',
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=False
)

# Bargozarie model paye ResNet50 bedune layehaye balayi
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE,
3))
base_model.trainable = False  # Freeze kardane vaznha
```

```
# Afzudane layehaye jadid be balaye shabake
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(NUM_CLASSES, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

مرحله آموزش

در مرحله اول، با هدف بهره‌گیری از قابلیت‌های از پیش‌آموخته‌شده شبکه‌ی ResNet-50 آموزش دیده رویImageNet ، تمامی لایه‌های مدل پایه غیرفعال (Freeze) شدند. در نتیجه، فقط لایه‌های جدیدی که در بالای مدل افزوده شده بودند، آموزش دیده‌اند. این لایه‌ها شامل یک لایه‌ی (Global Average Pooling)، دو لایه‌ی Dropout برای جلوگیری از (Overfitting)، یک لایه‌ی Fully Connected با ۱۲۸ نورون و تابع فعال‌سازیReLU ، و در نهایت یک لایه‌ی خروجی با تابع softmax برای طبقه‌بندی ۶ کلاس هستند. مدل با نرخ یادگیری ۰.۰۰۱ و برای ۱۰ دوره (epoch) آموزش داده شد. در این مرحله، هدف اصلی تطبیق لایه‌های بالایی با داده‌های جدید و استفاده از ویژگی‌های عمومی استخراج‌شده توسط شبکه‌ی پایه بود. کد و نتایج این بخش در زیر قابل مشاهده می‌باشد.

در مرحله دوم پس از آموزش اولیه، برای بهبود عملکرد مدل و افزایش توان تمایز آن نسبت به داده‌های تخصصی‌تر (عیوب سطحی)، فرآیند Fine-Tuning انجام شد. در این مرحله، ۳۰ لایه‌ی انتهایی شبکه‌ی پایه ResNet-50 از حالت Freeze خارج شدند تا بتوانند با نرخ یادگیری پایین‌تر، خود را با داده‌های جدید سازگار کنند. مدل مجدداً کامپایل شد، اما این بار با نرخ یادگیری کوچکتر برابر با 1e-5تا از تخریب وزن‌های آموخته‌شده جلوگیری شود. این فرآیند نیز به مدت ۱۰ دوره تکرار شد و باعث بهینه‌سازی دقیق‌تر ویژگی‌ها شد، به‌ویژه در لایه‌های عمیق شبکه که ویژگی‌های سطح بالا را مدل‌سازی می‌کنند.

پس از اتمام آموزش اولیه، مدل بر روی مجموعه داده‌ی اعتبارسنجی (Validation) ارزیابی شد. نتایج به‌دست‌آمده بسیار خوب بودند. این نتایج نشان می‌دهند که مدل توانسته است با دقت بسیار بالا شش نوع عیب سطحی را از یکدیگر تفکیک کند و احتمال

خطا بسیار پایین بوده است.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Bargozarie ResNet-50 bedune laye khuruji
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Frezze kardane hame layeha
for layer in base_model.layers:
    layer.trainable = False

# Afzudane head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(6, activation='softmax')(x)  # 6 کلاس


# Model nahayi
model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Amuzeshe head
history1 = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,
    verbose=2
)
```

```
Epoch 1/10
45/45 - 37s - 828ms/step - accuracy: 0.6736 - loss: 0.9989 - val_accuracy: 0.9556 - val_loss: 0.1542
Epoch 2/10
45/45 - 18s - 403ms/step - accuracy: 0.9319 - loss: 0.2185 - val_accuracy: 0.9833 - val_loss: 0.0950
Epoch 3/10
45/45 - 17s - 387ms/step - accuracy: 0.9514 - loss: 0.1444 - val_accuracy: 0.9667 - val_loss: 0.0819
Epoch 4/10
```

```
45/45 - 18s - 389ms/step - accuracy: 0.9625 - loss: 0.1123 - val_accuracy: 0.9778 - val_loss: 0.0589
Epoch 5/10
45/45 - 18s - 393ms/step - accuracy: 0.9778 - loss: 0.0775 - val_accuracy: 0.9861 - val_loss: 0.0401
Epoch 6/10
45/45 - 18s - 389ms/step - accuracy: 0.9701 - loss: 0.0915 - val_accuracy: 0.9806 - val_loss: 0.0536
Epoch 7/10
45/45 - 20s - 454ms/step - accuracy: 0.9764 - loss: 0.0684 - val_accuracy: 0.9806 - val_loss: 0.0437
Epoch 8/10
45/45 - 18s - 403ms/step - accuracy: 0.9799 - loss: 0.0656 - val_accuracy: 0.9889 - val_loss: 0.0276
Epoch 9/10
45/45 - 17s - 385ms/step - accuracy: 0.9764 - loss: 0.0762 - val_accuracy: 0.9917 - val_loss: 0.0223
Epoch 10/10
45/45 - 18s - 405ms/step - accuracy: 0.9826 - loss: 0.0518 - val_accuracy: 0.9944 - val_loss: 0.0263


# Baz kardane freeze barkhi az layeha baraye fine-tunning
for layer in base_model.layers[-30:]:  # Akharin 30 laye
    layer.trainable = True

# Learning rate kamtar
model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Amuzeshe Fine-Tuning
history2 = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,
    verbose=2
)

Epoch 1/10
45/45 - 51s - 1s/step - accuracy: 0.9924 - loss: 0.0244 - val_accuracy: 0.9944 - val_loss: 0.0100
Epoch 2/10
45/45 - 29s - 640ms/step - accuracy: 0.9958 - loss: 0.0139 - val_accuracy: 0.9972 - val_loss: 0.0066
Epoch 3/10
45/45 - 18s - 397ms/step - accuracy: 0.9972 - loss: 0.0081 - val_accuracy: 1.0000 - val_loss: 0.0051
Epoch 4/10
45/45 - 18s - 411ms/step - accuracy: 0.9958 - loss: 0.0134 - val_accuracy: 1.0000 - val_loss: 0.0054
Epoch 5/10
45/45 - 18s - 399ms/step - accuracy: 0.9979 - loss: 0.0138 - val_accuracy: 0.9972 - val_loss: 0.0057
Epoch 6/10
45/45 - 20s - 449ms/step - accuracy: 0.9993 - loss: 0.0065 - val_accuracy: 0.9972 - val_loss: 0.0056
Epoch 7/10
```

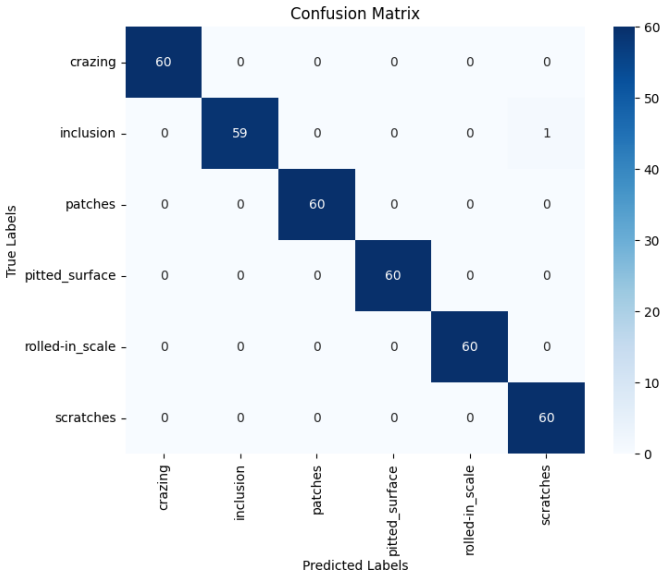
45/45 - 19s - 412ms/step - accuracy: 0.9951 - loss: 0.0165 - val_accuracy: 0.9972 - val_loss: 0.0049
Epoch 8/10
45/45 - 18s - 396ms/step - accuracy: 0.9951 - loss: 0.0155 - val_accuracy: 1.0000 - val_loss: 0.0047
Epoch 9/10
45/45 - 21s - 466ms/step - accuracy: 0.9979 - loss: 0.0075 - val_accuracy: 0.9972 - val_loss: 0.0057
Epoch 10/10
45/45 - 19s - 419ms/step - accuracy: 0.9986 - loss: 0.0049 - val_accuracy: 0.9972 - val_loss: 0.0050

ارزیابی نهایی

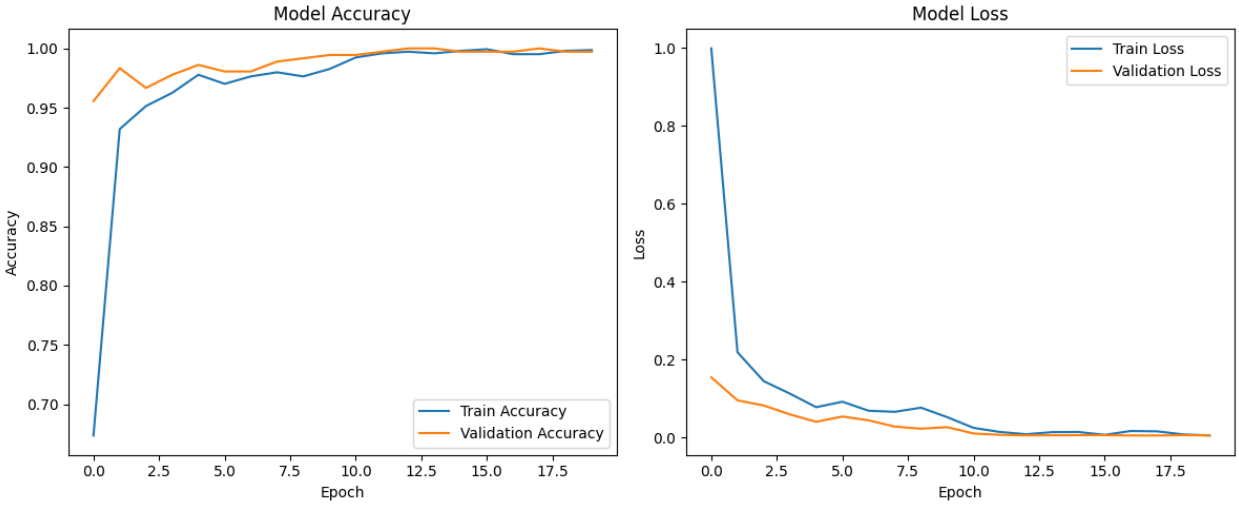
در این بخش از کدهای مشابه استفاده شده است. نتایج بصورت زیر می‌باشند:

```
[ ] # Arzyabi
    loss, accuracy = model.evaluate(val_generator, verbose=1)
    print(f"Deghate nahayi: {accuracy:.4f}")
    print(f"Tabe khata (Loss): {loss:.4f}")
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 145ms/step - accuracy: 0.9959 - loss: 0.0067
Deghate nahayi: 0.9972
Tabe khata (Loss): 0.0050
```



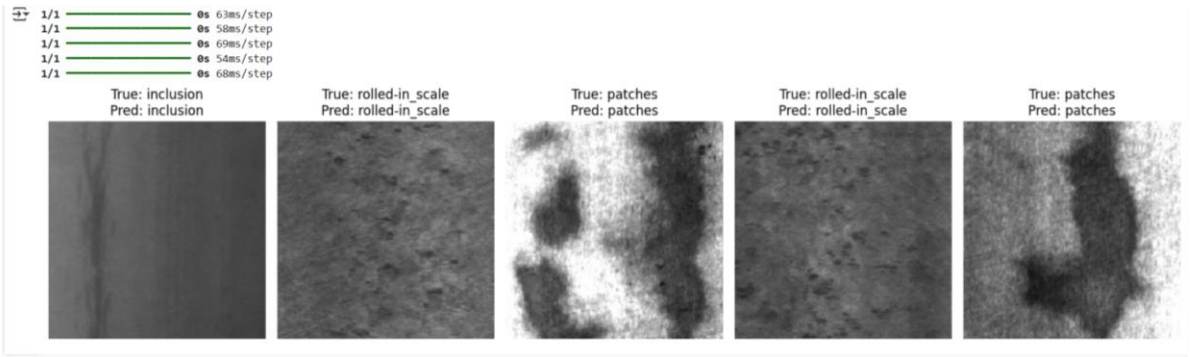
	precision	recall	f1-score	support
crazing	1.00	1.00	1.00	60
inclusion	1.00	0.98	0.99	60
patches	1.00	1.00	1.00	60
pitted_surface	1.00	1.00	1.00	60
rolled-in_scale	1.00	1.00	1.00	60
scratches	0.98	1.00	0.99	60
accuracy			1.00	360
macro avg	1.00	1.00	1.00	360
weighted avg	1.00	1.00	1.00	360



همچنین برای مشاهده عملکرد مدل از کد زیر استفاده شده است. مشاهده می‌شود مدل به خوبی عمل میکند.

```
# Check kardane amalkard
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras.preprocessing import image

# classes
class_names = list(val_generator.class_indices.keys())

import matplotlib.pyplot as plt
import numpy as np
import os
import random
from tensorflow.keras.preprocessing import image

def predict_and_plot_random_images(model, val_dir, class_names, n=5):
    plt.figure(figsize=(15, 5))

    for i in range(n):
        # Entekhabe class va tasvire tasadofi
        class_name = random.choice(class_names)
        class_path = os.path.join(val_dir, class_name)
        img_name = random.choice(os.listdir(class_path))
        img_path = os.path.join(class_path, img_name)

        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array_exp = np.expand_dims(img_array, axis=0)
        img_array_exp = preprocess_input(img_array_exp)
```

```
        preds = model.predict(img_array_exp)
        pred_class = class_names[np.argmax(preds)]

        plt.subplot(1, n, i + 1)
        plt.imshow(np.array(img).astype("uint8"))
        plt.title(f"True: {true_class}\nPred: {pred_class}", fontsize=12)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

predict_and_plot_random_images(model, val_dir, class_names)
```



مقایسه و تحلیل

از نظر دقت نهایی، مدل مبتنی بر یادگیری انتقالی عملکرد بهتری داشته و دقتی در حدود ۹۹٫۷ درصد به دست آورده است. این در حالی است که مدل CNN طراحی‌شده از ابتدا، با وجود داشتن معماری مناسب و تنظیمات دقیق، دقتی کمتر داشت. این تفاوت نشان می‌دهد که بهره‌گیری از ویژگی‌های از پیش آموخته‌شده توسط شبکه‌های عمیق و قدرتمندی مانند ResNet-50 می‌تواند توانایی مدل را در استخراج ویژگی‌های پیچیده افزایش داده و در نتیجه دقت طبقه‌بندی را به طور چشمگیری ارتقاء دهد.

از لحاظ زمان آموزش، مدل انتقال یادگیری نیاز به زمان بیشتری دارد. فرآیند آموزش آن شامل دو مرحله است: ابتدا آموزش فقط لایه‌های جدید (head) و سپس اجرای fine-tuning روی لایه‌های انتهایی شبکه. این روند علاوه بر زمان بیشتر، به منابع محاسباتی قوی‌تری مانند GPU نیز نیاز دارد. در مقابل، مدل CNN طراحی‌شده از ابتدا، معماری ساده‌تری دارد و با تعداد لایه‌ها و پارامترهای کمتر، در زمان کوتاه‌تری آموزش می‌بیند و اجرای آن حتی با منابع سخت‌افزاری محدود نیز امکان‌پذیر است.

در خصوص اندازه و پیچیدگی مدل نیز مدل ResNet-50شامل ده‌ها لایه و میلیون‌ها پارامتر است، در حالی که مدل CNN دارای چند لایه محدود و ساختاری ساده‌تر می‌باشد. این موضوع موجب می‌شود مدل طراحی‌شده از ابتدا سبک‌تر باشد و در کاربردهای عملی که به حافظه و توان پردازشی بالا دسترسی نیست، مناسب‌تر جلوه کند.

در نهایت، یادگیری انتقالی در شرایطی که داده‌ها محدود هستند یا دقت بسیار بالا مورد نیاز است، یک گزینه ایده‌آل محسوب

می‌شود، چرا که از دانشی که شبکه در مواجهه با مجموعه‌داده‌های عظیمی مانند ImageNet کسب کرده، بهره می‌برد. اما در مقابل، آموزش مدل از ابتدا مناسب پروژه‌هایی است که محدودیت منابع وجود دارد یا سادگی و سرعت در اولویت قرار دارد، هرچند ممکن است دقت آن کمی پایین‌تر باشد. در جمع‌بندی می‌توان گفت اگر هدف اصلی، رسیدن به بیشترین دقت ممکن در تشخیص عیوب سطحی باشد و محدودیتی از نظر منابع سخت‌افزاری و زمان وجود نداشته باشد، استفاده از یادگیری انتقالی مبتنی بر ResNet-50به‌طور واضح انتخاب بهتری است. اما اگر سادگی، سرعت و اجرای سبک‌تر مدنظر باشد، مدل طراحی‌شده از ابتدا می‌تواند گزینه‌ای قابل اتکا باشد.

****لازم به ذکر است گزارش و کد در گیت هاب نیز ارائه شده است. لینک آن در زیر آمده است****

https://github.com/maedehesmz8010/HW4_810602161_Esmaeilzadeh