

```

import tensorflow as tf # برای استفاده از قابلیت‌های یادگیری عمیق TensorFlow وارد کردن کتابخانه
from tensorflow.keras.datasets import fashion_mnist # برای استفاده در آموزش مدل Fashion MNIST وارد کردن دیتاست
from tensorflow.keras import layers, models, regularizers # وارد کردن ماژول‌های مختلف برای ساخت مدل و لایه‌ها
import matplotlib.pyplot as plt # برای رسم نمودارها Matplotlib وارد کردن کتابخانه
import numpy as np # برای انجام محاسبات عددی NumPy وارد کردن کتابخانه
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # وارد کردن متریک‌های ارزیابی مدل
import seaborn as sns # برای رسم نمودارهای زیبا و تحلیل داده‌ها Seaborn وارد کردن کتابخانه

```

```

# بارگذاری دیتاست Fashion MNIST
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

```

```

# نمایش شکل داده‌های آموزشی
print("Training Data Shape:", train_images.shape) # نمایش ابعاد تصاویر آموزشی
print("Testing Data Shape:", test_images.shape) # نمایش ابعاد تصاویر آزمایشی

```

```

# نرمال‌سازی تصاویر از بازه [0, 255] به [0.0, 1.0]
train_data_clean = train_images / 255.0 # تقسیم تصاویر آموزشی بر 255 برای نرمال‌سازی
test_data_clean = test_images / 255.0 # تقسیم تصاویر آزمایشی بر 255 برای نرمال‌سازی

```

```

# نمایش برچسب‌های منحصر به فرد در داده‌های آموزشی و آزمایشی
print("Unique Train Labels:", np.unique(train_labels)) # نمایش برچسب‌های منحصر به فرد در داده‌های آموزشی
print("Unique Test Labels:", np.unique(test_labels)) # نمایش برچسب‌های منحصر به فرد در داده‌های آزمایشی

```

```

⚡ Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ----- 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ----- 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ----- 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ----- 0s 0us/step
Training Data Shape: (60000, 28, 28)
Testing Data Shape: (10000, 28, 28)
Unique Train Labels: [0 1 2 3 4 5 6 7 8 9]
Unique Test Labels: [0 1 2 3 4 5 6 7 8 9]

```

```

# تابعی برای نمایش تصاویر نمونه
def display_images(images, labels, num_images=3):
    selected_classes = [0, 1, 2] # انتخاب کلاس‌های خاص برای نمایش

```

```

    # لیست‌های خالی برای ذخیره تصاویر و برچسب‌ها
    selected_images = []
    selected_labels = []

```

```

    # انتخاب یک تصویر از هر کلاس انتخاب شده
    for class_label in selected_classes:
        class_indices = np.where(labels == class_label)[0] # پیدا کردن ایندکس‌های تصاویر مربوط به کلاس
        selected_images.append(images[class_indices[0]]) # انتخاب اولین تصویر از کلاس
        selected_labels.append(labels[class_indices[0]]) # ذخیره برچسب مربوطه

```

```

    # تبدیل لیست‌ها به آرایه
    selected_images = np.array(selected_images)
    selected_labels = np.array(selected_labels)

```

```

    # ایجاد یک شکل برای نمایش تصاویر
    fig, axes = plt.subplots(1, num_images, figsize=(10, 4)) # ایجاد زیرنمودارها
    for i, ax in enumerate(axes):
        ax.imshow(selected_images[i].squeeze(), cmap='gray') # حذف بعد کتال و نمایش تصویر
        ax.set_title(f'Label: {selected_labels[i]}') # نمایش برچسب زیر تصویر
        ax.axis('off') # مخفی کردن محورهای نمودار
    plt.tight_layout() # تنظیم فضای بین زیرنمودارها
    plt.show() # نمایش تصاویر

```

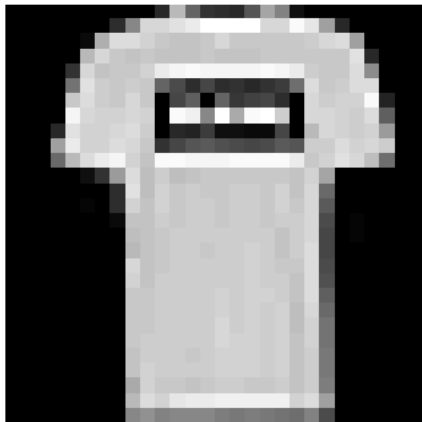
```

# نمایش داده‌های نمونه از دیتاست تمیز شده
display_images(train_data_clean, train_labels, num_images=3) # فراخوانی تابع برای نمایش سه تصویر نمونه

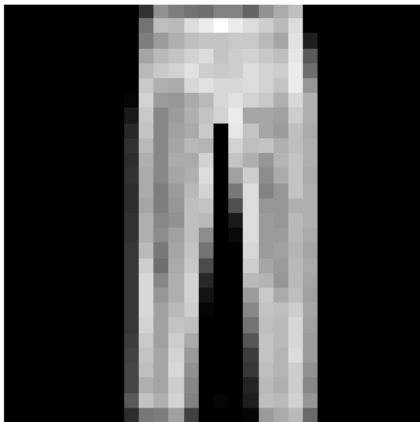
```



Label: 0



Label: 1



Label: 2



```
# تابعی برای افزودن نویز گوسی به تصاویر
def add_gaussian_noise(images, mean=0.0, severity=1):
    severity_levels = [0.08, 0.12, 0.18, 0.26, 0.38] # سطوح شدت نویز
    stddev = severity_levels[severity - 1] # دریافت انحراف معیار بر اساس شدت
    noise = np.random.normal(mean, stddev, images.shape) # تولید نویز گوسی
    noisy_images = images + noise # افزودن نویز به تصاویر
    noisy_images = np.clip(noisy_images, 0.0, 1.0) # اطمینان از اینکه مقادیر پیکسل‌ها در بازه [0.0, 1.0] هستند
    return noisy_images # بازگرداندن تصاویر با نویز
```

```
# افزودن نویز به داده‌های آموزشی و آزمایشی
train_data_noisy = add_gaussian_noise(train_data_clean, severity=5) # افزودن نویز به داده‌های آموزشی
test_data_noisy = add_gaussian_noise(test_data_clean, severity=5) # افزودن نویز به داده‌های آزمایشی
```

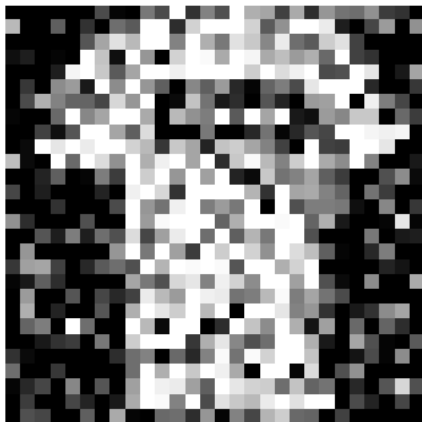
```
# نمایش شکل داده‌های آموزشی و آزمایشی با نویز
print("Noisy Training Data Shape:", train_data_noisy.shape) # نمایش ابعاد داده‌های آموزشی با نویز
print("Noisy Testing Data Shape:", test_data_noisy.shape) # نمایش ابعاد داده‌های آزمایشی با نویز
```

```
# نمایش تصاویر نمونه با نویز
display_images(train_data_noisy, train_labels, num_images=3) # فراخوانی تابع برای نمایش سه تصویر نمونه با نویز
```

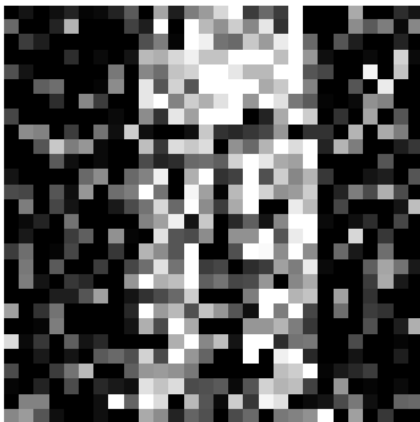


```
Noisy Training Data Shape: (60000, 28, 28)
Noisy Testing Data Shape: (60000, 28, 28)
```

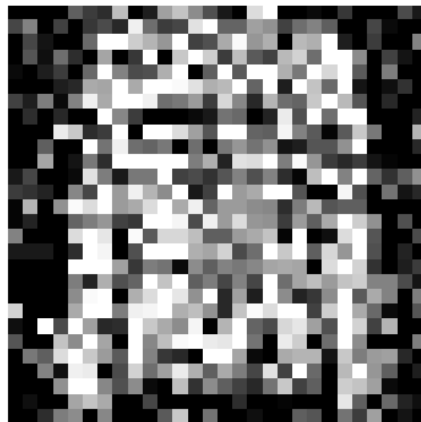
Label: 0



Label: 1



Label: 2



```
# برای تصاویر با اندازه کامل CCNN تابعی برای ایجاد مدل
def create_model():
    model = models.Sequential() # ایجاد یک مدل ترتیبی
```

```
# بلوک ۱
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) # لایه کانولوشن با 32 فیلتر و اندازه (3x3)
model.add(layers.BatchNormalization()) # نرمالسازی بچ
model.add(layers.MaxPooling2D((2, 2), strides=2)) # لایه ماکس پولینگ برای کاهش ابعاد
model.add(tf.keras.layers.Dropout(0.5)) # لایه دراپاوت برای جلوگیری از بیش‌پرازش
```

```
# بلوک ۲
model.add(layers.Conv2D(64, (3, 3), activation='relu')) # لایه کانولوشن با 64 فیلتر
model.add(layers.BatchNormalization()) # نرمالسازی بچ
```

```

# بلوک ۳
model.add(layers.Conv2D(64, (3, 3), activation='relu')) # لایه کفولوشن با 64 فیلتر
model.add(layers.BatchNormalization()) # نرمالسازی بچ

# Flatten و Dense لایه‌های
model.add(layers.Flatten()) # صاف کردن خروجی لایه‌های قبلی
model.add(layers.Dense(128, activation='relu')) # با 128 نرون Dense لایه
model.add(tf.keras.layers.Dropout(0.5)) # لایه دراپ‌اوت
model.add(layers.Dense(10, activation='softmax')) # Fashion MNIST لایه خروجی با 10 کلاس برای دیتاست

model.summary() # نمایش خلاصه‌ای از ساختار مدل

# کامپایل مدل
model.compile(optimizer='adam', # انتخاب بهینه‌ساز
              loss=tf.keras.losses.SparseCategoricalCrossentropy(), # تابع هزینه
              metrics=['accuracy']) # معیار دقت

return model # بازگشت مدل ایجاد شده

# ایجاد مدل تمیز برای دیتاست تمیز
clean_model = create_model() # فراخوانی تابع برای ایجاد مدل

# آموزش مدل با داده‌های تمیز
history_clean = clean_model.fit(train_data_clean, # داده‌های آموزشی
                               train_labels, # برچسب‌های آموزشی
                               epochs=10, # تعداد دوره‌های آموزش
                               batch_size=128, # اندازه بچ
                               validation_data=(test_data_clean, test_labels) # داده‌های اعتبارسنجی
                               )

# ارزیابی بر روی داده‌های آزمایشی تمیز
test_loss_clean, test_acc_clean = clean_model.evaluate(test_data_clean, test_labels, verbose=2) # ارزیابی مدل
print(f"Clean Data - Test Accuracy: {test_acc_clean:.4f}, Test Loss: {test_loss_clean:.4f}") # نمایش دقت و خسارت تست
print(f"Clean Data - Training Accuracy: {history_clean.history['accuracy'][-1]:.4f}") # نمایش دقت آموزش

```

```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
batch_normalization (BatchNormalization)	(None, 26, 26, 32)	128
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 11, 11, 64)	256
conv2d_2 (Conv2D)	(None, 9, 9, 64)	36,928
batch_normalization_2 (BatchNormalization)	(None, 9, 9, 64)	256
flatten (Flatten)	(None, 5184)	0
dense (Dense)	(None, 128)	663,680
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 721,354 (2.75 MB)  
 Trainable params: 721,034 (2.75 MB)  
 Non-trainable params: 320 (1.25 KB)

```

Epoch 1/10
469/469 ----- 127s 249ms/step - accuracy: 0.7231 - loss: 0.8058 - val_accuracy: 0.8390 - val_loss: 0.4223
Epoch 2/10
469/469 ----- 137s 240ms/step - accuracy: 0.8528 - loss: 0.4041 - val_accuracy: 0.8751 - val_loss: 0.3401
Epoch 3/10
469/469 ----- 113s 241ms/step - accuracy: 0.8763 - loss: 0.3471 - val_accuracy: 0.8968 - val_loss: 0.3079
Epoch 4/10
469/469 ----- 141s 240ms/step - accuracy: 0.8899 - loss: 0.3038 - val_accuracy: 0.8900 - val_loss: 0.3091
Epoch 5/10
469/469 ----- 141s 239ms/step - accuracy: 0.8945 - loss: 0.2858 - val_accuracy: 0.8588 - val_loss: 0.4891
Epoch 6/10
469/469 ----- 141s 236ms/step - accuracy: 0.9015 - loss: 0.2718 - val_accuracy: 0.9037 - val_loss: 0.2880
Epoch 7/10
469/469 ----- 143s 237ms/step - accuracy: 0.9065 - loss: 0.2530 - val_accuracy: 0.9114 - val_loss: 0.2520
Epoch 8/10
469/469 ----- 142s 238ms/step - accuracy: 0.9118 - loss: 0.2408 - val_accuracy: 0.9116 - val_loss: 0.2511
Epoch 9/10
469/469 ----- 142s 239ms/step - accuracy: 0.9141 - loss: 0.2341 - val_accuracy: 0.9078 - val_loss: 0.2749
Epoch 10/10
469/469 ----- 111s 236ms/step - accuracy: 0.9199 - loss: 0.2178 - val_accuracy: 0.8760 - val_loss: 0.4043
313/313 - 5s - 16ms/step - accuracy: 0.8760 - loss: 0.4043
Clean Data - Test Accuracy: 0.8760, Test Loss: 0.4043
Clean Data - Training Accuracy: 0.9191

```

# تابعی برای نمایش منحنی‌های آموزش و اعتبارسنجی

```

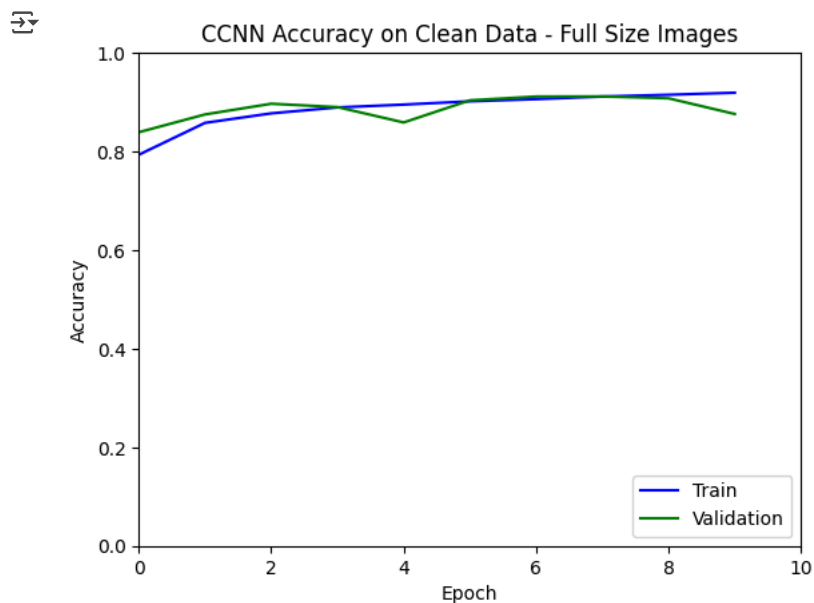
def plot_metrics(history, metric_name, title, ylim=1, xlim=10):
    plt.title(title) # عنوان نمودار
    plt.ylim(0, ylim) # تنظیم حدود محور Y
    plt.xlim(0, xlim) # تنظیم حدود محور X
    plt.plot(history.history[metric_name], color='blue', label='Train') # رسم منحنی دقت آموزش
    plt.plot(history.history['val_' + metric_name], color='green', label='Validation') # رسم منحنی دقت اعتبارسنجی
    plt.legend(loc='lower right') # قرار دادن افسانه در گوشه پایین راست
    plt.xlabel("Epoch") # برچسب محور X
    plt.ylabel("Accuracy") # برچسب محور Y

```

```

plot_metrics(history_clean, "accuracy", "CCNN Accuracy on Clean Data - Full Size Images")

```



```
# ایجاد مدل دیگری برای دیتاست نویزی
noisy_model = create_model() # فراخوانی تابع برای ایجاد مدل

# آموزش مدل با داده‌های نویزی
history_noisy = noisy_model.fit(train_data_noisy, # داده‌های آموزشی نویزی
                                train_labels, # برچسب‌های آموزشی
                                epochs=10, # تعداد دوره‌های آموزش
                                batch_size=128, # اندازه بچ
                                validation_data=(test_data_noisy, test_labels) # داده‌های اعتبارسنجی نویزی
                                )

# ارزیابی بر روی داده‌های آزمایشی نویزی
test_loss_noisy, test_acc_noisy = noisy_model.evaluate(test_data_noisy, test_labels, verbose=2) # ارزیابی مدل
print(f"Noisy Data - Test Accuracy: {test_acc_noisy:.4f}, Test Loss: {test_loss_noisy:.4f}") # نمایش دقت و خسارت تست
print(f"Noisy Data - Training Accuracy: {history_noisy.history['accuracy'][-1]:.4f}") # نمایش دقت آموزش
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_3 (BatchNormalization)	(None, 26, 26, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_2 (Dropout)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18,496
batch_normalization_4 (BatchNormalization)	(None, 11, 11, 64)	256
conv2d_5 (Conv2D)	(None, 9, 9, 64)	36,928
batch_normalization_5 (BatchNormalization)	(None, 9, 9, 64)	256
flatten_1 (Flatten)	(None, 5184)	0
dense_2 (Dense)	(None, 128)	663,680
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 721,354 (2.75 MB)

Trainable params: 721,034 (2.75 MB)

Non-trainable params: 320 (1.25 KB)

Epoch 1/10

469/469 123s 254ms/step - accuracy: 0.6441 - loss: 1.0458 - val\_accuracy: 0.6933 - val\_loss: 0.8460

Epoch 2/10

469/469 137s 244ms/step - accuracy: 0.7729 - loss: 0.6084 - val\_accuracy: 0.8098 - val\_loss: 0.5274

Epoch 3/10

469/469 113s 241ms/step - accuracy: 0.8012 - loss: 0.5361 - val\_accuracy: 0.8226 - val\_loss: 0.4845

Epoch 4/10

469/469 143s 243ms/step - accuracy: 0.8138 - loss: 0.5014 - val\_accuracy: 0.8333 - val\_loss: 0.4668

Epoch 5/10

469/469 116s 247ms/step - accuracy: 0.8221 - loss: 0.4706 - val\_accuracy: 0.8348 - val\_loss: 0.4630

Epoch 6/10

469/469 113s 242ms/step - accuracy: 0.8299 - loss: 0.4484 - val\_accuracy: 0.8327 - val\_loss: 0.4593

Epoch 7/10

469/469 111s 237ms/step - accuracy: 0.8368 - loss: 0.4247 - val\_accuracy: 0.8325 - val\_loss: 0.4650

Epoch 8/10

469/469 142s 237ms/step - accuracy: 0.8450 - loss: 0.4033 - val\_accuracy: 0.8409 - val\_loss: 0.4367

Epoch 9/10

469/469 142s 238ms/step - accuracy: 0.8536 - loss: 0.3812 - val\_accuracy: 0.8409 - val\_loss: 0.4601

Epoch 10/10

469/469 110s 235ms/step - accuracy: 0.8562 - loss: 0.3704 - val\_accuracy: 0.8365 - val\_loss: 0.4810

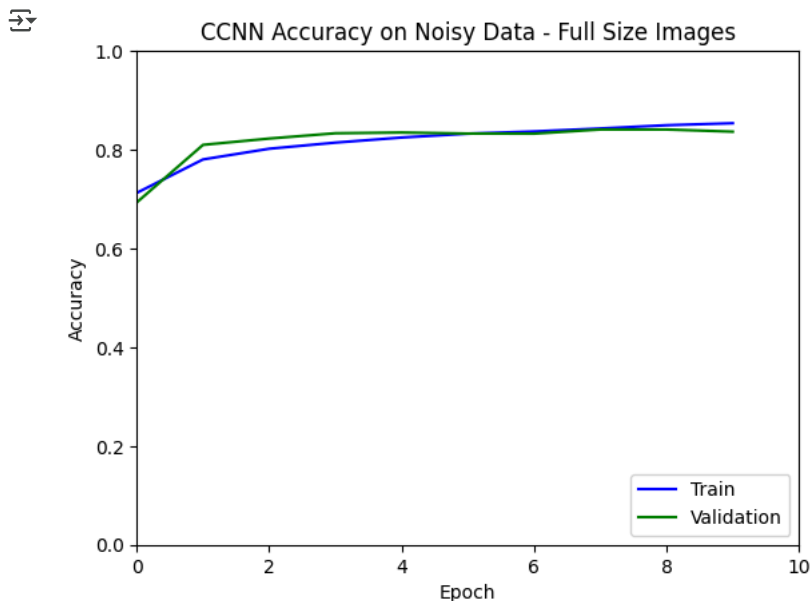
313/313 - 4s - 14ms/step - accuracy: 0.8365 - loss: 0.4810

Noisy Data - Test Accuracy: 0.8365, Test Loss: 0.4810

Noisy Data - Training Accuracy: 0.8536

رسم منحنی دقت مدل بر روی داده‌های نویزی

plot\_metrics(history\_noisy, "accuracy", "CCNN Accuracy on Noisy Data - Full Size Images")



```
# استخراج برچسب‌های واقعی برای سایر متریک‌های عملکرد
y_true = test_labels # برچسب‌های واقعی داده‌های آزمایشی

# پیش‌بینی برای داده‌های آزمایشی تمیز
y_pred_clean = clean_model.predict(test_data_clean) # پیش‌بینی برچسب‌ها برای داده‌های آزمایشی تمیز
y_pred_clean = np.argmax(y_pred_clean, axis=-1) # تبدیل احتمال‌ها به ایندکس‌های کلاس

# پیش‌بینی برای داده‌های آزمایشی نویزی
y_pred_noisy = noisy_model.predict(test_data_noisy) # پیش‌بینی برچسب‌ها برای داده‌های آزمایشی نویزی
y_pred_noisy = np.argmax(y_pred_noisy, axis=-1) # تبدیل احتمال‌ها به ایندکس‌های کلاس

313/313 6s 18ms/step
313/313 5s 14ms/step

# محاسبه متریک‌های مدل تمیز
accuracy_clean = accuracy_score(y_true, y_pred_clean) # محاسبه دقت
precision_clean = precision_score(y_true, y_pred_clean, average='weighted') # محاسبه دقت وزنی
recall_clean = recall_score(y_true, y_pred_clean, average='weighted') # محاسبه فراخوان وزنی
f1_clean = f1_score(y_true, y_pred_clean, average='weighted') # وزنی محاسبه امتیاز F1

# نمایش متریک‌ها
print("\nClean Data Metrics:")
print(f"Accuracy: {accuracy_clean:.4f}") # نمایش دقت
print(f"Precision: {precision_clean:.4f}") # نمایش دقت
print(f"Recall: {recall_clean:.4f}") # نمایش فراخوان
print(f"F1 Score: {f1_clean:.4f}") # نمایش امتیاز F1

Clean Data Metrics:
Accuracy: 0.8760
Precision: 0.8856
Recall: 0.8760
F1 Score: 0.8726

# محاسبه متریک‌های مدل نویزی
accuracy_noisy = accuracy_score(y_true, y_pred_noisy) # محاسبه دقت
precision_noisy = precision_score(y_true, y_pred_noisy, average='weighted') # محاسبه دقت وزنی
recall_noisy = recall_score(y_true, y_pred_noisy, average='weighted') # محاسبه فراخوان وزنی
f1_noisy = f1_score(y_true, y_pred_noisy, average='weighted') # وزنی محاسبه امتیاز F1

# نمایش متریک‌ها
print("\nNoisy Data Metrics:")
print(f"Accuracy: {accuracy_noisy:.4f}") # نمایش دقت
print(f"Precision: {precision_noisy:.4f}") # نمایش دقت
print(f"Recall: {recall_noisy:.4f}") # نمایش فراخوان
print(f"F1 Score: {f1_noisy:.4f}") # نمایش امتیاز F1

Noisy Data Metrics:
Accuracy: 0.8365
Precision: 0.8371
```

Recall: 0.8365  
F1 Score: 0.8321

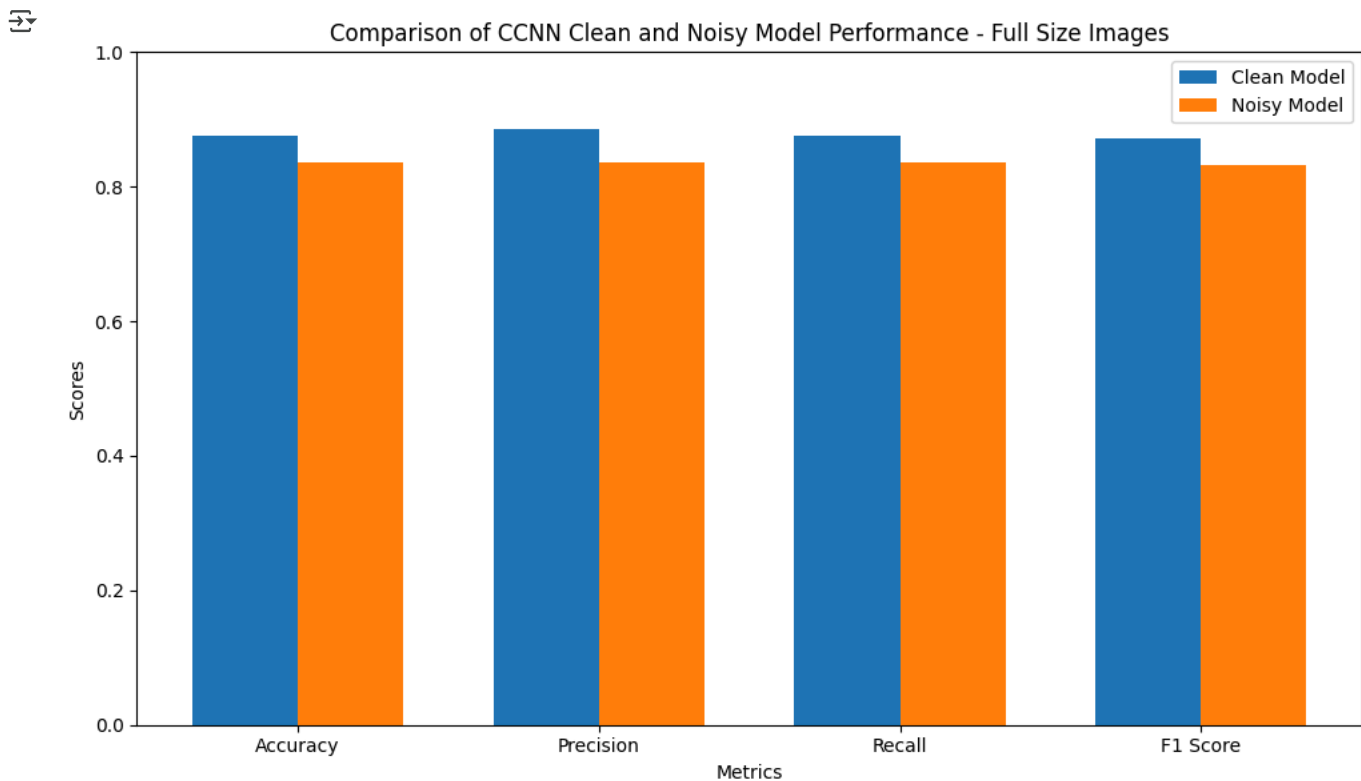
```
# تعریف متریک‌ها و مقادیر مربوطه
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score'] # نام متریک‌ها
clean_values = [accuracy_clean, precision_clean, recall_clean, f1_clean] # مقادیر متریک‌های مدل تمیز
noisy_values = [accuracy_noisy, precision_noisy, recall_noisy, f1_noisy] # مقادیر متریک‌های مدل نویزی

# تنظیمات برای رسم نمودار
x = np.arange(len(metrics)) # مکان برجسب‌ها
width = 0.35 # عرض میله‌ها
fig, ax = plt.subplots(figsize=(10, 6)) # ایجاد شکل و محور

# رسم میله‌های مدل تمیز و نویزی
rects1 = ax.bar(x - width/2, clean_values, width, label='Clean Model') # میله‌های مدل تمیز
rects2 = ax.bar(x + width/2, noisy_values, width, label='Noisy Model') # میله‌های مدل نویزی

# تنظیمات محور و عنوان
ax.set_xlabel('Metrics') # برجسب محور X
ax.set_ylabel('Scores') # برجسب محور Y
ax.set_title('Comparison of CCNN Clean and Noisy Model Performance - Full Size Images') # عنوان نمودار
ax.set_xticks(x) # تنظیم برجسب‌های محور X
ax.set_xticklabels(metrics) # برجسب‌های متریک‌ها
ax.legend() # نمایش افسانه
ax.set_ylim(0, 1) # از 0 تا 1 Y محدوده ثابت محور

# نمایش نمودار
plt.tight_layout() # تنظیمات نهایی
plt.show() # نمایش نمودار
```



In the next section we perform multiclass classification on images that are scaled down to 4x4 pixels to show a fair comparison with the image sizes provided to the QCNN. Subsets of the data are also taken in the same random sampling method used in the QCNN.

```
# تعداد نمونه‌هایی که در زیرمجموعه‌ها خواهد بود
# حفظ نسبت اصلی 6:1 از داده‌های آموزشی به آزمایشی
num_train = 6000 # تعداد کل نمونه‌های آموزشی
num_test = 1000 # تعداد کل نمونه‌های آزمایشی

# تعداد نمونه‌ها در هر کلاس (10 کلاس)
num_train_per_class = num_train // 10 # تعداد نمونه‌های آموزشی در هر کلاس
num_test_per_class = num_test // 10 # تعداد نمونه‌های آزمایشی در هر کلاس
```



```

# لیست‌هایی برای ذخیره زیرمجموعه‌های تصاویر و برچسب‌ها
train_images_subset = []
train_labels_subset = []
test_images_subset = []
test_labels_subset = []

# تکرار بر روی هر کلاس و نمونه‌گیری به طور مساوی
for i in range(10):
    class_indices_train = np.where(train_labels == i)[0] # ایندکس‌های کلاس در داده‌های آموزشی
    class_indices_test = np.where(test_labels == i)[0] # ایندکس‌های کلاس در داده‌های آزمایشی

    # نمونه‌گیری تصادفی از تصاویر در هر کلاس
    sampled_train_indices = np.random.choice(class_indices_train, num_train_per_class, replace=False) # ایندکس‌های نمونه‌های آموزشی
    sampled_test_indices = np.random.choice(class_indices_test, num_test_per_class, replace=False) # ایندکس‌های نمونه‌های آزمایشی

    # اضافه کردن آن‌ها به لیست
    train_images_subset.append(train_data_clean[sampled_train_indices]) # تصاویر آموزشی نمونه‌گیری شده
    train_labels_subset.append(train_labels[sampled_train_indices]) # برچسب‌های آموزشی نمونه‌گیری شده
    test_images_subset.append(test_data_clean[sampled_test_indices]) # تصاویر آزمایشی نمونه‌گیری شده
    test_labels_subset.append(test_labels[sampled_test_indices]) # برچسب‌های آزمایشی نمونه‌گیری شده

# تبدیل لیست‌ها به آرایه‌ها
train_images_subset = np.concatenate(train_images_subset) # ادغام تصاویر آموزشی
train_labels_subset = np.concatenate(train_labels_subset) # ادغام برچسب‌های آموزشی
test_images_subset = np.concatenate(test_images_subset) # ادغام تصاویر آزمایشی
test_labels_subset = np.concatenate(test_labels_subset) # ادغام برچسب‌های آزمایشی

# نمایش شکل داده‌های آموزشی و آزمایشی
print("Training Data Shape (After Subset):", train_images_subset.shape) # نمایش شکل داده‌های آموزشی
print("Testing Data Shape (After Subset):", test_images_subset.shape) # نمایش شکل داده‌های آزمایشی

# چاپ کلاس‌های منحصر به فرد برای تولید اینکه همه 10 کلاس شامل شده‌اند
print("Unique Train Labels:", np.unique(train_labels)) # کلاس‌های منحصر به فرد در داده‌های آموزشی
print("Unique Test Labels:", np.unique(test_labels)) # کلاس‌های منحصر به فرد در داده‌های آزمایشی

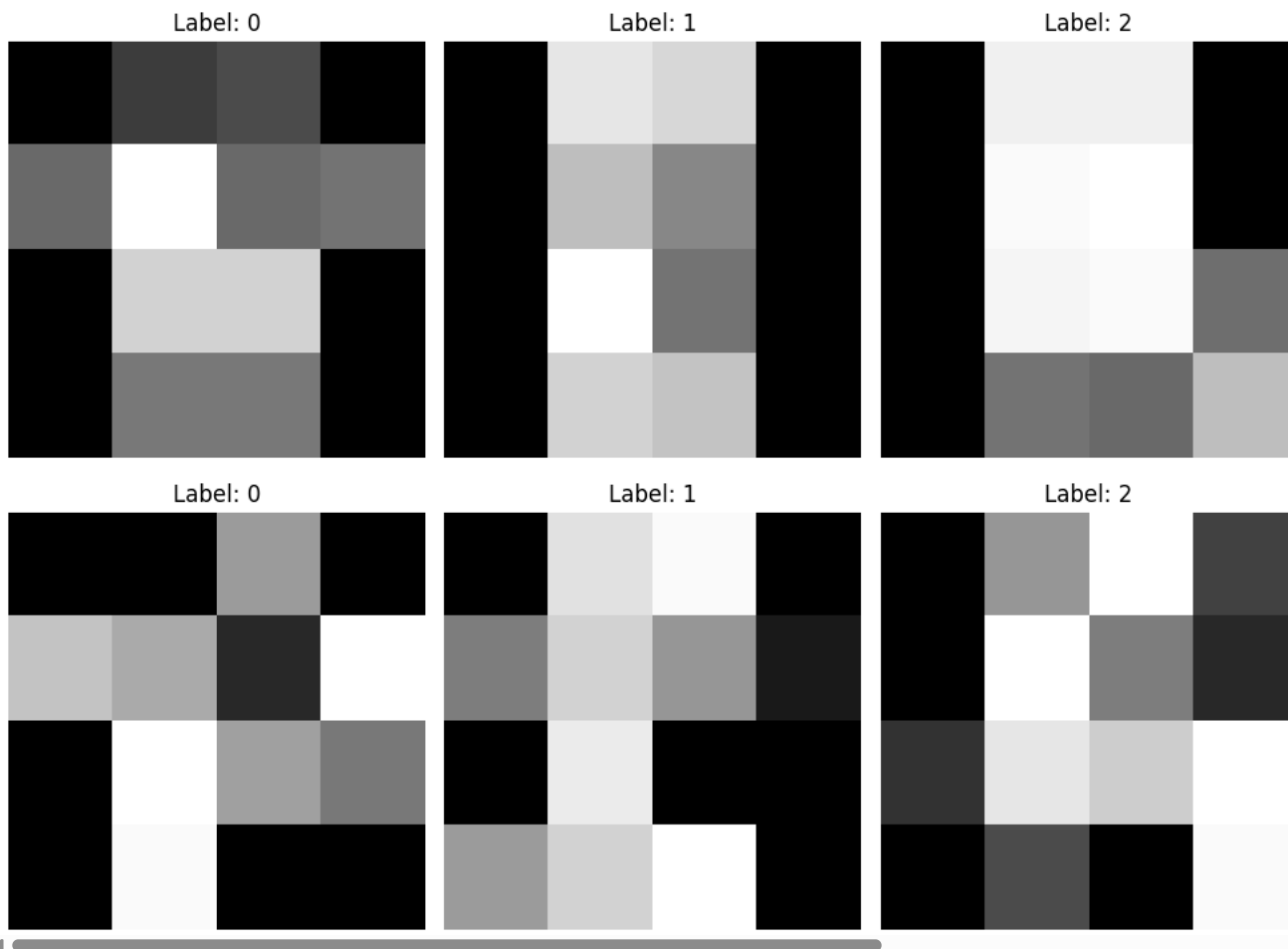
🔍 Training Data Shape (After Subset): (6000, 28, 28)
Testing Data Shape (After Subset): (1000, 28, 28)
Unique Train Labels: [0 1 2 3 4 5 6 7 8 9]
Unique Test Labels: [0 1 2 3 4 5 6 7 8 9]

# Create new resized input images
def resize_images_tf(images, new_size=(4, 4)):
    images = tf.expand_dims(images, axis=-1) # Add the channel dimension (1 for grayscale)
    resized_images = tf.image.resize(images, new_size)
    return resized_images

# Resize and show the clean and noisy 4x4 images
train_data_clean_4x4 = resize_images_tf(train_images_subset, new_size=(4, 4))
test_data_clean_4x4 = resize_images_tf(test_images_subset, new_size=(4, 4))
display_images(train_data_clean_4x4, train_labels_subset, num_images=3)

train_data_noisy_4x4 = add_gaussian_noise(train_data_clean_4x4, severity=5)
test_data_noisy_4x4 = add_gaussian_noise(test_data_clean_4x4, severity=5)
display_images(train_data_noisy_4x4, train_labels_subset, num_images=3)

```



```
# برای تصاویر 4x4 CNN تابعی برای ایجاد مدل
# این مدل نسخه‌ای مقیاس‌پذیر از مدل قبلی است
# با حذف دراپ‌اوت و اندازه هسته کوچکتر برای سازگاری با اندازه ورودی کوچکتر
def create_model_4x4():
    model = models.Sequential() # ایجاد مدل ترتیبی

    # بلوک 1
    model.add(layers.Conv2D(filters=4, kernel_size=(2, 2), activation='tanh', input_shape=(4, 4, 1))) # لایه کانولوشن
    model.add(layers.AveragePooling2D(pool_size=(2, 2), padding='same')) # لایه متوسط‌گیری

    # بلوک 2
    model.add(layers.Conv2D(filters=8, kernel_size=(2, 2), activation='tanh')) # لایه کانولوشن

    # بلوک 3
    model.add(layers.Conv2D(filters=8, kernel_size=(1, 1), activation='tanh')) # لایه کانولوشن

    # لایه‌های تخت و Dense
    model.add(layers.Flatten()) # تخت کردن داده‌ها
    model.add(layers.Dense(10, activation='softmax', kernel_regularizer=regularizers.l2(0.01))) # با 10 نورون برای 10 کلاس Dense لایه

    model.summary() # نمایش خلاصه مدل


    model.compile(optimizer='adam', # کمپایل مدل با بهینه‌ساز آدام
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(), # تابع هزینه
                  metrics=['accuracy']) # متریک دقت

    return model # بازگشت مدل

# ایجاد مدل تمیز برای دیتاست مقیاس‌پذیر
# برای داده‌های زیرمجموعه‌شده از اندازه دسته 32 استفاده خواهیم کرد زیرا دیتاست به طور قابل توجهی کوچکتر است
clean_model_4x4 = create_model_4x4() # 4x4 مدل ایجاد
history_clean_4x4 = clean_model_4x4.fit(train_data_clean_4x4, # آموزش مدل
                                       train_labels_subset,
                                       epochs=10, # تعداد دوره‌های آموزش
                                       batch_size=32, # اندازه دسته
                                       validation_data=(test_data_clean_4x4, test_labels_subset) # داده‌های اعتبارسنجی
                                       )
```

# ارزیابی بر روی داده‌های آزمایشی تمیز

```
test_loss_clean_4x4, test_acc_clean_4x4 = clean_model_4x4.evaluate(test_data_clean_4x4, test_labels_subset, verbose=2)
print(f"Clean 4x4 Data - Test Accuracy: {test_acc_clean_4x4:.4f}, Test Loss: {test_loss_clean_4x4:.4f}")
print(f"Clean 4x4 Data - Training Accuracy: {history_clean_4x4.history['accuracy'][-1]:.4f}")
```

 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`,  
super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

Model: "sequential\_2"


Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 3, 3, 4)	20
average_pooling2d (AveragePooling2D)	(None, 2, 2, 4)	0
conv2d_7 (Conv2D)	(None, 1, 1, 8)	136
conv2d_8 (Conv2D)	(None, 1, 1, 8)	72
flatten_2 (Flatten)	(None, 8)	0
dense_4 (Dense)	(None, 10)	90

Total params: 318 (1.24 KB)


Trainable params: 318 (1.24 KB)

Non-trainable params: 0 (0.00 B)


Epoch 1/10

188/188  2s 4ms/step - accuracy: 0.1250 - loss: 2.3848 - val\_accuracy: 0.2210 - val\_loss: 2.2233

Epoch 2/10

188/188  1s 3ms/step - accuracy: 0.2305 - loss: 2.1377 - val\_accuracy: 0.2920 - val\_loss: 1.8998


Epoch 3/10

188/188  1s 3ms/step - accuracy: 0.3253 - loss: 1.8618 - val\_accuracy: 0.3330 - val\_loss: 1.7799


Epoch 4/10

188/188  1s 4ms/step - accuracy: 0.3604 - loss: 1.7607 - val\_accuracy: 0.3840 - val\_loss: 1.7189


Epoch 5/10

188/188  1s 5ms/step - accuracy: 0.4182 - loss: 1.6951 - val\_accuracy: 0.4020 - val\_loss: 1.6771


Epoch 6/10

188/188  1s 3ms/step - accuracy: 0.4361 - loss: 1.6582 - val\_accuracy: 0.4260 - val\_loss: 1.6385

Epoch 7/10

188/188  1s 3ms/step - accuracy: 0.4499 - loss: 1.6279 - val\_accuracy: 0.4460 - val\_loss: 1.6072


Epoch 8/10

188/188  1s 3ms/step - accuracy: 0.4638 - loss: 1.5896 - val\_accuracy: 0.4560 - val\_loss: 1.5846

Epoch 9/10

188/188  1s 3ms/step - accuracy: 0.4726 - loss: 1.5765 - val\_accuracy: 0.4440 - val\_loss: 1.5629

Epoch 10/10

188/188  1s 3ms/step - accuracy: 0.4703 - loss: 1.5581 - val\_accuracy: 0.4550 - val\_loss: 1.5496

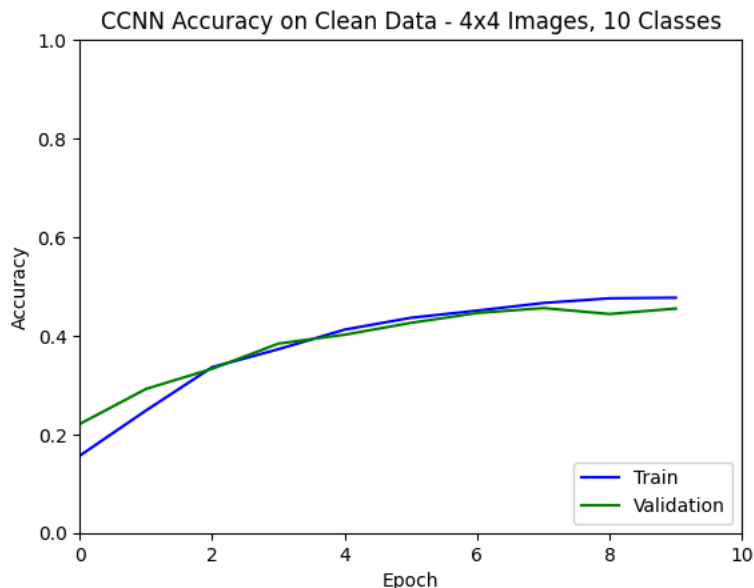
32/32 - 0s - 3ms/step - accuracy: 0.4550 - loss: 1.5496

Clean 4x4 Data - Test Accuracy: 0.4550, Test Loss: 1.5496

Clean 4x4 Data - Training Accuracy: 0.4770

# استفاده از تابع برای رسم دقت

```
plot_metrics(history_clean_4x4, "accuracy", "CCNN Accuracy on Clean Data - 4x4 Images, 10 Classes")
```



```
# ایجاد مدل نویزی برای دیتاست مقیاس‌پذیر
noisy_model_4x4 = create_model_4x4() # 4x4 مدل ایجاد
history_noisy_4x4 = noisy_model_4x4.fit(train_data_noisy_4x4, # آموزش مدل
                                       train_labels_subset,
                                       epochs=10, # تعداد دوره‌های آموزش
                                       batch_size=32, # اندازه دسته
                                       validation_data=(test_data_noisy_4x4, test_labels_subset) # داده‌های اعتبارسنجی
                                      )

# ارزیابی بر روی داده‌های آزمایشی نویزی
test_loss_noisy_4x4, test_acc_noisy_4x4 = noisy_model_4x4.evaluate(test_data_noisy_4x4, test_labels_subset, verbose=2)
print(f"Noisy 4x4 Data - Test Accuracy: {test_acc_noisy_4x4:.4f}, Test Loss: {test_loss_noisy_4x4:.4f}")
print(f"Noisy 4x4 Data - Training Accuracy: {history_noisy_4x4.history['accuracy'][-1]:.4f}")
```

🔗 Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 3, 3, 4)	20
average_pooling2d_1 (AveragePooling2D)	(None, 2, 2, 4)	0
conv2d_10 (Conv2D)	(None, 1, 1, 8)	136
conv2d_11 (Conv2D)	(None, 1, 1, 8)	72
flatten_3 (Flatten)	(None, 8)	0
dense_5 (Dense)	(None, 10)	90

Total params: 318 (1.24 KB)  
Trainable params: 318 (1.24 KB)  
Non-trainable params: 0 (0.00 B)

Epoch 1/10  
188/188 ————— 2s 4ms/step - accuracy: 0.0938 - loss: 2.3943 - val\_accuracy: 0.1820 - val\_loss: 2.2957

Epoch 2/10  
188/188 ————— 1s 3ms/step - accuracy: 0.1924 - loss: 2.2459 - val\_accuracy: 0.2210 - val\_loss: 2.0509

Epoch 3/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2294 - loss: 2.0356 - val\_accuracy: 0.2420 - val\_loss: 1.9680

Epoch 4/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2529 - loss: 1.9662 - val\_accuracy: 0.2660 - val\_loss: 1.9274

Epoch 5/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2558 - loss: 1.9439 - val\_accuracy: 0.2720 - val\_loss: 1.9006

Epoch 6/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2726 - loss: 1.9086 - val\_accuracy: 0.2990 - val\_loss: 1.8822

Epoch 7/10  
188/188 ————— 1s 4ms/step - accuracy: 0.2790 - loss: 1.9028 - val\_accuracy: 0.2930 - val\_loss: 1.8714

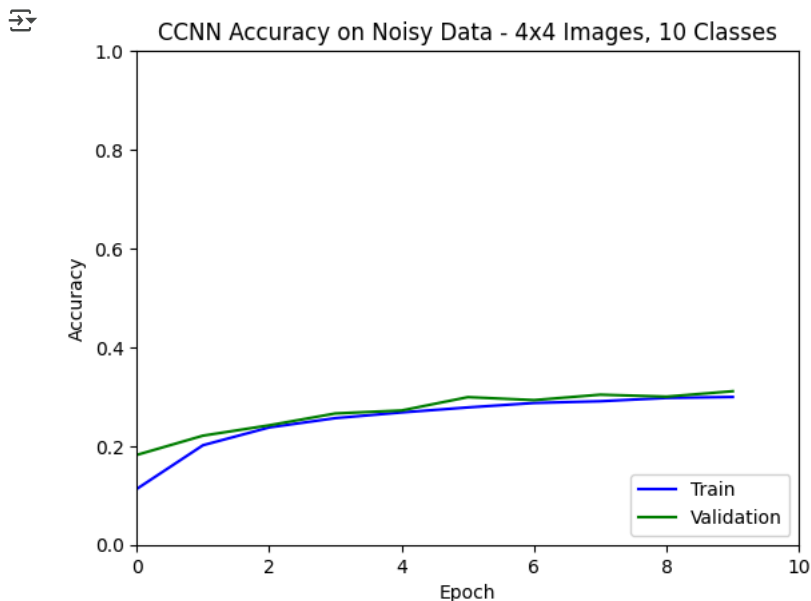
Epoch 8/10  
188/188 ————— 1s 5ms/step - accuracy: 0.2907 - loss: 1.8893 - val\_accuracy: 0.3040 - val\_loss: 1.8640

Epoch 9/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2992 - loss: 1.8765 - val\_accuracy: 0.3000 - val\_loss: 1.8595

Epoch 10/10  
188/188 ————— 1s 3ms/step - accuracy: 0.2932 - loss: 1.8813 - val\_accuracy: 0.3110 - val\_loss: 1.8516

32/32 - 0s - 3ms/step - accuracy: 0.3110 - loss: 1.8516  
Noisy 4x4 Data - Test Accuracy: 0.3110, Test Loss: 1.8516  
Noisy 4x4 Data - Training Accuracy: 0.2993

```
# استفاده از تابع برای رسم دقت
plot_metrics(history_noisy_4x4, "accuracy", "CCNN Accuracy on Noisy Data - 4x4 Images, 10 Classes")
```



# مقایسه متریک‌های عملکرد دیگر

y\_true = test\_labels\_subset # برچسب‌های واقعی داده‌های آزمایشی

# پیش‌بینی‌های مدل تمیز

y\_pred\_clean\_4x4 = clean\_model\_4x4.predict(test\_data\_clean\_4x4) # پیش‌بینی بر روی داده‌های تمیز

y\_pred\_clean\_4x4 = np.argmax(y\_pred\_clean\_4x4, axis=-1) # تبدیل احتمال‌ها به ایندکس‌های کلاس

# پیش‌بینی‌های مدل نویزی

y\_pred\_noisy\_4x4 = noisy\_model\_4x4.predict(test\_data\_noisy\_4x4) # پیش‌بینی بر روی داده‌های نویزی

y\_pred\_noisy\_4x4 = np.argmax(y\_pred\_noisy\_4x4, axis=-1) # تبدیل احتمال‌ها به ایندکس‌های کلاس

```
32/32 ————— 0s 4ms/step
32/32 ————— 0s 4ms/step
```

# متریک‌های داده‌های تمیز

accuracy\_clean\_4x4 = accuracy\_score(y\_true, y\_pred\_clean\_4x4) # محاسبه دقت مدل (نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها)

precision\_clean\_4x4 = precision\_score(y\_true, y\_pred\_clean\_4x4, average='weighted') # محاسبه دقت (Precision) محاسبه دقت با وزندهی بر اساس تعداد نمونه‌ها

recall\_clean\_4x4 = recall\_score(y\_true, y\_pred\_clean\_4x4, average='weighted') # محاسبه فراخوان (Recall) محاسبه فراخوان با وزندهی بر اساس تعداد نمونه‌ها

f1\_clean\_4x4 = f1\_score(y\_true, y\_pred\_clean\_4x4, average='weighted') # محاسبه نمره F1 محاسبه نمره F1 با وزندهی بر اساس تعداد نمونه‌ها

# چاپ متریک‌های داده‌های تمیز

print("\nClean Data Metrics:")

print(f"Accuracy: {accuracy\_clean\_4x4:.4f}") # نمایش دقت

print(f"Precision: {precision\_clean\_4x4:.4f}") # نمایش دقت (Precision)

print(f"Recall: {recall\_clean\_4x4:.4f}") # نمایش فراخوان (Recall)

print(f"F1 Score: {f1\_clean\_4x4:.4f}") # نمایش نمره F1

```
Clean Data Metrics:
Accuracy: 0.4550
Precision: 0.4299
Recall: 0.4550
F1 Score: 0.3948
```

# متریک‌های داده‌های نویزی

accuracy\_noisy\_4x4 = accuracy\_score(y\_true, y\_pred\_noisy\_4x4) # محاسبه دقت مدل بر روی داده‌های نویزی (نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها)

precision\_noisy\_4x4 = precision\_score(y\_true, y\_pred\_noisy\_4x4, average='weighted') # محاسبه دقت (Precision) محاسبه دقت با وزندهی بر اساس تعداد نمونه‌ها

recall\_noisy\_4x4 = recall\_score(y\_true, y\_pred\_noisy\_4x4, average='weighted') # محاسبه فراخوان (Recall) محاسبه فراخوان با وزندهی بر اساس تعداد نمونه‌ها

f1\_noisy\_4x4 = f1\_score(y\_true, y\_pred\_noisy\_4x4, average='weighted') # محاسبه نمره F1 محاسبه نمره F1 با وزندهی بر اساس تعداد نمونه‌ها

print("\nNoisy Data Metrics:")

print(f"Accuracy: {accuracy\_noisy\_4x4:.4f}")

print(f"Precision: {precision\_noisy\_4x4:.4f}")

print(f"Recall: {recall\_noisy\_4x4:.4f}")

print(f"F1 Score: {f1\_noisy\_4x4:.4f}")

```
Noisy Data Metrics:
Accuracy: 0.3110
Precision: 0.2638
```

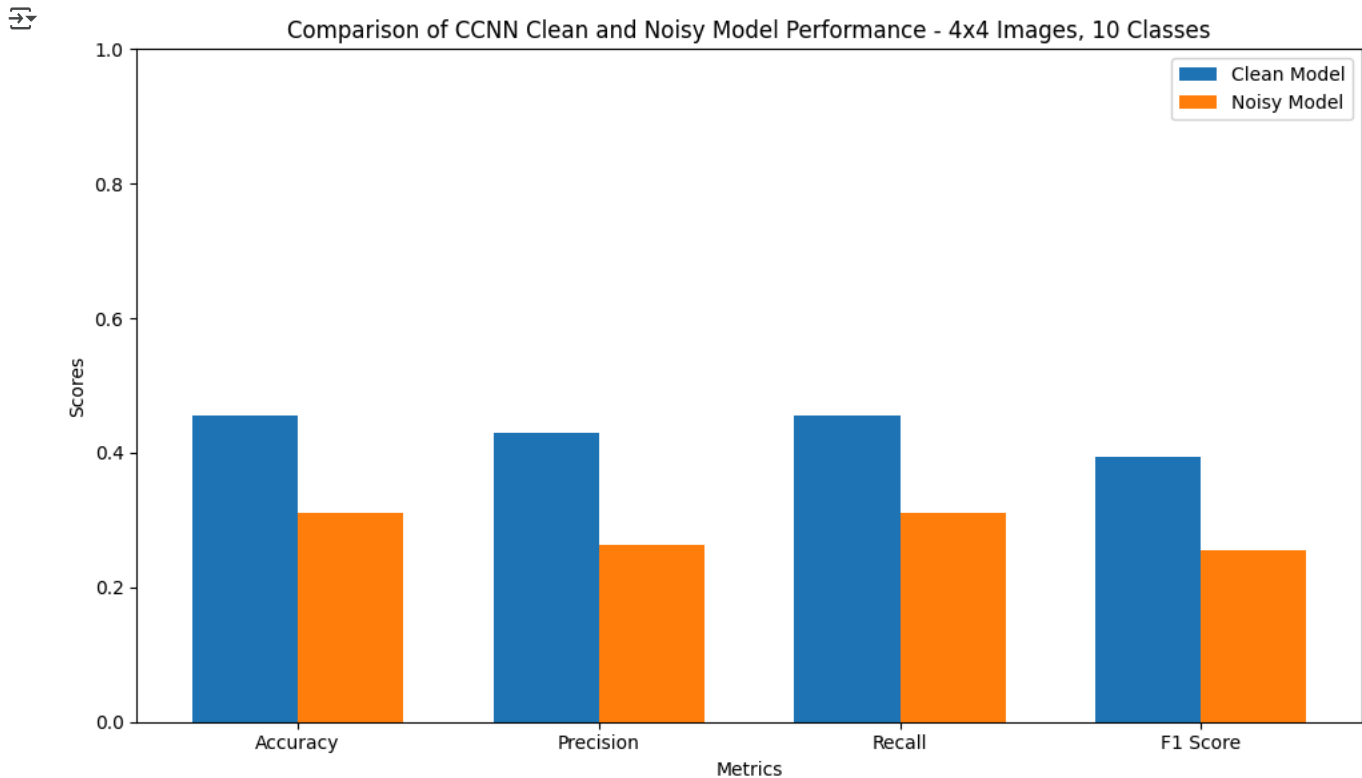
Recall: 0.3110  
F1 Score: 0.2560

```
# تعیین متریک‌ها و مقادیر آنها
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score'] # لیست متریک‌ها
clean_values = [accuracy_clean_4x4, precision_clean_4x4, recall_clean_4x4, f1_clean_4x4] # مقادیر متریک‌های داده‌های تمیز
noisy_values = [accuracy_noisy_4x4, precision_noisy_4x4, recall_noisy_4x4, f1_noisy_4x4] # مقادیر متریک‌های داده‌های نویزی

# تنظیمات نمودار
x = np.arange(len(metrics)) # مکان برجسبها بر روی محور
width = 0.35 # عرض میله‌ها
fig, ax = plt.subplots(figsize=(10, 6)) # ایجاد شکل و محور

# ایجاد میله‌ها برای مدل تمیز و نویزی
rects1 = ax.bar(x - width/2, clean_values, width, label='Clean Model') # میله‌های مدل تمیز
rects2 = ax.bar(x + width/2, noisy_values, width, label='Noisy Model') # میله‌های مدل نویزی

# تنظیمات محور و عنوان
ax.set_xlabel('Metrics') # برجسب محور x
ax.set_ylabel('Scores') # برجسب محور y
ax.set_title('Comparison of CCNN Clean and Noisy Model Performance - 4x4 Images, 10 Classes') # عنوان نمودار
ax.set_xticks(x) # تعیین مکان برجسبها بر روی محور
ax.set_xticklabels(metrics) # تعیین برجسب‌های محور
ax.legend() # نمایش افسانه (Legend)
ax.set_ylim(0, 1) # تنظیم محدوده محور y از 0 تا 1
plt.tight_layout() # بهینه‌سازی چیدمان
plt.show() # نمایش نمودار
```



The next section provides a different comparison with the binary classification QCNN model, where two classes are selected and again scaled down to 4x4. This model is simplified to have a minimal number of trainable parameters to be a fair comparison with the binary QCNN model.

```
# انتخاب فقط تی‌شرت/بالا (0) و شلوار (1)
classes_to_keep = [0, 1] # کلاس‌هایی که باید نگه‌داشته شوند
train_filter = np.isin(train_labels, classes_to_keep) # فیلتر کردن برجسب‌های آموزشی
train_data_bin = train_data_clean[train_filter] # داده‌های آموزشی فیلتر شده
train_labels_bin = train_labels[train_filter] # برجسب‌های آموزشی فیلتر شده

test_filter = np.isin(test_labels, classes_to_keep) # فیلتر کردن برجسب‌های آزمایشی
test_data_bin = test_data_clean[test_filter] # داده‌های آزمایشی فیلتر شده
test_labels_bin = test_labels[test_filter] # برجسب‌های آزمایشی فیلتر شده

# نمایش شکل داده‌های آموزشی و آزمایشی بلینری
```

```

print("نمایش شکل داده‌های آموزشی:", train_data_bin.shape) # نمایش شکل داده‌های آموزشی
print("نمایش شکل داده‌های آزمایشی:", test_data_bin.shape) # نمایش شکل داده‌های آزمایشی

# پیکسل 4x4 تبدیل تصاویر این کلاس‌ها به اندازه 4
train_binary_4x4 = resize_images_tf(train_data_bin, new_size=(4,4)) # 4x4 تغییر اندازه تصاویر آموزشی به 4
test_binary_4x4 = resize_images_tf(test_data_bin, new_size=(4,4)) # 4x4 تغییر اندازه تصاویر آزمایشی به 4

🔗 (28, 28, 12000) شکل داده‌های آموزشی باینری:
(28, 28, 2000) شکل داده‌های آزمایشی باینری:

```

```

def display_images_bin(images, labels, num_images=3):
    # انتخاب کلاس‌های منحصر به فرد
    selected_classes = np.unique(labels)[:num_images] # num_images انتخاب کلاس‌های منحصر به فرد تا تعداد

    # لیست‌های خالی برای ذخیره تصاویر و برچسب‌ها
    selected_images = []
    selected_labels = []

    # انتخاب یک تصویر از هر کلاس انتخاب‌شده
    for class_label in selected_classes:
        class_indices = np.where(labels == class_label)[0] # پیدا کردن ایندکس‌های کلاس
        selected_images.append(images[class_indices[0]]) # انتخاب اولین تصویر از کلاس
        selected_labels.append(labels[class_indices[0]]) # ذخیره برچسب مربوطه

    # تبدیل لیست‌ها به آرایه‌ها
    selected_images = np.array(selected_images) # تبدیل لیست تصاویر به آرایه
    selected_labels = np.array(selected_labels) # تبدیل لیست برچسب‌ها به آرایه

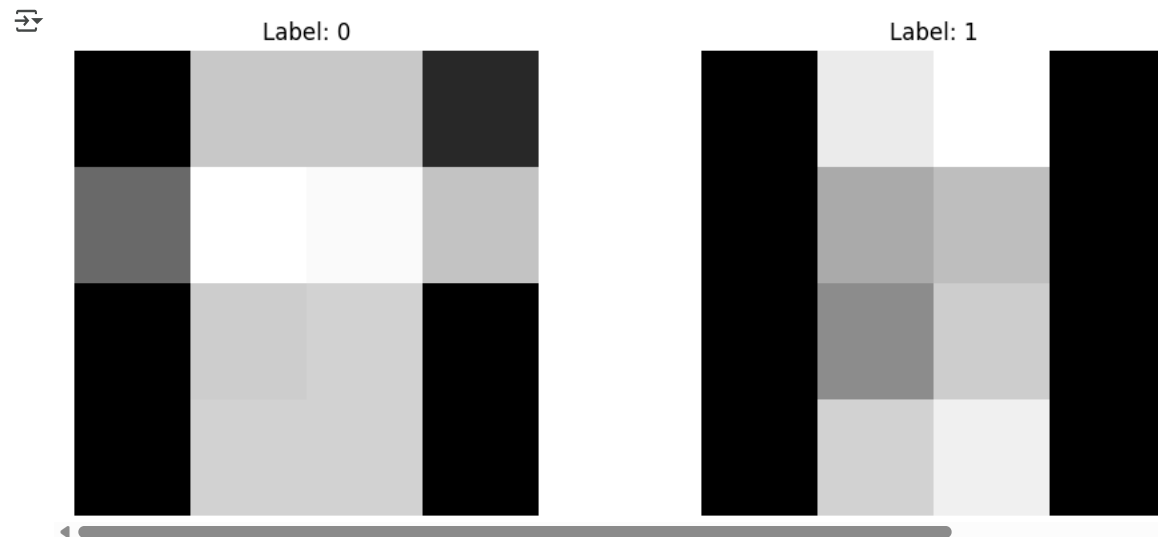
    # ایجاد یک شکل برای نمایش تصاویر
    fig, axes = plt.subplots(1, num_images, figsize=(10, 4)) # ایجاد زیرنمودارها

    for i, ax in enumerate(axes):
        ax.imshow(selected_images[i].squeeze(), cmap='gray') # نمایش تصویر با رنگ خاکستری
        ax.set_title(f'Label: {selected_labels[i]}') # نمایش برچسب تصویر
        ax.axis('off') # پنهان کردن محور

    plt.tight_layout() # بهینه‌سازی چیدمان
    plt.show() # نمایش شکل

# فراخوانی تابع برای نمایش تصاویر باینری
display_images_bin(train_binary_4x4, train_labels_bin, num_images=2) # نمایش 2 تصویر از داده‌های آموزشی باینری

```



```

# ایجاد یک مدل برای وظایف دست‌نویس باینری
# مدل بسیار ساده برای داشتن تعداد محدودی از پارامترهای قابل آموزش به منظور مقایسه عادلانه با
def create_binary_model():
    model = models.Sequential() # ایجاد یک مدل ترتیبی
    model.add(layers.Conv2D(4, (2,2), activation='relu', input_shape=(4,4,1))) # 2x2 لایه کانولوشن با 4 فیلتر و اندازه 2
    model.add(layers.Flatten()) # مسطح کردن خروجی از لایه کانولوشن
    model.add(layers.Dense(2)) # با 2 نورون Dense لایه
    model.add(layers.Dense(1)) # نهایی با 1 نورون برای پیش‌بینی باینری Dense لایه

    model.summary() # نمایش خلاصه مدل

    model.compile(optimizer='adam', # کامپایل مدل با استفاده از بهینه‌ساز

```

```
loss=tf.keras.losses.BinaryCrossentropy(), # استفاده از تابع هزینه باینری کراس انتروپی
metrics=['accuracy']) # دقت به عنوان متریک ارزیابی
```

```
return model # بازگشت مدل
```

```
# ایجاد و آموزش مدل باینری
```

```
clean_model_bin = create_binary_model() # ایجاد مدل باینری
history_bin = clean_model_bin.fit(train_binary_4x4, # آموزش مدل با داده‌های آموزشی باینری
                                train_labels_bin, # برچسب‌های آموزشی باینری
                                epochs=10, # تعداد دوره‌های آموزش
                                batch_size=128, # اندازه دسته
                                validation_data=(test_binary_4x4, test_labels_bin)) # داده‌های آزمایشی برای اعتبارسنجی
```

```
# ارزیابی بر روی داده‌های آزمایشی تمیز
```

```
test_loss_clean_bin, test_acc_clean_bin = clean_model_bin.evaluate(test_binary_4x4, test_labels_bin, verbose=2) # ارزیابی مدل
print(f"Clean Binary Data - Test Accuracy: {test_acc_clean_bin:.4f}, Test Loss: {test_loss_clean_bin:.4f}") # نمایش دقت و خطای تست
print(f"Clean Binary Data - Training Accuracy: {history_bin.history['accuracy'][-1]:.4f}") # نمایش دقت نهایی آموزش
```

```
→ /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`,
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 3, 3, 4)	20
flatten_4 (Flatten)	(None, 36)	0
dense_6 (Dense)	(None, 2)	74
dense_7 (Dense)	(None, 1)	3

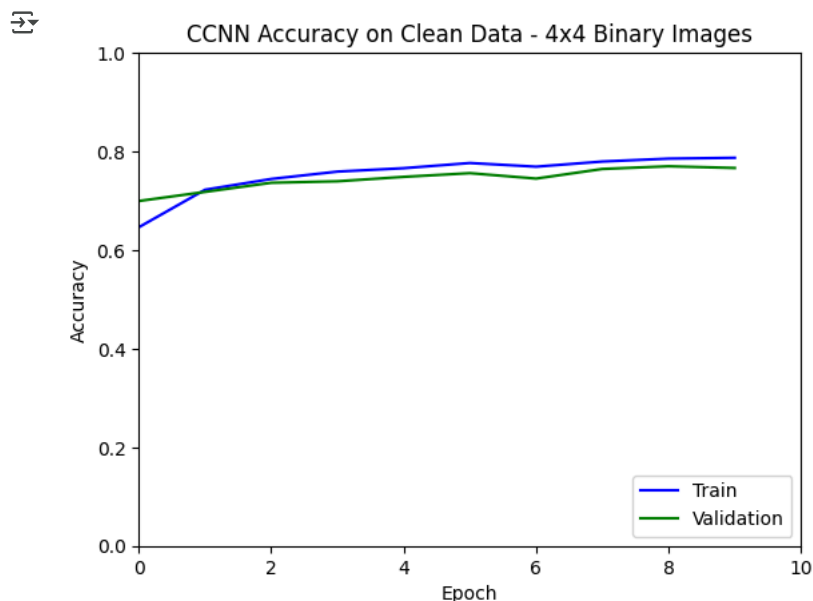
```
Total params: 97 (388.00 B)
Trainable params: 97 (388.00 B)
Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/10
94/94 ━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5895 - loss: 1.4812 - val_accuracy: 0.6995 - val_loss: 0.6216
Epoch 2/10
94/94 ━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7177 - loss: 0.5796 - val_accuracy: 0.7180 - val_loss: 0.5966
Epoch 3/10
94/94 ━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7418 - loss: 0.5465 - val_accuracy: 0.7365 - val_loss: 0.5658
Epoch 4/10
94/94 ━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7616 - loss: 0.5045 - val_accuracy: 0.7395 - val_loss: 0.5436
Epoch 5/10
94/94 ━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7635 - loss: 0.4838 - val_accuracy: 0.7485 - val_loss: 0.5315
Epoch 6/10
94/94 ━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7757 - loss: 0.4816 - val_accuracy: 0.7560 - val_loss: 0.4995
Epoch 7/10
94/94 ━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7837 - loss: 0.4609 - val_accuracy: 0.7450 - val_loss: 0.5200
Epoch 8/10
94/94 ━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7724 - loss: 0.4753 - val_accuracy: 0.7645 - val_loss: 0.4962
Epoch 9/10
94/94 ━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7888 - loss: 0.4437 - val_accuracy: 0.7700 - val_loss: 0.4912
Epoch 10/10
94/94 ━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7857 - loss: 0.4396 - val_accuracy: 0.7665 - val_loss: 0.4881
63/63 - 0s - 3ms/step - accuracy: 0.7665 - loss: 0.4881
Clean Binary Data - Test Accuracy: 0.7665, Test Loss: 0.4881
Clean Binary Data - Training Accuracy: 0.7872
```

```
# فرخوانی تابع برای ترسیم دقت
```

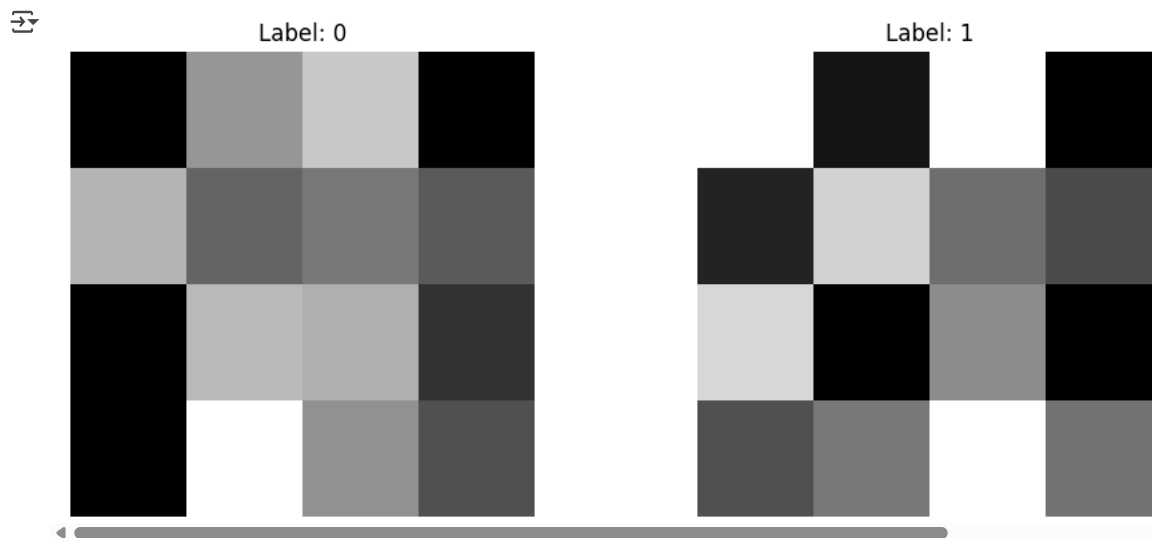
```
plot_metrics(history_bin, "accuracy", "CNN Accuracy on Clean Data - 4x4 Binary Images")
```





```
# حالا اضافه کردن نویز به تصاویر بلینری
train_data_noisy_bin = add_gaussian_noise(train_binary_4x4, severity=5) # افزودن نویز گاوسی به داده‌های آموزشی بلینری
test_data_noisy_bin = add_gaussian_noise(test_binary_4x4, severity=5) # افزودن نویز گاوسی به داده‌های آزمایشی بلینری

# نمایش تصاویر با نویز
display_images_bin(train_data_noisy_bin, train_labels_bin, num_images=2) # نمایش 2 تصویر از داده‌های آموزشی بلینری که نویز به آنها اضافه شده است
```



```
# ایجاد و آموزش مدل بلینری با داده‌های نویزی
noisy_model_bin = create_binary_model() # ایجاد مدل بلینری

history_noisy_bin = noisy_model_bin.fit(train_data_noisy_bin, # آموزش مدل با داده‌های آموزشی بلینری نویزی
                                       train_labels_bin, # برچسب‌های آموزشی بلینری
                                       epochs=10, # تعداد دوره‌های آموزش
                                       batch_size=128, # اندازه دسته
                                       validation_data=(test_data_noisy_bin, test_labels_bin)) # داده‌های آزمایشی برای اعتبارسنجی

# ارزیابی بر روی داده‌های آزمایشی نویزی
test_loss_noisy_bin, test_acc_noisy_bin = noisy_model_bin.evaluate(test_data_noisy_bin, test_labels_bin, verbose=2) # ارزیابی مدل
print(f"Noisy Binary Data - Test Accuracy: {test_acc_noisy_bin:.4f}, Test Loss: {test_loss_noisy_bin:.4f}") # نمایش دقت و خطای تست
print(f"Noisy Binary Data - Training Accuracy: {history_noisy_bin.history['accuracy'][-1]:.4f}") # نمایش دقت نهایی
```

Model: "sequential\_5"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 3, 3, 4)	20
flatten_5 (Flatten)	(None, 36)	0
dense_8 (Dense)	(None, 2)	74
dense_9 (Dense)	(None, 1)	3

Total params: 97 (388.00 B)  
Trainable params: 97 (388.00 B)  
Non-trainable params: 0 (0.00 B)

Epoch 1/10  
94/94 2s 6ms/step - accuracy: 0.4978 - loss: 6.2771 - val\_accuracy: 0.5000 - val\_loss: 3.0330