



تمرین چهارم هوش مصنوعی

آرین محمدخانی - ۸۱۰۶۰۳۱۳۶



(شبکه‌های عصبی)

۱. مقدمه

در این تمرین که هدف آن آشنایی با ساختار شبکه پرسپترون چندلایه (MLP) و شبکه پیچشی عمیق (CNN) و مقایسه عملکرد آن‌ها در کاربردهای طبقه‌بندی چندکلاسه است، تمامی مراحل خواسته شده مطابق با صورت سوال انجام شده است. این تمرین به دلیل انجام آزمون و خطاهای متعدد، به ویژه روی شبکه CNN، طی یک هفته زمان بر بوده است. ساختارهای مختلف شبکه نتایج متفاوتی ارائه داده‌اند که در این گزارش آورده شده تا دلایل تفاوت نتایج بهتر درک شود. فایل‌های کد به دو فرمت py و ipynb موجود بوده و تمامی نتایج داخل فایل hw4.ipynb قرار دارد. با توجه به اینکه بررسی‌ها با تنظیمات مختلف هایپارامترها چندین بار انجام شده، کد ارائه شده مربوط به آخرین تلاش است، اما نتایج کامل در گزارش درج شده‌اند. نتایجی که بیش از حد طولانی بوده‌اند، مانند مراحل آموزش مدل‌ها، به صورت فشرده آورده شده‌اند، ولی تمامی تصاویر و نمودارهای کامل نمایش داده شده‌اند. همچنین برای کسب امتیاز بیشتر، فایل کامل پروژه در گیت‌هاب با آدرس زیر نیز آپلود شده است.

https://github.com/ArianAZH/AI_HW4_CNN_MLP

۲. بخش اول (دادگان HAR U)

۱/۲. معرفی روش‌های نرمال سازی و آماده سازی داده‌ها

در شبکه‌های عصبی، به‌ویژه در مدل‌های MLP، نرمال‌سازی داده‌ها یک گام اساسی در پیش‌پردازش محسوب می‌شود که تأثیر چشمگیری بر عملکرد و پایداری مدل دارد. در MLP، هر نورون به تمام نورون‌های لایه قبلی متصل است و وزن‌ها به صورت تصادفی مقداردهی اولیه می‌شوند. اگر ویژگی‌های ورودی مقیاس‌های بسیار متفاوتی داشته باشند، این تفاوت باعث می‌شود یادگیری وزن‌ها سخت‌تر شود، به‌ویژه در مراحل اولیه آموزش، که شبکه به شدت به مقیاس داده‌ها حساس است. بنابراین، نرمال‌سازی ویژگی‌ها باعث می‌شود تمام ورودی‌ها در محدوده مشابهی قرار گیرند و گرادینان‌ها به‌طور مؤثرتری منتقل شوند.

برای مدل‌های MLP، روش‌هایی مانند Z-score (استانداردسازی) که مقادیر ورودی را با میانگین صفر و انحراف معیار یک نرمال می‌کنند، معمولاً انتخاب اول هستند، چون این مدل‌ها به توزیع

آماري داده‌ها وابسته‌اند. همچنين در مواردی که داده‌ها دارای پرت (outlier) باشند، استفاده از Robust Scaling که از میانه و دامنه بین چارک‌ها (IQR) استفاده می‌کند، می‌تواند باعث پایداری بیشتر شود. در مقابل، اگر داده‌ها در یک بازه مشخص مثل [۰, ۲۵۵] باشند (مثلاً تصاویر)، استفاده از Min-Max Scaling نیز می‌تواند مؤثر باشد، چون باعث می‌شود تمام ویژگی‌ها در بازه‌ی [۰, ۱] یا [-۱, ۱] قرار بگیرند که با تابع فعال‌سازهایی مثل Sigmoid یا Tanh سازگارتر است. در MLP، برخلاف شبکه‌های CNN یا RNN، به علت اتصال کامل نورون‌ها و عدم استفاده از فیلترهای مکانی یا زمانی، اهمیت نرمال‌سازی اولیه بیشتر است و اغلب در کنار آن از Batch Normalization نیز در لایه‌های پنهان استفاده می‌شود تا پایداری بیشتری در جریان یادگیری حاصل شود.

جدول 1، روش‌های نرمال سازی

روش نرمال‌سازی	کاربرد معمول	معایب	مزایا
Min-Max Scaling	تصاویر، داده‌های بین ۰ و ۲۵۵	حساس به داده‌های پرت	سادگی، نگاشت به بازه مشخص
Z-score (Standardization)	داده‌های عددی مهندسی‌شده	نیاز به محاسبه میانگین و انحراف معیار	میانگین صفر، مقاوم‌تر نسبت به MinMax
Robust Scaling	داده‌های با outlier زیاد	از دست دادن برخی اطلاعات آماری	مقاوم به داده‌های پرت
Log Scaling	داده‌های skewed مالی/سنسور	فقط برای داده‌های مثبت مناسب	کاهش تأثیر مقادیر بزرگ
Unit Vector Scaling	NLP، embedding	در داده‌های بزرگ ممکن است بی‌ثبات شود	حفظ جهت بردارها
Batch Normalization	بین لایه‌های شبکه	فقط در حین آموزش استفاده می‌شود	سرعت یادگیری بیشتر، پایداری شبکه
Layer Normalization	مدل‌های ترتیبی و NLP	در بعضی موارد کندتر از BatchNorm	مناسب برای RNN و Transformer

اما شبکه‌های عصبی پیچشی عمدتاً برای پردازش داده‌های تصویری طراحی شده‌اند و بنابراین نیاز به نرمال‌سازی خاص‌تری دارند که با ماهیت پیکسلی و کانالی تصاویر هماهنگ باشد. نرمال‌سازی صحیح نه تنها باعث تسریع در آموزش و بهبود دقت مدل می‌شود، بلکه در مدل‌های از پیش آموزش‌دیده (Transfer Learning) نیز یک ضرورت محسوب می‌شود. در ادامه، مهم‌ترین روش‌های نرمال‌سازی برای داده‌های تصویری در CNN آورده شده است.

جدول 2، روش‌های نرمال‌سازی

روش نرمال‌سازی	توضیحات اصلی	کاربردها و مزایا
Image Normalization to [0,1]	تقسیم مقدار پیکسل‌ها بر ۲۵۵ برای رساندن آن‌ها به بازه [0,1]	ساده، سریع، مناسب برای اکثر مدل‌ها با ReLU؛ برای تصاویر خاکستری یا RGB اولیه
Z-score Normalization (mean/std)	نرمال‌سازی کل تصویر با میانگین و انحراف معیار داده‌ها	رایج در مدل‌های از پیش آموزش‌دیده مانند VGG، ResNet؛ به همگونی نوردهی کمک می‌کند
Per-channel Normalization	نرمال‌سازی جداگانه برای هر کانال (R, G, B) با میانگین و std مختص آن کانال	بهبود تعادل رنگ‌ها؛ دقت بیشتر برای مدل‌های RGB یا رنگی؛ در داده‌های طبیعی مفید است

در این پروژه، با هدف بهبود عملکرد مدل‌های یادگیری ماشین، پس از بارگذاری داده‌ها از فایل، از روش StandardScaler برای نرمال‌سازی استفاده شد. این روش به دلیل عملکرد مناسب در داده‌هایی با توزیع نزدیک به نرمال و تأثیر مثبت آن بر پایداری و سرعت همگرایی مدل، انتخاب گردید. پس از نرمال‌سازی، دیتاست آموزش UCI HAR با نسبت ۸۵٪ برای آموزش و ۱۵٪ برای آزمون تقسیم شد. هر نمونه در این مجموعه شامل ۵۶۱ ویژگی عددی می‌باشد. این تقسیم‌بندی برای آموزش و ارزیابی اولیه شبکه‌های عصبی صورت گرفته است. اما برای بررسی نهایی عملکرد مدل و مقایسه با نتایج واقعی، از داده‌های Validation موجود در پوشه جداگانه‌ای نیز استفاده شده است. جهت ارزیابی بهتر، دو سناریو بررسی گردید: استفاده از داده‌های آزمون جداشده برای تست پس از آموزش، و استفاده مستقیم از داده‌های Validation برای ارزیابی مدل در هر مرحله از یادگیری ماشین. در نهایت، نتایج به‌دست‌آمده از این دو رویکرد با یکدیگر مقایسه شده‌اند که تفاوت‌های قابل توجهی در عملکرد مدل مشاهده شد و در ادامه گزارش به آن پرداخته می‌شود.

```

folder_path1 = "UCI HAR Dataset/UCI HAR Dataset"
Xtrain1 = np.loadtxt(f"{folder_path1}/train/X_train.txt")
ytrain1 = np.loadtxt(f"{folder_path1}/train/y_train.txt")
X_val1 = np.loadtxt(f"{folder_path1}/test/X_test.txt")
y_val1 = np.loadtxt(f"{folder_path1}/test/y_test.txt")
ytrain1 = ytrain1.astype(int) - 1
y_val1 = y_val1.astype(int) - 1
scaler = StandardScaler()
Xtrain1 = scaler.fit_transform(Xtrain1)
X_val1 = scaler.transform(X_val1)
X_train1, X_test1, y_train1, y_test1 = train_test_split(Xtrain1, ytrain1, test_size=0.15,
random_state=42)
print("Train shape:", X_train1.shape)
print("Test shape:", X_test1.shape)

```

۲/۲. طراحی شبکه

مدل MLP شامل دو لایه‌ی پنهان با ۱۲۸ و ۶۴ نورون و تابع فعال‌سازی ReLU طراحی شده است. از آن‌جایی که ReLU به دلیل سادگی محاسباتی و جلوگیری از مشکل ناپدید شدن گرادیان در شبکه‌های عمیق عملکرد مناسبی دارد، برای لایه‌های پنهان انتخاب شده است. همچنین از Dropout با نرخ ۰.۵ استفاده شد تا از بیش‌برازش (Overfitting) جلوگیری شود. در نهایت، لایه‌ی خروجی با شش نورون و تابع Softmax برای طبقه‌بندی چندکلاسه در نظر گرفته شد.

```

model1_mlp = Sequential([
    Dense(128, activation='relu',
input_shape=(561,)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')
])

```

مدل CNN شامل سه لایه‌ی کانولوشن با تعداد فیلترهای ۳۲، ۶۴ و ۱۲۸ و کرنل‌های ۳×۳ است. پس از هر لایه‌ی کانولوشن، از نرمال‌سازی دسته‌ای (BatchNormalization) و تابع فعال‌سازی ReLU برای تسریع همگرایی و بهبود پایداری استفاده شده و با یک لایه MaxPooling ابعاد کاهش یافته است. همچنین Dropout پس از لایه‌های دوم و سوم به منظور کنترل بیش‌برازش به کار گرفته شده است. در انتها، ویژگی‌ها به کمک لایه Flatten مسطح شده و وارد یک لایه Dense با ۲۵۶ نورون و سپس لایه‌ی خروجی Softmax با شش نورون شده‌اند. دلایل انتخاب این معماری به شرح زیر می‌باشد.

- ReLU به عنوان تابع فعال‌سازی رایج در لایه‌های پنهان برای جلوگیری از اشباع و حفظ گرادیان مؤثر است.
- BatchNormalization به کاهش وابستگی به مقدار اولیه وزن‌ها و افزایش سرعت آموزش کمک می‌کند.
- Dropout برای جلوگیری از بیش‌برازش ضروری است، به‌ویژه در مدل‌های با ظرفیت بالا.
- MaxPooling به کاهش ابعاد، حفظ ویژگی‌های مهم و جلوگیری از بیش‌برازش کمک می‌کند.

از آنجایی که ورودی‌ها بصورت بردار می‌باشند و شبکه CNN برای پردازش تصویر است، باید این داده‌ها را با دستور reshape تبدیل به تصاویر ساختگی کرد.

کد ۳

```
X_train_cnn = X_train1.reshape(-1, 33, 17, 1)
X_test_cnn = X_test1.reshape(-1, 33, 17, 1)
y_train_cat = to_categorical(y_train1, num_classes=6)
y_test_cat = to_categorical(y_test1, num_classes=6)
X_val_cnn = X_val1.reshape(-1, 33, 17, 1)
y_val_cat = to_categorical(y_val1, num_classes=6)
```

```

model1_cnn = Sequential([
    Conv2D(32, (3, 3), padding='same', input_shape=(33, 17, 1)),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), padding='same'),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(128, (3, 3), padding='same'),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(256),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')])

```

و پس از آن با دستورهای زیر ، با نرخ یادگیری ۰.۰۰۱ و ۳۰ دوره آموزش، دو شبکه MLP و CNN آموزش داده می شود.

کد ۴

```

model1_mlp.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_mlp = model1_mlp.fit(X_train1, y_train1, epochs=30, validation_data=(X_test1, y_test1))

model1_cnn.compile(optimizer=Adam(0.001), loss='categorical_crossentropy', metrics=['accuracy'])
history_cnn = model1_cnn.fit(X_train_cnn, y_train_cat, epochs=30, validation_data=(X_test_cnn, y_test_cat))

```

که دقت اولیه شبکه MLP برابر با ۰.۶۷ بوده که پس از ۳۰ دوره آموزش، به ۰.۹۷۷ روی داده های تست افزایش یافته است. در مقابل، شبکه CNN با دقت اولیه ۰.۷۱ شروع کرده و در پایان آموزش به دقت ۰.۹۹ دست یافته است. همچنین، مقدار Loss در هر دو مدل به طور پیوسته

کاهش یافته و به حدود ۰.۰۶ در مدل MLP و ۰.۰۲ در مدل CNN رسیده است که این کاهش نشان‌دهنده بهبود همگرایی و کاهش خطای مدل‌ها در طول آموزش می‌باشد.

به‌طور کلی، هر دو مدل پس از چند دوره آموزش به دقت بالایی دست یافته‌اند. با این حال، مدل CNN به‌ویژه در مراحل نهایی آموزش، عملکرد بهتری نسبت به مدل MLP از خود نشان داده و دقت بالاتری را ثبت کرده است. این نتایج گویای توانایی بالاتر شبکه‌های پیچشی در استخراج ویژگی‌های مؤثر از داده‌های ساختاریافته نظیر داده‌های حرکتی UCI HAR می‌باشد.

```
Epoch 22/30
196/196 [=====] - 0s 2ms/step - loss: 0.0715 - accuracy: 0.9730 - val_loss: 0.0433 - val_accuracy: 0.9
864
Epoch 23/30
196/196 [=====] - 0s 2ms/step - loss: 0.0784 - accuracy: 0.9704 - val_loss: 0.0376 - val_accuracy: 0.9
855
Epoch 24/30
196/196 [=====] - 0s 2ms/step - loss: 0.0786 - accuracy: 0.9736 - val_loss: 0.0439 - val_accuracy: 0.9
855
Epoch 25/30
196/196 [=====] - 0s 2ms/step - loss: 0.0650 - accuracy: 0.9749 - val_loss: 0.0578 - val_accuracy: 0.9
810
Epoch 26/30
196/196 [=====] - 0s 2ms/step - loss: 0.0585 - accuracy: 0.9813 - val_loss: 0.0519 - val_accuracy: 0.9
846
Epoch 27/30
196/196 [=====] - 0s 2ms/step - loss: 0.0780 - accuracy: 0.9744 - val_loss: 0.0384 - val_accuracy: 0.9
882
Epoch 28/30
196/196 [=====] - 0s 2ms/step - loss: 0.0609 - accuracy: 0.9770 - val_loss: 0.0388 - val_accuracy: 0.9
873
Epoch 29/30
196/196 [=====] - 0s 2ms/step - loss: 0.0555 - accuracy: 0.9816 - val_loss: 0.0460 - val_accuracy: 0.9
882
Epoch 30/30
196/196 [=====] - 0s 2ms/step - loss: 0.0659 - accuracy: 0.9773 - val_loss: 0.0368 - val_accuracy: 0.9
855

196/196 [=====] - 2s 9ms/step - loss: 0.0323 - accuracy: 0.9878 - val_loss: 0.0775 - val_accuracy: 0.9
764
Epoch 23/30
196/196 [=====] - 2s 9ms/step - loss: 0.0357 - accuracy: 0.9864 - val_loss: 0.0374 - val_accuracy: 0.9
864
Epoch 24/30
196/196 [=====] - 2s 9ms/step - loss: 0.0273 - accuracy: 0.9912 - val_loss: 0.0474 - val_accuracy: 0.9
882
Epoch 25/30
196/196 [=====] - 2s 9ms/step - loss: 0.0244 - accuracy: 0.9917 - val_loss: 0.0596 - val_accuracy: 0.9
846
Epoch 26/30
196/196 [=====] - 2s 9ms/step - loss: 0.0285 - accuracy: 0.9901 - val_loss: 0.0931 - val_accuracy: 0.9
692
Epoch 27/30
196/196 [=====] - 2s 9ms/step - loss: 0.0291 - accuracy: 0.9890 - val_loss: 0.0708 - val_accuracy: 0.9
728
Epoch 28/30
196/196 [=====] - 2s 9ms/step - loss: 0.0298 - accuracy: 0.9880 - val_loss: 0.0551 - val_accuracy: 0.9
819
Epoch 29/30
196/196 [=====] - 2s 9ms/step - loss: 0.0307 - accuracy: 0.9901 - val_loss: 0.0509 - val_accuracy: 0.9
846
Epoch 30/30
196/196 [=====] - 2s 9ms/step - loss: 0.0203 - accuracy: 0.9936 - val_loss: 0.0566 - val_accuracy: 0.9
777
```

خروجی ۱

۳/۲. ارزیابی و تحلیل شبکه‌ها

ترسیم نمودار دقت و loss بر حسب دوره به همراه دقت نهایی بر حسب داده های validation و confusion matrix با دستورات زیر بدست آمده است.


```

def plot_history(history, title):
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(title)
    plt.xlabel('Epoch')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

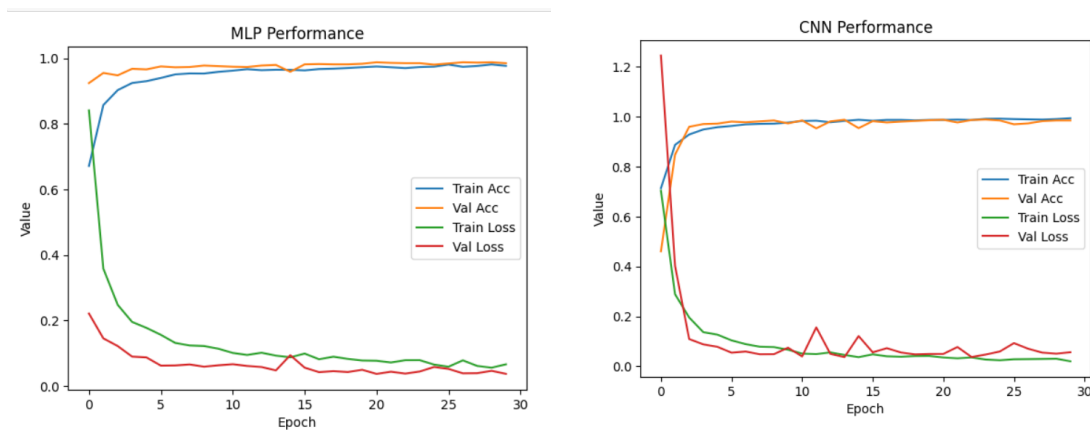
plot_history(history_mlp, "MLP Performance")
plot_history(history_cnn, "CNN Performance")

loss_mlp1, acc_mlp1 = model1_mlp.evaluate(X_val1, y_val1, verbose=0)
print(f"accuracy on mlp test data {acc_mlp1*100:.2f}%")
loss_cnn1, acc_cnn1 = model1_cnn.evaluate(X_test_cnn, y_test_cat, verbose=0)
print(f" accuracy on cnn test data {acc_cnn1*100:.2f}%")

y_pred_mlp = model1_mlp.predict(X_val1).argmax(axis=1)
y_pred_cnn = model1_cnn.predict(X_val_cnn).argmax(axis=1)
cm_mlp = confusion_matrix(y_val1, y_pred_mlp)
cm_cnn = confusion_matrix(y_val1, y_pred_cnn)
ConfusionMatrixDisplay(cm_mlp).plot()
plt.title("MLP Confusion Matrix")
plt.show()
ConfusionMatrixDisplay(cm_cnn).plot()
plt.title("CNN Confusion Matrix")
plt.show()

```

که نمودارهای حاصل در تصاویر زیر نشان داده شده‌اند.



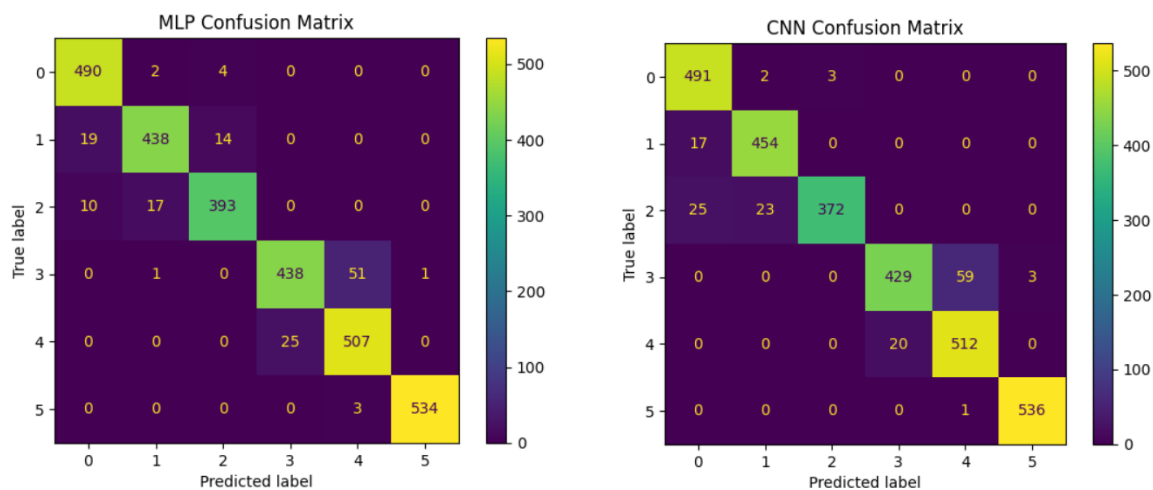
خروجی ۲

و دقت نهایی برای دو مدل در جدول زیر نشان داده شده است.

جدول ۳، دقت‌های شبکه

Accuracy on validation data	
CNN	MLP
98.46%	95.01%

و در انتها confusion matrix برای هر دو مدل برابرست با:



خروجی ۳

با مقایسه نتایج حاصل از ارزیابی دو مدل MLP و CNN، می‌توان به وضوح نتیجه گرفت که مدل CNN عملکرد بهتری نسبت به مدل MLP در این مسئله داشته است. از دلایل این برتری می‌توان به کاهش تعداد پارامترها و جلوگیری از بیش‌برازش اشاره کرد مدل CNN با استفاده از

اشتراک‌گذاری وزن‌ها در لایه‌های کانولوشن و استفاده از pooling، تعداد پارامترها را نسبت به MLP کاهش می‌دهد و در نتیجه خطر overfitting کمتر است.

همانطور که اشاره شد در ابتدا، به دو روش شبکه‌ها آموزش دیده‌اند (با استفاده از داده‌های آموزش تقسیم شده یا آموزش و Validation که در اینجا به روش اول یعنی تقسیم‌بندی داده‌های آموزش و اعتبارسنجی نهایی با داده‌های Validation پرداخته شد. از آنجایی که در این قسمت از تمرین داده‌ها برداری و شبکه‌های آموزش دیده در دسته‌بندی قدرتمند می‌باشند. دقت‌ها در هر دو روش نزدیک به هم و بالا ۰.۹ می‌باشد. از دلایل دقت بالا در این شبکه‌ها می‌توان به استفاده از dropout در هر لایه از شبکه‌ها، BatchNormalization در شبکه CNN و انتخاب مناسب هایپرپارامترهایی همچون تعداد لایه‌های کانولوشن، کرنل‌ها با ابعاد مناسب و همچنین نسبت مناسب نورون‌ها در لایه‌های پنهان MLP می‌باشد که تمامی این انتخاب‌ها با الهام از پیشنهادات مراجع آنلاین می‌باشد.

۳. بخش دوم (دادگان NEU Defects Surface)

در این پروژه، ابتدا تصاویر موجود در دیتاست Defects Surface NEU به صورت سیاه و سفید و با اندازه‌ی 200×200 پیکسل از پوشه‌ی داده‌ها استخراج شدند. سپس، با استفاده از روش Image Normalization، نرمال‌سازی روی داده‌ها اعمال گردید تا مقادیر پیکسل‌ها در بازه‌ی ۰ تا ۱ قرار گیرند و آموزش مدل به شکل مؤثرتری انجام شود. در ادامه، داده‌ها به دو شیوه برای آموزش و ارزیابی مدل تقسیم‌بندی شدند. در روش اول، ۸۵٪ از داده‌ها برای آموزش و ۱۵٪ برای آزمون در نظر گرفته شد. این تقسیم‌بندی در مرحله‌ی آموزش شبکه به کار گرفته شده و در کد نیز پیاده‌سازی گردیده است. در روش دوم، هیچ‌گونه تقسیم داخلی روی داده‌های آموزشی صورت نگرفته و در عوض، از داده‌های validation برای ارزیابی عملکرد مدل در طول آموزش و همچنین تست نهایی استفاده شده است. در کد ارائه‌شده در این پروژه، روش اول پیاده‌سازی شده و مورد تحلیل قرار گرفته است. این روش امکان کنترل بهتر بر روی فرآیند آموزش و آزمون را فراهم کرده است اما در انتها نتیجه روش دوم نیز ارائه داده شده است.

```

def load_images_from_folders(base_dir, image_size=(200, 200)):
    images = []
    labels = []
    class_names = sorted(os.listdir(base_dir))
    class_to_idx = {name: idx for idx, name in enumerate(class_names)}

    for class_name in class_names:
        class_dir = os.path.join(base_dir, class_name)
        for fname in os.listdir(class_dir):
            img_path = os.path.join(class_dir, fname)
            try:
                img = Image.open(img_path).convert("L").resize(image_size)
                img = np.array(img) / 255.0
                images.append(img)
                labels.append(class_to_idx[class_name])
            except Exception as e:
                print(f"error, image not found{img_path}: {e}")

    images = np.array(images).reshape(-1, image_size[0], image_size[1], 1)
    labels = np.array(labels)
    return images, labels, class_names

train_dir = 'NEU-DET Dataset/NEU-DET/train/images'
val_dir = 'NEU-DET Dataset/NEU-DET/validation/images'

Xtrain2, ytrain2, class_names = load_images_from_folders(train_dir)
X_val2, y_val2, _ = load_images_from_folders(val_dir)

X_train2, X_test2, y_train2, y_test2 = train_test_split(Xtrain2, ytrain2, test_size=0.15, random_state=42)

print("Train set:", X_train2.shape, y_train2.shape)
print("Validation set:", X_test2.shape, y_test2.shape)
print("Classes:", class_names)

```

۱,۳. طراحی شبکه

برای مدل MLP، دو شبکه طراحی شده است، که هر دو شبکه با دو لایه‌ی پنهان طراحی شد که از Flatten برای تبدیل تصویر به بردار، نورون‌های Dense، از توابع فعال‌سازی ReLU به همراه Dropout با درصد ۴۰ استفاده شده است اما در یکی از شبکه‌ها از BatchNormalization برای جلوگیری از بیش‌برازش و بهبود پایداری استفاده شده است. در خروجی هر دو نیز از تابع softmax برای طبقه‌بندی شش‌کلاسه استفاده شده است. در انتها با نرخ یادگیری ۰.۰۰۱ مدل‌ها آموزش داده شده است.

کد ۷

```
model2_mlp = Sequential([
    Flatten(input_shape=(200, 200, 1)),

    Dense(256),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.4),

    Dense(128),
    BatchNormalization(),
    Activation('relu'),
    Dense(6, activation='softmax')
])

model2_mlp.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

در این پروژه، سه مدل مختلف شبکه‌ی عصبی پیچشی طراحی و پیاده‌سازی شده‌اند که همگی دارای معماری یکسانی در ساختار کلی خود بوده اما تفاوت آن‌ها در میزان Dropout به‌کاررفته در هر لایه می‌باشد. مدل اول بدون استفاده از Dropout، مدل دوم با استفاده از Dropout با نرخ ۰.۲۵ در هر لایه و مدل سوم با استفاده از Dropout با نرخ ۰.۴۰ در هر لایه که هر سه شبکه از سه لایه‌ی کانولوشن (Conv2D) به همراه Batch Normalization، MaxPooling و تابع فعال‌ساز ReLU بهره می‌برند که به استخراج تدریجی و مؤثر

ویژگی‌های مکانی از تصاویر کمک می‌کند. در انتهای هر معماری، از لایه‌های Fully Connected برای تصمیم‌گیری نهایی استفاده شده است. تمامی مدل‌ها با استفاده از الگوریتم Adam و نرخ یادگیری ۰.۰۰۱ آموزش داده شده‌اند. جهت تشخیص بهتر معماری شبکه جدول زیر ارائه شده است.

جدول 4، ساختار شبکه CNN

توضیحات	Network Stage
تشخیص ویژگی‌های پایه‌ای مانند لبه‌ها	Conv2D (32 filters, 3×3) + ReLU
پایدارسازی فرآیند آموزش	BatchNormalization
کاهش ابعاد مکانی برای کاهش محاسبات	MaxPooling (2×2)
با حذف تصادفی نورون‌ها overfitting کنترل	Dropout (0, 0.25, or 0.40)
استخراج ویژگی‌های پیچیده‌تر	Conv2D (64 filters, 3×3) + ReLU + BN + Pool + Dropout
یادگیری ویژگی‌های سطح بالاتر	Conv2D (128 filters, 3×3) + ReLU + BN + Pool + Dropout
تبدیل ویژگی‌های مکانی به بردار مسطح	Flatten
ترکیب ویژگی‌ها برای طبقه‌بندی	Dense (256 neurons + ReLU + Dropout 0.5)
لایه خروجی برای شناسایی ۶ کلاس عیب سطحی	Dense (6 neurons + Softmax)

در مدل پرسپترون چندلایه که در لایه نرمال سازی می‌شود، در طول ۳۰ دوره آموزش، به تدریج عملکرد بهتری در داده‌های آموزشی نشان داد. دقت مدل از مقدار اولیه حدود ۰.۴۳۶۳ شروع شده و در پایان به حدود ۰.۸۲۱۱ رسید که حاکی از یادگیری مناسب مدل روی داده‌های آموزش است. با این حال، دقت اعتبارسنجی (validation accuracy) نوسانات زیادی داشت و از حدود ۰.۲۱۳۰ در ابتدا تا حداکثر حدود ۰.۴۸۶۱ در دوره ۲۹ افزایش یافت، اما در پایان مجدد به ۰.۳۱۹۴ افت کرد.

مقدار خطای اعتبارسنجی (val_loss) نیز در اکثر دوره‌ها بالا بود و نشان‌دهنده overfitting مدل است؛ یعنی مدل داده‌های آموزش را خوب یاد گرفته ولی در تعمیم به داده‌های جدید

(آزمایشی) عملکرد مطلوبی ندارد. این مسئله به‌ویژه در دوره‌هایی مانند ۱۲ و ۱۹ که مقدار val_loss به‌ترتیب به ۸.۵ و ۱۰.۳ رسید، مشهود است.

در مدل شبکه عصبی کانولوشنی (CNN) در ابتدای آموزش با مقادیر بسیار بالای خطا (val_loss) و دقت پایین در داده‌های آزمایشی مواجه بود (val_accuracy = 0.1528)، اما از دوره‌ی ۲۰ به بعد پیشرفت محسوسی در عملکرد مدل مشاهده شد. به‌طور مشخص، دقت اعتبارسنجی در دوره ۲۳ به ۰.۵۵۵۶ و در دوره ۲۵ به ۰.۶۳۴۳ رسید که بالاترین مقدار در میان تمام دوره‌ها بود. این بهبود ناگهانی در عملکرد احتمالاً به دلیل بهتر شدن استخراج ویژگی‌ها توسط لایه‌های کانولوشنی در مراحل بعدی آموزش است.

```
Epoch 21/30
39/39 [=====] - 27s 704ms/step - loss: 0.3710 - accuracy: 0.8840 - val_loss: 15.9011 - val_accuracy: 0.1481
Epoch 22/30
39/39 [=====] - 28s 706ms/step - loss: 0.2953 - accuracy: 0.9085 - val_loss: 40.7396 - val_accuracy: 0.4537
Epoch 23/30
39/39 [=====] - 28s 709ms/step - loss: 0.4061 - accuracy: 0.8791 - val_loss: 26.6150 - val_accuracy: 0.3981
Epoch 24/30
39/39 [=====] - 28s 709ms/step - loss: 0.2988 - accuracy: 0.9052 - val_loss: 4.2260 - val_accuracy: 0.5787
Epoch 25/30
39/39 [=====] - 28s 716ms/step - loss: 0.3717 - accuracy: 0.8946 - val_loss: 57.6027 - val_accuracy: 0.4028
Epoch 26/30
39/39 [=====] - 28s 710ms/step - loss: 0.2866 - accuracy: 0.9134 - val_loss: 46.8952 - val_accuracy: 0.3426
Epoch 27/30
39/39 [=====] - 28s 707ms/step - loss: 0.2943 - accuracy: 0.9216 - val_loss: 33.1500 - val_accuracy: 0.4074
Epoch 28/30
39/39 [=====] - 28s 705ms/step - loss: 0.5748 - accuracy: 0.8676 - val_loss: 15.4952 - val_accuracy: 0.3750
Epoch 29/30
39/39 [=====] - 28s 705ms/step - loss: 0.4676 - accuracy: 0.8725 - val_loss: 25.8509 - val_accuracy: 0.1759
Epoch 30/30
39/39 [=====] - 27s 705ms/step - loss: 0.3114 - accuracy: 0.9167 - val_loss: 52.5748 - val_accuracy: 0.1620
```

```

Epoch 21/30
39/39 [=====] - 27s 690ms/step - loss: 0.9063 - accuracy: 0.6577 - val_loss: 60.6457 - val_accuracy: 0.2315
Epoch 22/30
39/39 [=====] - 27s 692ms/step - loss: 0.7690 - accuracy: 0.6822 - val_loss: 34.1305 - val_accuracy: 0.3704
Epoch 23/30
39/39 [=====] - 27s 690ms/step - loss: 0.7583 - accuracy: 0.7173 - val_loss: 63.5668 - val_accuracy: 0.2500
Epoch 24/30
39/39 [=====] - 27s 691ms/step - loss: 0.7595 - accuracy: 0.7034 - val_loss: 17.1378 - val_accuracy: 0.5602
Epoch 25/30
39/39 [=====] - 27s 696ms/step - loss: 0.6265 - accuracy: 0.7590 - val_loss: 4.0409 - val_accuracy: 0.6528
Epoch 26/30
39/39 [=====] - 27s 700ms/step - loss: 0.6563 - accuracy: 0.7672 - val_loss: 2.8312 - val_accuracy: 0.6713
Epoch 27/30
39/39 [=====] - 30s 760ms/step - loss: 0.6219 - accuracy: 0.7377 - val_loss: 8.2489 - val_accuracy: 0.6019
Epoch 28/30
39/39 [=====] - 28s 709ms/step - loss: 0.5281 - accuracy: 0.8064 - val_loss: 21.0215 - val_accuracy: 0.5093
Epoch 29/30
39/39 [=====] - 27s 695ms/step - loss: 0.6096 - accuracy: 0.7794 - val_loss: 13.0958 - val_accuracy: 0.5880
Epoch 30/30
39/39 [=====] - 27s 697ms/step - loss: 0.4822 - accuracy: 0.7958 - val_loss: 18.3808 - val_accuracy: 0.4167

```

خروجی ۴

نوسانات شدید در مقادیر Loss نشان می‌دهد که شبکه نیازمند داده‌های آزمون بیشتر یا تعداد دوره‌های بیشتر در مرحله یادگیری شبکه CNN می‌باشد. در روش دوم که یادگیری بر روی داده‌های تست و validation انجام می‌شود از آنجایی که تعداد داده‌ها افزایش می‌یابد، مقدار loss همگرا شده و نوسانات حذف می‌شود.

۲/۳. ارزیابی و تحلیل شبکه‌ها

ترسیم نمودار دقت و loss بر حسب دوره به همراه دقت نهایی بر حسب داده‌های validation و confusion matrix با دستورات زیر بدست آمده است.


```

plot_history(history2_mlp, "MLP Accuracy/Loss")
plot_history(history2_cnn, "CNN Accuracy/Loss")

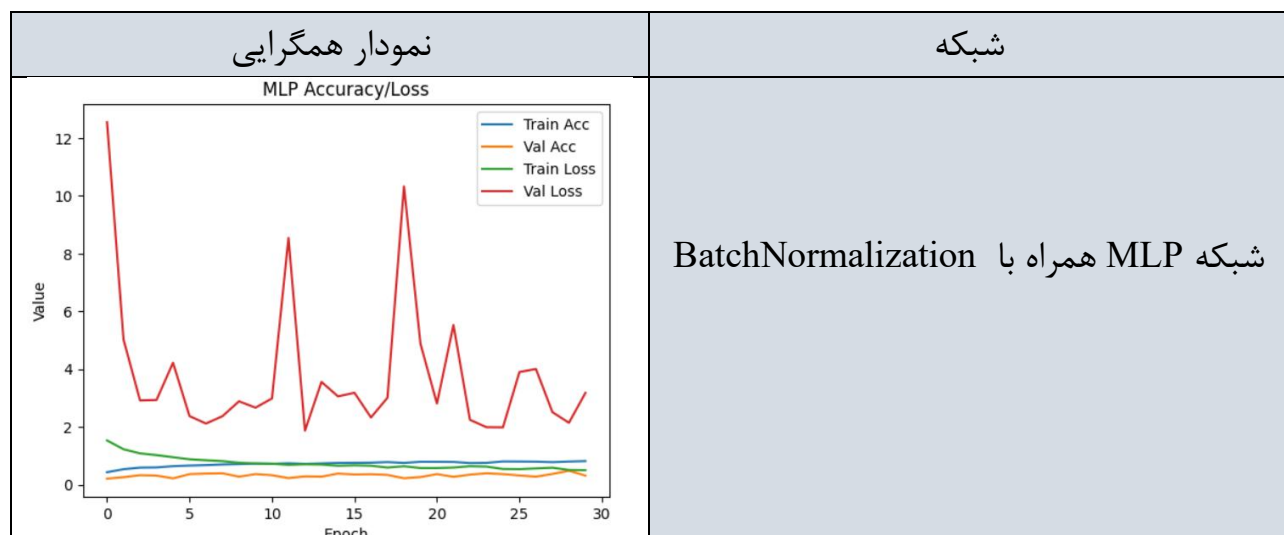
loss_mlp, acc_mlp = model2_mlp.evaluate(X_val2, y_val2, verbose=0)
print(f"accuracy on mlp test data {acc_mlp*100:.2f}%")
loss_cnn, acc_cnn = model2_cnn.evaluate(X_val2, y_val2, verbose=0)
print(f" accuracy on cnn test data {acc_cnn*100:.2f}%")

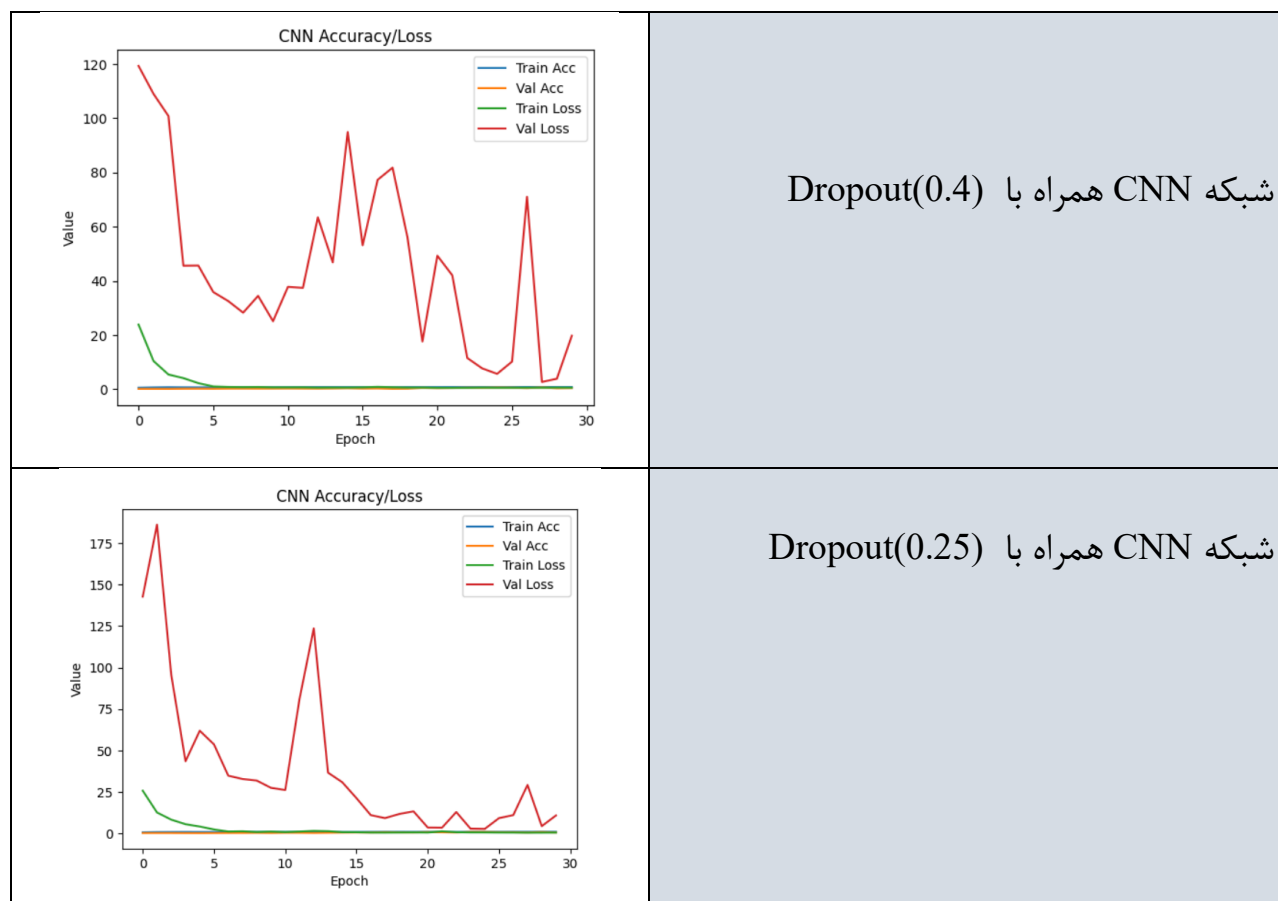
def show_conf_matrix(model, X_test, y_test, title):
    y_pred = model.predict(X_test).argmax(axis=1)
    y_true = y_test
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
    disp.plot(xticks_rotation=45)
    plt.title(title)
    plt.show()
show_conf_matrix(model2_mlp, X_val2, y_val2, "MLP Confusion Matrix")
show_conf_matrix(model2_cnn, X_val2, y_val2, "CNN Confusion Matrix")

```

که نمودارهای حاصل از روش اول در یادگیری شبکه، در تصاویر زیر نشان داده شده‌اند.

جدول ۵، نمودارهای همگرایی در طی آموزش



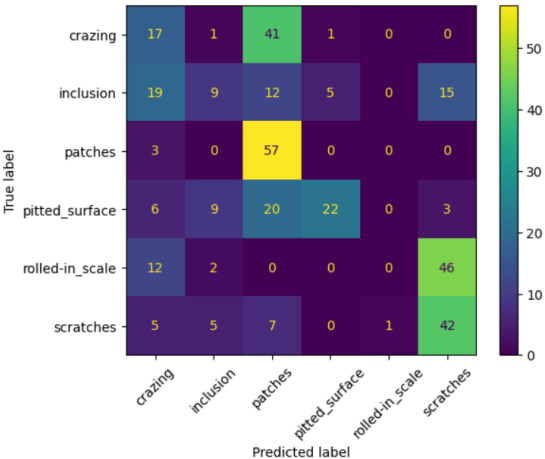
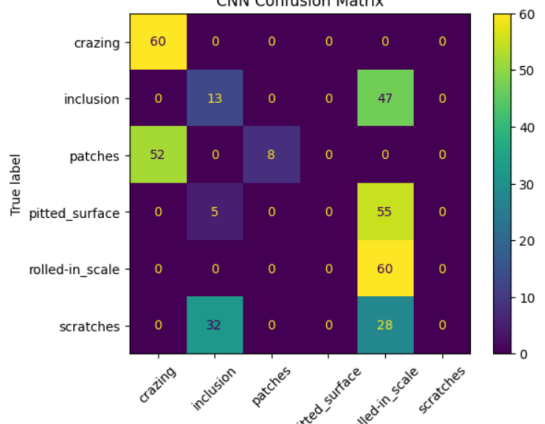


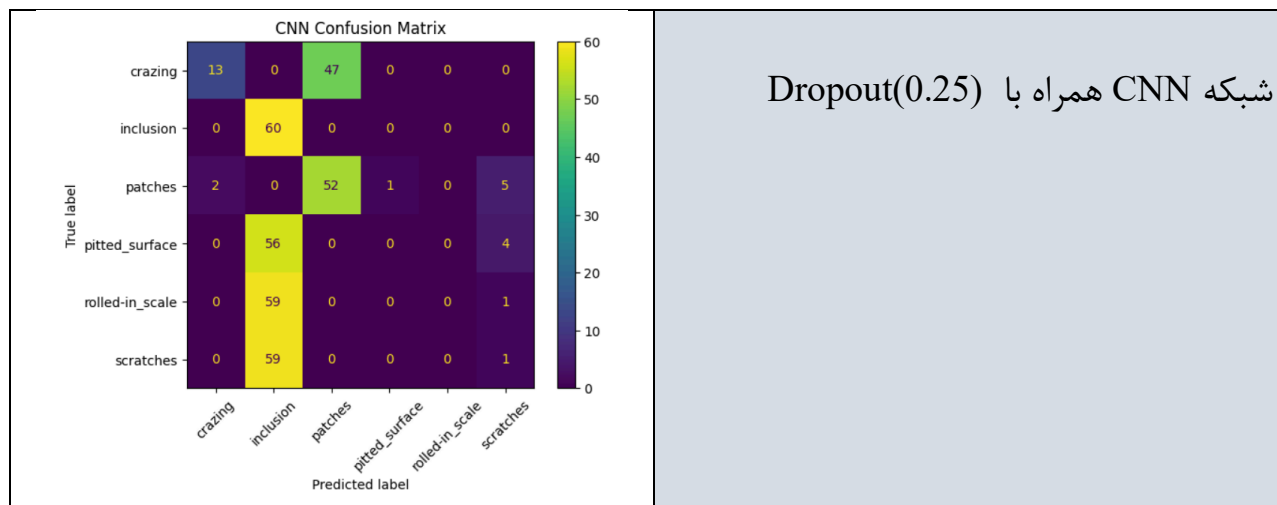
و دقت نهایی برای مدل‌ها در جدول زیر نشان داده شده است.
جدول ۶، دقت نهایی شبکه‌ها

Accuracy on validation data	
حالت اول	
40.83%	شبکه MLP همراه با BatchNormalization
20%	شبکه MLP بدون BatchNormalization
35%	شبکه CNN همراه با Dropout(0.25)
39.17%	شبکه CNN همراه با Dropout(0.4)
60%	شبکه CNN همراه با Dropout(0)
حالت دوم	
17%	شبکه MLP بدون BatchNormalization
83%	شبکه CNN همراه با Dropout(0)

در صورت اضافه کردن یک لایه کانولوشن به شبکه‌های عصبی پیچشی دقت‌ها هم در آموزش و هم در validation افزایش بسزایی دارد، بطور مثال برای شبکه عصبی پیچشی با چهار لایه کانولوشن و Dropout(0) در انتهای دوره آزمون به دقتی حدود ۸۰ درصد می‌رسد. و در انتها confusion matrix برای هر مدل در جدول زیر نمایش داده شده است.

جدول ۷، confusion matrix

confusion matrix	شبکه																																																	
<p>MLP Confusion Matrix</p>  <table><tr><th>True label \ Predicted label</th><th>crazing</th><th>inclusion</th><th>patches</th><th>pitted_surface</th><th>rolled-in_scale</th><th>scratches</th></tr><tr><th>crazing</th><td>17</td><td>1</td><td>41</td><td>1</td><td>0</td><td>0</td></tr><tr><th>inclusion</th><td>19</td><td>9</td><td>12</td><td>5</td><td>0</td><td>15</td></tr><tr><th>patches</th><td>3</td><td>0</td><td>57</td><td>0</td><td>0</td><td>0</td></tr><tr><th>pitted_surface</th><td>6</td><td>9</td><td>20</td><td>22</td><td>0</td><td>3</td></tr><tr><th>rolled-in_scale</th><td>12</td><td>2</td><td>0</td><td>0</td><td>0</td><td>46</td></tr><tr><th>scratches</th><td>5</td><td>5</td><td>7</td><td>0</td><td>1</td><td>42</td></tr></table>	True label \ Predicted label	crazing	inclusion	patches	pitted_surface	rolled-in_scale	scratches	crazing	17	1	41	1	0	0	inclusion	19	9	12	5	0	15	patches	3	0	57	0	0	0	pitted_surface	6	9	20	22	0	3	rolled-in_scale	12	2	0	0	0	46	scratches	5	5	7	0	1	42	شبکه MLP همراه با BatchNormalization
True label \ Predicted label	crazing	inclusion	patches	pitted_surface	rolled-in_scale	scratches																																												
crazing	17	1	41	1	0	0																																												
inclusion	19	9	12	5	0	15																																												
patches	3	0	57	0	0	0																																												
pitted_surface	6	9	20	22	0	3																																												
rolled-in_scale	12	2	0	0	0	46																																												
scratches	5	5	7	0	1	42																																												
<p>CNN Confusion Matrix</p>  <table><tr><th>True label \ Predicted label</th><th>crazing</th><th>inclusion</th><th>patches</th><th>pitted_surface</th><th>rolled-in_scale</th><th>scratches</th></tr><tr><th>crazing</th><td>60</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>inclusion</th><td>0</td><td>13</td><td>0</td><td>0</td><td>47</td><td>0</td></tr><tr><th>patches</th><td>52</td><td>0</td><td>8</td><td>0</td><td>0</td><td>0</td></tr><tr><th>pitted_surface</th><td>0</td><td>5</td><td>0</td><td>0</td><td>55</td><td>0</td></tr><tr><th>rolled-in_scale</th><td>0</td><td>0</td><td>0</td><td>0</td><td>60</td><td>0</td></tr><tr><th>scratches</th><td>0</td><td>32</td><td>0</td><td>0</td><td>28</td><td>0</td></tr></table>	True label \ Predicted label	crazing	inclusion	patches	pitted_surface	rolled-in_scale	scratches	crazing	60	0	0	0	0	0	inclusion	0	13	0	0	47	0	patches	52	0	8	0	0	0	pitted_surface	0	5	0	0	55	0	rolled-in_scale	0	0	0	0	60	0	scratches	0	32	0	0	28	0	شبکه CNN همراه با Dropout(0.4)
True label \ Predicted label	crazing	inclusion	patches	pitted_surface	rolled-in_scale	scratches																																												
crazing	60	0	0	0	0	0																																												
inclusion	0	13	0	0	47	0																																												
patches	52	0	8	0	0	0																																												
pitted_surface	0	5	0	0	55	0																																												
rolled-in_scale	0	0	0	0	60	0																																												
scratches	0	32	0	0	28	0																																												



۳/۳. تغییر های پارامترها

• جایگزینی Dropout با Block Dropout در معماری CNN

Dropout معمولی در هر بار آموزش، برخی نورون‌ها را به صورت تصادفی غیرفعال می‌کند تا از بیش‌برازش جلوگیری شود، اما این روش ممکن است تاثیر نامنظمی روی لایه‌های مختلف داشته باشد و گاهی باعث افت کارایی در برخی شبکه‌ها شود. Block Dropout یا Spatial Dropout نوعی بهبود یافته است که کل فیلترها (کانال‌ها) را به جای نورون‌های مجزا غیرفعال می‌کند. این کار باعث می‌شود که شبکه بهتر بتواند به جای تکیه بر ویژگی‌های خاص یک کانال، به اطلاعات کلی‌تر توجه کند. همچنین باعث می‌شود وابستگی مکانی بین پیکسل‌ها حفظ شود که برای تصاویر اهمیت بالایی دارد. دلیل این جایگزینی، حفظ وابستگی فضایی در ویژگی‌ها، افزایش پایداری آموزش و کاهش overfitting بهتر در تصاویر و کاهش همبستگی بین فیلترها و افزایش تعمیم‌پذیری مدل می‌باشد. برای طراحی این شبکه از دستور Dropout2D استفاده می‌شود و سایر پارامترها مانند ساختار شبکه عصبی پیچشی ساده می‌باشد.

```
model2_cnn2 = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(200, 200, 1)),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),

    Conv2D(64, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),

    Conv2D(128, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')
])
model2_cnn2.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history2_cnn2 = model2_cnn2.fit(X_train2, y_train2, epochs=30, validation_data=(X_test2, y_test2))
```

• مفهوم تجزیه فیلترها (Factorization Kernel)

تجزیه فیلترها یعنی جایگزینی یک فیلتر بزرگ $k \times k$ با دو فیلتر کوچکتر متوالی $k \times 1$ و $1 \times k$ که به ترتیب در دو جهت مختلف اعمال می‌شوند. مزایای این روش، کاهش تعداد پارامترها و محاسبات شبکه، افزایش سرعت آموزش و اجرا و حفظ قدرت استخراج ویژگی‌های مکانی بدون افت دقت.

```

model2_cnn_fact = Sequential([
    Conv2D(32, (3,1), padding='same', activation='relu', input_shape=(200, 200, 1)),
    Conv2D(32, (1,3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),

    Conv2D(64, (3,1), padding='same', activation='relu'),
    Conv2D(64, (1,3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),

    Conv2D(128, (3,1), padding='same', activation='relu'),
    Conv2D(128, (1,3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    SpatialDropout2D(0.4),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')
])

model2_cnn_fact.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history2_cnn_fact = model2_cnn_fact.fit(X_train2, y_train2, epochs=30,
validation_data=(X_test2, y_test2))

```

۴/۳. بررسی و تحلیل نتایج تغییر هایپرپارامترها

ترسیم نمودار دقت و loss بر حسب دوره به همراه دقت نهایی بر حسب داده های validation و confusion matrix با دستورات زیر بدست آمده است.

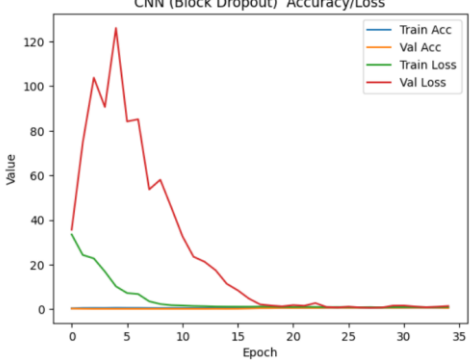
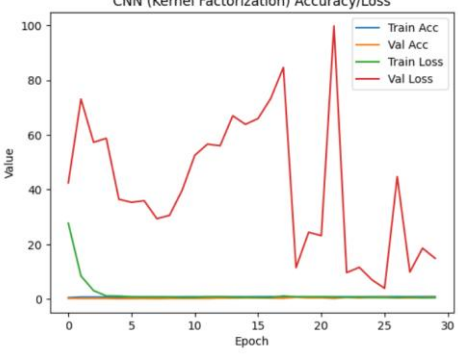
```
plot_history(history2_cnn2, "CNN (Block Dropout) Accuracy/Loss")
plot_history(history2_cnn_fact, "CNN (Kernel Factorization) Accuracy/Loss")

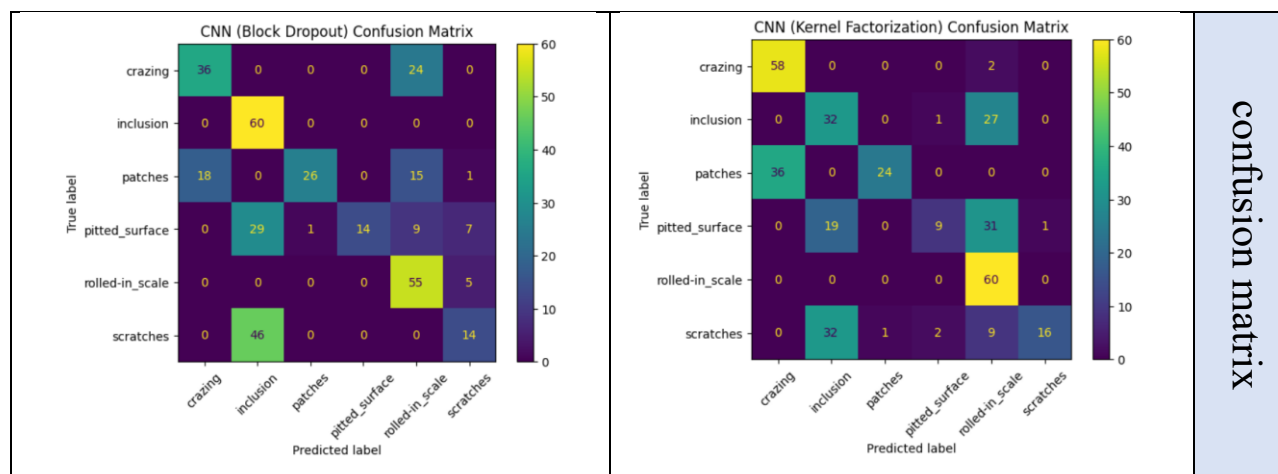
loss_cnn, acc_cnn = model2_cnn2.evaluate(X_val2, y_val2, verbose=0)
print(f" accuracy on cnn (Block Dropout) test data {acc_cnn*100:.2f}%")

loss_cnn, acc_cnn = model2_cnn_fact.evaluate(X_val2, y_val2, verbose=0)
print(f" accuracy on cnn (Kernel Factorization) test data {acc_cnn*100:.2f}%")

show_conf_matrix(model2_cnn2, X_val2, y_val2, "CNN (Block Dropout) Confusion Matrix")
show_conf_matrix(model2_cnn_fact, X_val2, y_val2, "CNN (Kernel Factorization) Confusion Matrix")
```

که نتایج بدست آمده در جدول زیر نمایش داده شده است.
جدول ۸، نتایج تغییر هایپر پارامترها

Block Dropout	Kernel Factorization	
		نمودار آموزش
59.94%	55.28%	دقت



در این مسئله، مدل CNN با استفاده از Block Dropout عملکرد بهتری نسبت به سایر مدل‌ها داشته است. علت این برتری به توانایی Block Dropout در حفظ ساختار مکانی ویژگی‌ها بازمی‌گردد؛ درحالی‌که Dropout معمولی تنها به صورت تصادفی نورون‌ها را حذف می‌کند، Block Dropout بلوک‌هایی از نقشه ویژگی را غیرفعال می‌سازد که منجر به کاهش هم‌وابستگی و بهبود تعمیم‌پذیری مدل می‌شود. همچنین، مدل دارای Kernel Factorization نیز نسبت به مدل پایه بهبود قابل توجهی داشته است؛ زیرا با تجزیه فیلترها، پیچیدگی مدل کاهش یافته و یادگیری ویژگی‌های مؤثرتر امکان‌پذیر شده است. با این حال، نوسانات بیشتر در روند یادگیری این مدل نشان می‌دهد که تنظیمات آن هنوز جای بهبود دارد. در مجموع، مدل Dropout Block با دقت بالاتر و روند آموزش پایدارتر، بهترین عملکرد را در میان مدل‌های بررسی‌شده ارائه داده است.

۴. بخش سوم: یادگیری انتقالی (Transfer learning)

۴/۱. توضیح مراحل آماده‌سازی و آموزش مدل ResNet50

ابتدا تصاویر از پوشه‌های مربوط به هر کلاس خوانده و به سایز 224×224 تبدیل می‌شوند تا با ورودی ResNet50 سازگار باشند. سپس تصاویر به صورت RGB بارگذاری می‌شوند و مقادیر پیکسل‌ها نرمال‌سازی می‌شوند (تقسیم بر ۲۵۵). برای افزایش تعمیم‌پذیری مدل و مقاومت آن نسبت به تغییرات و نویزهای مختلف، از تکنیک‌های داده‌افزایی استفاده می‌شود. چرخش تصاویر در بازه ۲۰ درجه، بزرگ‌نمایی تصادفی تا ۲۰٪ و وارونگی افقی تصادفی، سپس داده‌افزایی به کمک

کلاس ImageDataGenerator انجام شده و مجموعه آموزش به صورت دسته‌ای (batch) به مدل داده می‌شود. همچنین به منظور بررسی کیفیت داده‌ها، به صورت تصادفی از هر کلاس یک نمونه تصویر همراه با برچسب نمایش داده می‌شود.

کد ۱۲

```
train_dir = 'NEU-DET Dataset/NEU-DET/train/images'
val_dir = 'NEU-DET Dataset/NEU-DET/validation/images'

img_size = (224, 224)
batch_size = 32

def load_dataset(base_dir):
    images = []
    labels = []
    class_names = sorted(os.listdir(base_dir))
    class_to_idx = {cls: i for i, cls in enumerate(class_names)}

    for cls in class_names:
        cls_dir = os.path.join(base_dir, cls)
        for fname in os.listdir(cls_dir):
            img_path = os.path.join(cls_dir, fname)
            img = Image.open(img_path).convert('RGB').resize(img_size)
            images.append(np.array(img))
            labels.append(class_to_idx[cls])

    return np.array(images), np.array(labels), class_names

X_train, y_train, class_names = load_dataset(train_dir)
X_val, y_val, _ = load_dataset(val_dir)
print("Train:", X_train.shape, "Val:", X_val.shape)

X_train = X_train / 255.0
X_val = X_val / 255.0

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

val_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

۲/۴. آماده‌سازی مدل ResNet50 با یادگیری انتقالی

مدل پایه ResNet50 با وزن‌های از پیش‌آموزش‌دیده شده روی ImageNet بارگذاری می‌شود؛ لایه‌های انتهایی حذف شده (include_top=False) چون تعداد کلاس‌های هدف متفاوت است. یک لایه GlobalAveragePooling2D برای کاهش ابعاد استخراج ویژگی‌ها اضافه می‌شود. سپس دو لایه جدید به عنوان Head شبکه افزوده می‌شود.

- Dense با ۱۲۸ نورون و تابع فعال‌سازی ReLU برای یادگیری ویژگی‌های پیچیده‌تر
- Dense خروجی با ۶ نورون و تابع Softmax برای طبقه‌بندی ۶ کلاس عیب سطحی

کد ۱۳

```
for i, class_name in enumerate(class_names):
    idx = np.where(y_train == i)[0][0]
    plt.imshow(X_train[idx])
    plt.title(f"Class: {class_name}")
    plt.axis("off")
    plt.show()

base_model = ResNet50(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
```

۳/۴. مرحله آموزش و Fine-tuning

در ابتدا، وزن‌های تمام لایه‌های ResNet50 فریز (غیرفعال برای یادگیری) می‌شوند و فقط لایه‌های جدید Head آموزش داده می‌شوند. این کار باعث می‌شود مدل با داده‌های جدید تطبیق یابد بدون آنکه وزن‌های عمومی استخراج ویژگی‌ها تغییر کنند. پس از آن، برخی از لایه‌های انتهایی مدل پایه (۲۰ لایه آخر) باز می‌شوند و آموزش مدل به صورت مرحله‌ای (Fine-tuning) انجام می‌شود. این کار باعث بهبود یادگیری ویژگی‌های خاص داده‌های جدید و افزایش دقت نهایی می‌شود. نرخ یادگیری در مرحله Fine-tuning بسیار کم انتخاب شده تا تغییرات در وزن‌ها به آرامی و بهینه انجام شود.

```

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

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
output = Dense(6, activation='softmax')(x)

model_resnet = Model(inputs=base_model.input, outputs=output)
model_resnet.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model_resnet.summary()
history_resnet_head = model_resnet.fit(train_generator, validation_data=val_generator, epochs=20)

for layer in base_model.layers[-20:]:
    layer.trainable = True

model_resnet.compile(optimizer=Adam(1e-5), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history_resnet_ft = model_resnet.fit(train_generator, validation_data=val_generator, epochs=20)

```

در تحلیل روند آموزش مدل ResNet50 برای طبقه‌بندی عیوب سطحی، می‌توان مشاهده کرد که مدل اولیه (قبل از آزادسازی لایه‌های انتهایی) عملکرد نسبتاً ضعیفی داشت؛ دقت آموزش و اعتبارسنجی در حد پایین (حداکثر حدود ۳۶٪) باقی ماند و بهبودی محدود در طول دوره‌ها مشاهده شد، که نشان‌دهنده یادگیری ناکارآمد ویژگی‌ها بود. اما پس از آزادسازی ۲۰ لایه آخر مدل و انجام Fine-Tuning با نرخ یادگیری پایین عملکرد مدل به‌طور چشمگیری بهبود یافت. در دوره‌های ابتدایی Fine-Tuning، مدل دچار overfitting شد (افزایش دقت آموزش و کاهش دقت اعتبارسنجی)، ولی پس از چند دوره، با بهینه‌سازی وزن‌ها در لایه‌های عمیق‌تر، مدل توانست به دقت بالایی در اعتبارسنجی برسد؛ به‌طور خاص، از دوره ۱۲ به بعد، دقت اعتبارسنجی تا بیش از ۸۵٪ نیز افزایش یافت و مقدار val_loss نیز به تدریج کاهش یافت. این روند نشان می‌دهد که

Fine-Tuning با استفاده از ResNet50 از پیش آموزش دیده، در شناسایی دقیق عیوب سطحی بسیار مؤثر بوده و مدل در تشخیص ویژگی‌های ظریف‌تر تصاویر عملکرد بهتری پیدا کرده است.

```
Epoch 10/20
45/45 [=====] - 43s 965ms/step - loss: 1.6545 - accuracy: 0.2847 - val_loss: 1.5918 - val_accuracy: 0.2611
Epoch 11/20
45/45 [=====] - 43s 964ms/step - loss: 1.6287 - accuracy: 0.3118 - val_loss: 1.5582 - val_accuracy: 0.4667
Epoch 12/20
45/45 [=====] - 44s 968ms/step - loss: 1.5943 - accuracy: 0.3215 - val_loss: 1.5262 - val_accuracy: 0.4222
Epoch 13/20
45/45 [=====] - 44s 968ms/step - loss: 1.5967 - accuracy: 0.3097 - val_loss: 1.5411 - val_accuracy: 0.4806
Epoch 14/20
45/45 [=====] - 43s 964ms/step - loss: 1.5872 - accuracy: 0.3187 - val_loss: 1.5209 - val_accuracy: 0.3250
Epoch 15/20
45/45 [=====] - 44s 971ms/step - loss: 1.5748 - accuracy: 0.3181 - val_loss: 1.5153 - val_accuracy: 0.3389
Epoch 16/20
45/45 [=====] - 44s 985ms/step - loss: 1.5594 - accuracy: 0.3222 - val_loss: 1.4882 - val_accuracy: 0.3611
Epoch 17/20
45/45 [=====] - 44s 968ms/step - loss: 1.5323 - accuracy: 0.3458 - val_loss: 1.4622 - val_accuracy: 0.3833
Epoch 18/20
45/45 [=====] - 44s 973ms/step - loss: 1.5625 - accuracy: 0.3278 - val_loss: 1.4885 - val_accuracy: 0.3389
Epoch 19/20
45/45 [=====] - 44s 969ms/step - loss: 1.5189 - accuracy: 0.3674 - val_loss: 1.4607 - val_accuracy: 0.3583
Epoch 20/20
45/45 [=====] - 44s 972ms/step - loss: 1.5258 - accuracy: 0.3444 - val_loss: 1.4417 - val_accuracy: 0.4500

Epoch 10/20
45/45 [=====] - 50s 1s/step - loss: 0.5697 - accuracy: 0.8139 - val_loss: 2.3901 - val_accuracy: 0.3306
Epoch 11/20
45/45 [=====] - 50s 1s/step - loss: 0.5444 - accuracy: 0.8236 - val_loss: 1.3097 - val_accuracy: 0.4750
Epoch 12/20
45/45 [=====] - 50s 1s/step - loss: 0.5737 - accuracy: 0.8007 - val_loss: 0.7002 - val_accuracy: 0.7528
Epoch 13/20
45/45 [=====] - 50s 1s/step - loss: 0.5048 - accuracy: 0.8535 - val_loss: 0.8012 - val_accuracy: 0.7333
Epoch 14/20
45/45 [=====] - 50s 1s/step - loss: 0.5014 - accuracy: 0.8382 - val_loss: 0.5198 - val_accuracy: 0.8444
Epoch 15/20
45/45 [=====] - 50s 1s/step - loss: 0.5158 - accuracy: 0.8333 - val_loss: 0.8916 - val_accuracy: 0.7028
Epoch 16/20
45/45 [=====] - 50s 1s/step - loss: 0.4825 - accuracy: 0.8417 - val_loss: 0.4635 - val_accuracy: 0.8361
Epoch 17/20
45/45 [=====] - 50s 1s/step - loss: 0.4695 - accuracy: 0.8479 - val_loss: 0.4624 - val_accuracy: 0.8500
Epoch 18/20
45/45 [=====] - 50s 1s/step - loss: 0.4407 - accuracy: 0.8535 - val_loss: 0.6265 - val_accuracy: 0.7556
Epoch 19/20
45/45 [=====] - 50s 1s/step - loss: 0.4170 - accuracy: 0.8729 - val_loss: 0.5797 - val_accuracy: 0.8556
Epoch 20/20
45/45 [=====] - 50s 1s/step - loss: 0.3936 - accuracy: 0.8639 - val_loss: 0.7903 - val_accuracy: 0.7333
```

خروجی ۵

۴/۴. ارزیابی مدل

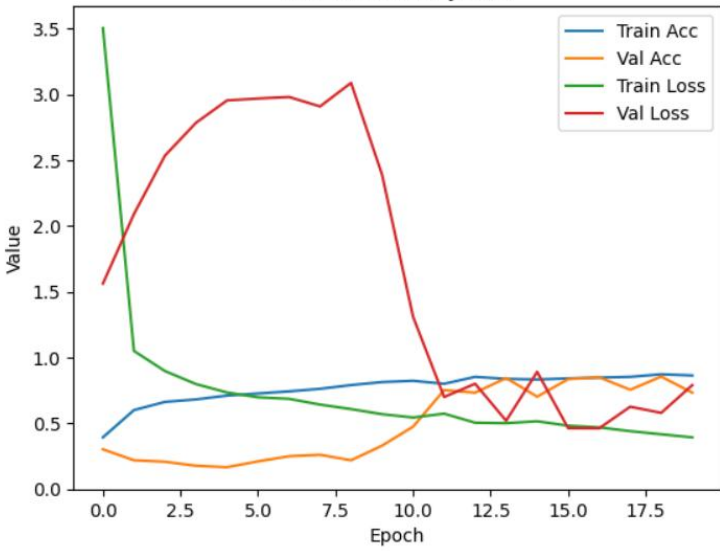
با دستورات زیر مدل ارزیابی می‌شود.

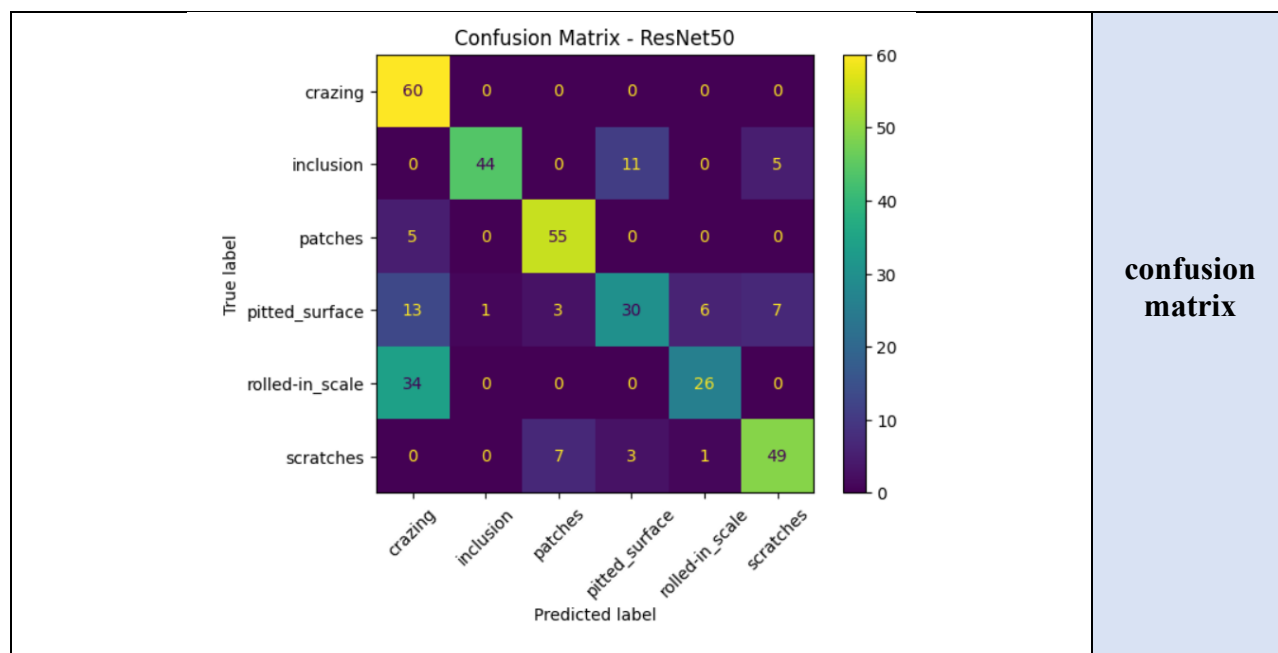
```
plot_history(history_resnet_ft, "Resnet Accuracy/Loss")

y_pred = model_resnet.predict(X_val).argmax(axis=1)
loss, acc = model_resnet.evaluate(X_val, y_val)
print(f"Accuracy: {acc:.4f} - Loss: {loss:.4f}")
cm = confusion_matrix(y_val, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(xticks_rotation=45)
plt.title("Confusion Matrix - ResNet50")
plt.show()

print(classification_report(y_val, y_pred, target_names=class_names))
```

جدول ۹، ارزیابی مدل Resnet50

خروجی‌ها		نمودار
Resnet Accuracy/Loss		
		
73.33%		دقت



مدل ResNet50 نسبت به سایر معماری‌های بررسی‌شده در این پروژه، عملکرد بهتری از خود نشان داده است. دلیل اصلی این برتری، بهره‌گیری از یادگیری انتقالی و معماری عمیق و قدرتمند آن است که از پیش روی دیتاست بزرگی مانند ImageNet آموزش دیده است. این امر موجب می‌شود مدل در ابتدای آموزش، ویژگی‌های عمومی تصاویر (مانند لبه‌ها، بافت‌ها و شکل‌ها) را از قبل شناخته باشد و تنها با تنظیم جزئی (fine-tuning) روی داده‌های خاص پروژه (مانند تصاویر عیوب سطحی)، به دقت بالایی دست یابد. فریز اولیه لایه‌ها و سپس بازکردن تدریجی آن‌ها نیز باعث می‌شود ResNet50 به‌صورت کنترل‌شده دانش عمومی خود را با ویژگی‌های اختصاصی داده‌ها ترکیب کند.

در مقایسه با مدل‌های پایه CNN، نسخه دارای Block Dropout و مدل Kernel Factorization، مدل ResNet50 نه تنها سریع‌تر همگرا شده، بلکه در مقابله با overfitting نیز موفق‌تر عمل کرده است. استفاده از GlobalAveragePooling و Dropout در انتهای مدل نیز به کاهش پارامترها و افزایش تعمیم‌پذیری کمک کرده است. علاوه بر این، با اعمال داده‌افزایی مناسب (مانند چرخش، زوم و آینه‌ای کردن تصاویر)، توانایی مدل برای مواجهه با تنوع داده و نویز افزایش یافته است. همه این عوامل باعث شده‌اند که ResNet50 حتی با تعداد دوره آموزشی

کمر به دقت اعتبارسنجی ۷۳.۳۳ درصد برسد و بهترین مدل در این مسئله تشخیص عیوب سطحی فولاد شناخته شود.