

یق است که توسط گوگل توسعه داده شده است. این کتابخانه به شما این امکان را می‌دهد که مدل‌های یادگیری عمیق را بسازید و آموزش دهید TensorFlow

```
#!pip install tensorflow==2.15.0
!pip install tensorflow==2.15.0 protobuf==3.20.3
!pip install tensorflow-quantum==0.7.3
```

```
import importlib, pkg_resources
importlib.reload(pkg_resources)
```

 Show hidden output

```
# TensorFlow وارد کردن کتابخانه
import tensorflow as tf
```

```
# TensorFlow Quantum وارد کردن کتابخانه
import tensorflow_quantum as tfq
```

```
# Keras از Fashion MNIST وارد کردن مجموعه داده
from tensorflow.keras.datasets import fashion_mnist
```

```
# Keras وارد کردن ابزارهای منظمسازی از
from tensorflow.keras import regularizers
```

```
# Cirq وارد کردن کتابخانه
import cirq
```

```
# SymPy وارد کردن
import sympy
```

```
# NumPy وارد کردن
import numpy as np
```

```
# scikit-learn وارد کردن متریک‌های ارزیابی از
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# Jupyter Notebook تنظیم نمایش نمودارها در
%matplotlib inline
```

```
# Matplotlib وارد کردن کتابخانه
import matplotlib.pyplot as plt
```

```
# SVG برای نمایش مدارهای کوانتومی به صورت Cirq از SVGCircuit
from cirq.contrib.svg import SVGCircuit
```

```
# Fashion MNIST بارگذاری مجموعه داده
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```


```
# T-shirt (0) و Trouser (1) انتخاب فقط کلاس
classes_to_keep = [0, 1] # کلاسهایی که می‌خواهیم نگه‌داریم
train_filter = np.isin(train_labels, classes_to_keep) # فیلتر کردن داده‌های آموزشی
train_images = train_images[train_filter] # اعمال فیلتر به تصاویر آموزشی
train_labels = train_labels[train_filter] # اعمال فیلتر به برچسب‌های آموزشی
```

```
test_filter = np.isin(test_labels, classes_to_keep) # فیلتر کردن داده‌های آزمایشی
test_images = test_images[test_filter] # اعمال فیلتر به تصاویر آزمایشی
test_labels = test_labels[test_filter] # اعمال فیلتر به برچسب‌های آزمایشی
```

```
# تغییر مقیاس تصاویر به بازه [0, 1]
train_images = train_images / 255.0 # نرمال‌سازی تصاویر آموزشی
test_images = test_images / 255.0 # نرمال‌سازی تصاویر آزمایشی
```

```
# QCNN (یک کانال) اضافه کردن ابعاد کانال برای سازگاری با پایپ‌لاین
train_images = train_images[..., np.newaxis] # اضافه کردن بعد جدید به تصاویر آموزشی
test_images = test_images[..., np.newaxis] # اضافه کردن بعد جدید به تصاویر آزمایشی
```

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

 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>  
29515/29515 [=====] - 0s 0us/step  
Download data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>  
26421880/26421880 [=====] - 1s 0us/step  
Download 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 [=====] - 1s 0us/step
Training Data Shape: (12000, 28, 28, 1)
Testing Data Shape: (2000, 28, 28, 1)
```

```
# پیدا کردن اولین ایندکس برای کلاس 0 (T-shirt)
class_0_index = np.where(train_labels == 0)[0][0]

# پیدا کردن اولین ایندکس برای کلاس 1 (Trouser)
class_1_index = np.where(train_labels == 1)[0][0]

# انتخاب ایندکس‌های کلاس‌های مورد نظر
selected = [class_0_index, class_1_index]

# چاپ جفت‌های ورودی و خروجی
for index in selected:
    input_image = train_images[index] # گرفتن تصویر ورودی برای ایندکس انتخاب شده
    output_label = train_labels[index] # گرفتن برچسب خروجی برای ایندکس انتخاب شده

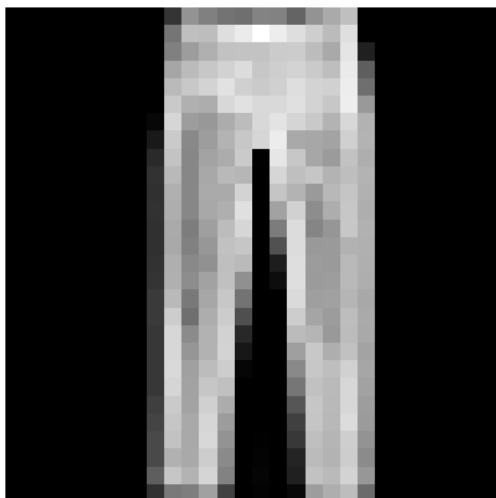
# نمایش تصویر
plt.imshow(input_image, cmap='gray') # نمایش تصویر به صورت خاکستری
plt.title(f'Label: {output_label}') # عنوان تصویر شامل برچسب
plt.axis('off') # پنهان کردن محورهای نمودار
plt.show() # نمایش تصویر
```



Label: 0



Label: 1



```
# برای کدگذاری به کیوبیت‌ها بسیار بزرگ است و قادر به شبیه‌سازی مدار نخواهد بود. 28x28
# 4x تغییر اندازه تصاویر به 4
train_images = tf.image.resize(train_images, (4, 4)).numpy() # تغییر اندازه تصاویر آموزشی
test_images = tf.image.resize(test_images, (4, 4)).numpy() # تغییر اندازه تصاویر آزمایشی

# چاپ جفت‌های ورودی و خروجی
for index in selected:
```

```

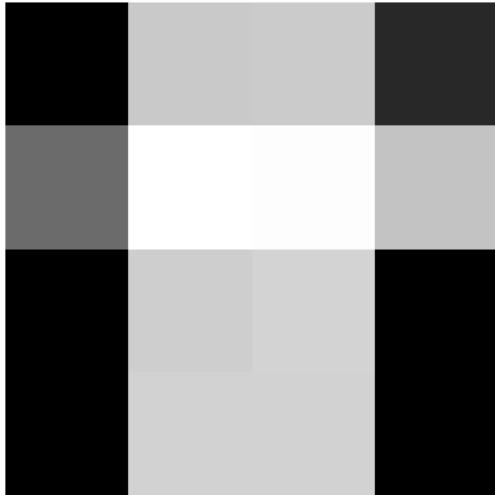
input_image = train_images[index] # گرفتن تصویر ورودی برای ایندکس انتخاب شده
output_label = train_labels[index] # گرفتن برچسب خروجی برای ایندکس انتخاب شده

# نمایش تصویر
plt.imshow(input_image, cmap='gray') # نمایش تصویر به صورت خاکستری
plt.title(f'Label: {output_label}') # عنوان تصویر شامل برچسب
plt.axis('off') # پنهان کردن محورها
plt.show() # نمایش تصویر

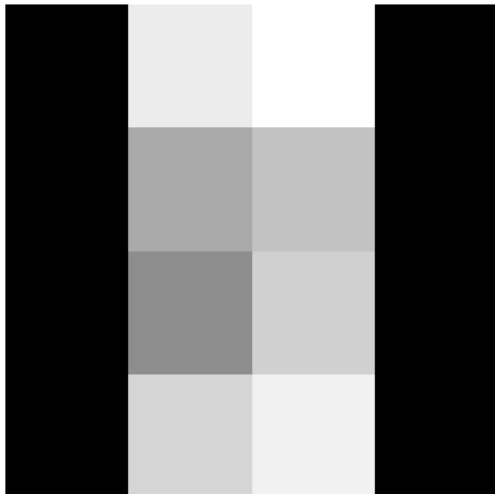
```



Label: 0



Label: 1



```

# تابعی برای افزودن نویز گاوسی به تصاویر
def add_gaussian_noise(images, mean=0.0, severity=1):
    # سطوح شدت مختلف برای نویز
    severity_levels = [0.08, 0.12, 0.18, 0.26, 0.38]
    stddev = severity_levels[severity - 1] # گرفتن انحراف معیار بر اساس شدت (1 تا 5)

    # تولید نویز گاوسی
    noise = np.random.normal(mean, stddev, images.shape)

    # افزودن نویز به تصاویر
    noisy_images = images + noise

    # اطمینان از اینکه مقادیر پیکسلها در بازه نرمال شده قرار دارند
    noisy_images = np.clip(noisy_images, 0.0, 1.0)

    return noisy_images # بازگشت تصاویر با نویز

# افزودن نویز به داده‌های آموزشی و آزمایشی
train_data_noisy = add_gaussian_noise(train_images, severity=5) # حداکثر شدت نویز برای داده‌های آموزشی
test_data_noisy = add_gaussian_noise(test_images, severity=5) # حداکثر شدت نویز برای داده‌های آزمایشی

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

```

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

```
# چاپ جفت‌های ورودی و خروجی
```

```
for index in selected:
```

```
    input_image = train_data_noisy[index] # گرفتن تصویر ورودی با نویز برای ایندکس انتخاب شده
```

```
    output_label = train_labels[index] # گرفتن برچسب خروجی برای ایندکس انتخاب شده
```

```
# نمایش تصویر
```

```
plt.imshow(input_image, cmap='gray') # نمایش تصویر به صورت خاکستری
```

```
plt.title(f'Label: {output_label}') # عنوان تصویر شامل برچسب
```

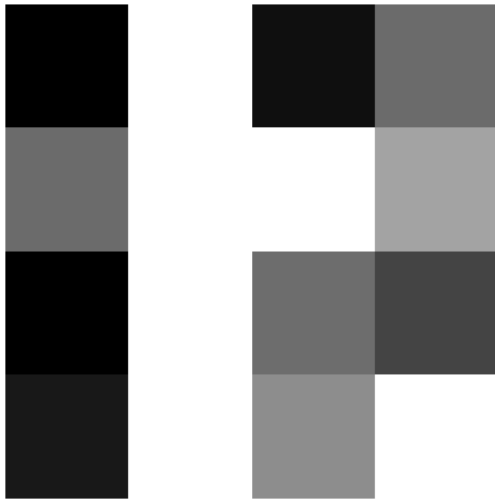
```
plt.axis('off') # پنهان کردن محورهای نمودار
```

```
plt.show() # نمایش تصویر
```

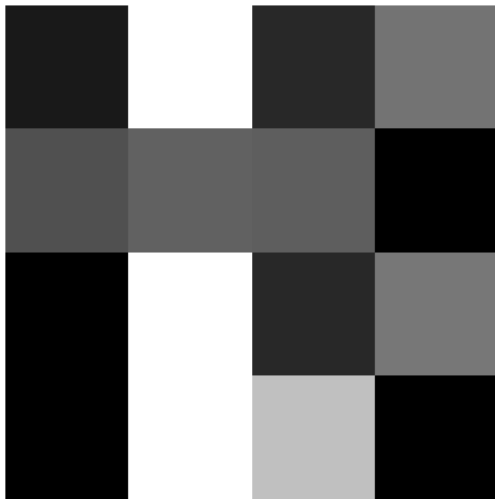
```
➡ Noisy Training Data Shape: (12000, 4, 4, 1)
```

```
Noisy Testing Data Shape: (2000, 4, 4, 1)
```

Label: 0



Label: 1



```
# کدگذاری هر پیکسل به یک کوبیت، فعالسازی حالت بستگی به آستانه دارد
```

```
THRESHOLD = 0.5 # تعیین آستانه برای کدگذاری باینری
```

```
# ایجاد کدگذاری باینری برای داده‌های پاک
```

```
x_train_bin_clean = np.array(train_images > THRESHOLD, dtype=np.float32) # کدگذاری تصاویر آموزشی پاک
```

```
x_test_bin_clean = np.array(test_images > THRESHOLD, dtype=np.float32) # کدگذاری تصاویر آزمایشی پاک
```

```
# ایجاد کدگذاری باینری برای داده‌های نویزی
```

```
x_train_bin_noisy = np.array(train_data_noisy > THRESHOLD, dtype=np.float32) # کدگذاری تصاویر آموزشی با نویز
```

```
x_test_bin_noisy = np.array(test_data_noisy > THRESHOLD, dtype=np.float32) # کدگذاری تصاویر آزمایشی با نویز
```

```
# تبدیل تصاویر کلاسیک به داده‌های کوانتومی
```

```
def convert_to_circuit(image):
```

```
    values = np.ndarray.flatten(image) # مسطح کردن تصویر به یک آرایه یک بعدی
```

```
    qubits = cirq.GridQubit.rect(4, 4) # 4x4 ایجاد کوبیت‌ها در یک شبکه 4
```

```
    circuit = cirq.Circuit() # ایجاد مدار کوانتومی جدید
```

```

for i, value in enumerate(values):
    if value: # اگر مقدار پیکسل 1 باشد
        circuit.append(cirq.X(qubits[i])) # اعمال گیت X (NOT) مربوطه
return circuit # بازگشت مدار کوانتومی

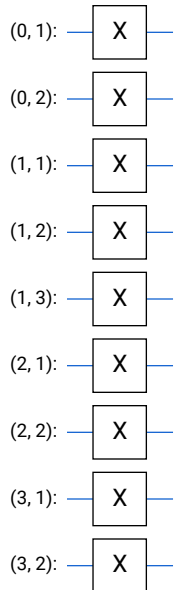
# تبدیل تصاویر آموزشی و آزمایشی پاک به مدار کوانتومی
x_train_circ_clean = [convert_to_circuit(x) for x in x_train_bin_clean] # تبدیل داده‌های آموزشی پاک
x_test_circ_clean = [convert_to_circuit(x) for x in x_test_bin_clean] # تبدیل داده‌های آزمایشی پاک

# تبدیل تصاویر آموزشی و آزمایشی نویزی به مدار کوانتومی
x_train_circ_noisy = [convert_to_circuit(x) for x in x_train_bin_noisy] # تبدیل داده‌های آموزشی نویزی
x_test_circ_noisy = [convert_to_circuit(x) for x in x_test_bin_noisy] # تبدیل داده‌های آزمایشی نویزی

# نمایش مدار کوانتومی برای اولین تصویر آموزشی پاک
SVGcircuit(x_train_circ_clean[0]) # نمایش مدار کوانتومی برای تصویر اول

```

⚠ WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.  
WARNING:matplotlib.font\_manager:findfont: Font family 'Arial' not found.



```

# به تئورها Cirq تبدیل مدارهای
x_train_tfirc_clean = tfq.convert_to_tensor(x_train_circ_clean) # تبدیل مدارهای آموزشی پاک به تئور
x_test_tfirc_clean = tfq.convert_to_tensor(x_test_circ_clean) # تبدیل مدارهای آزمایشی پاک به تئور

x_train_tfirc_noisy = tfq.convert_to_tensor(x_train_circ_noisy) # تبدیل مدارهای آموزشی نویزی به تئور
x_test_tfirc_noisy = tfq.convert_to_tensor(x_test_circ_noisy) # تبدیل مدارهای آزمایشی نویزی به تئور

# توابعی برای ایجاد لایه‌های مدار کوانتومی
class CircuitLayerBuilder():
    def __init__(self, data_qubits, readout):
        self.data_qubits = data_qubits # کیوبیت‌های داده
        self.readout = readout # کیوبیت خواندن (readout)

    def add_layer(self, circuit, gate, prefix):
        # افزودن یک لایه به مدار
        for i, qubit in enumerate(self.data_qubits):

```

```
symbol = sympy.Symbol(prefix + '-' + str(i)) # ایجاد نماد برای پارامتر گیت
circuit.append(gate(qubit, self.readout)**symbol) # افزودن گیت به مدار با نماد
```

# نمونه‌ای از نحوه ایجاد مدار

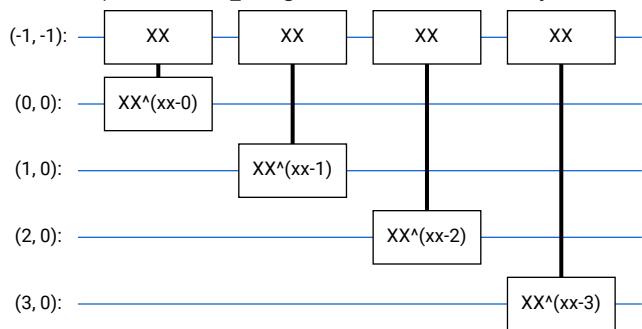
```
demo_builder = CircuitLayerBuilder(data_qubits=cirq.GridQubit.rect(4, 1), # ایجاد کیوبیت‌های داده در یک شبکه 4x1
readout=cirq.GridQubit(-1, -1)) # (1-, 1-) در مختصات
```

circuit = cirq.Circuit() # ایجاد مدار کوانتومی جدید

demo\_builder.add\_layer(circuit, gate=cirq.XX, prefix='xx') # به مدار XX افزودن یک لایه با گیت

SVGCircuit(circuit) # نمایش مدار کوانتومی

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```



# و عملیات خواندن QCNN ایجاد مدار مدل

```
def create_quantum_model():
    data_qubits = cirq.GridQubit.rect(4, 4) # 4x4 ایجاد کیوبیت‌های داده در یک شبکه
    readout = cirq.GridQubit(-1, -1) # [1-, 1-) در مختصات ایجاد یک کیوبیت خواندن
    circuit = cirq.Circuit() # ایجاد مدار کوانتومی جدید
```

# آماده‌سازی کیوبیت خواندن

```
circuit.append(cirq.X(readout)) # بر روی کیوبیت خواندن X اعمال گیت
circuit.append(cirq.H(readout)) # اعمال گیت هادامارد بر روی کیوبیت خواندن
```

```
builder = CircuitLayerBuilder(
    data_qubits=data_qubits, # ارسال کیوبیت‌های داده به سازنده
    readout=readout # ارسال کیوبیت خواندن به سازنده
)
```

# لایه اول، درهم‌تنیدگی

```
builder.add_layer(circuit, cirq.XX, "xx1") # XX افزودن لایه با گیت
builder.add_layer(circuit, cirq.ZZ, "zz1") # ZZ افزودن لایه با گیت
```

# لایه دوم، گیت‌های کوانتومی پارامتری

```
for q in data_qubits:
    theta = sympy.Symbol(f"theta_{q.row}_{q.col}", regularizer=regularizers.l2(0.01)) # y پارامتر برای چرخش حول محور
    phi = sympy.Symbol(f"phi_{q.row}_{q.col}", regularizer=regularizers.l2(0.01)) # z پارامتر برای چرخش حول محور
    circuit.append(cirq.ry(theta)(q)) # با theta پارامتر y افزودن گیت چرخش حول محور
    circuit.append(cirq.rz(phi)(q)) # با phi پارامتر z افزودن گیت چرخش حول محور
```

# در نهایت، آماده‌سازی کیوبیت خواندن

```
circuit.append(cirq.H(readout)) # اعمال گیت هادامارد مجدداً بر روی کیوبیت خواندن
```

```
return circuit, cirq.Z(readout) # بازگشت مدار و عملیات خواندن
```

# اجرای تابع

```
model_circuit, model_readout = create_quantum_model() # ایجاد مدار و عملیات خواندن
```

# ساخت مدل پاک

```
clean_model = tf.keras.Sequential([
    # رمزگذاری شده است tf.string ورودی داده‌ها که به عنوان یک رشته
    tf.keras.layers.Input(shape=(1, dtype=tf.string))
])
```

```
tf.keras.layers.Dense(1, dtype='float32'),
# انتظار مقدار گیت خواندن را باز می‌گرداند، در بازه [1, -1] PQC لایه
tfq.layers.PQC(model_circuit, model_readout),
])
```

# کامپایل مدل پاک و چاپ خلاصه آن

```
clean_model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), # تعیین تابع ضرر به عنوان تقاطع باینری
    optimizer=tf.keras.optimizers.Adam(), # استفاده از بهینه‌ساز آدام
    metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0.0)] # 0.0 تعیین معیار دقت باینری با آستانه
)
```

print(clean\_model.summary()) # چاپ خلاصه مدل شامل تعداد لایه‌ها و پارامترها

```
Model: "sequential"
Layer (type)                Output Shape                Param #
-----
pqc (PQC)                   (None, 1)                  64
-----
Total params: 64 (256.00 Byte)
Trainable params: 64 (256.00 Byte)
Non-trainable params: 0 (0.00 Byte)
None
```

# آموزش مدل پاک با داده‌های پاک

```
history_clean = clean_model.fit(
    x_train_tfirc_clean, train_labels, # داده‌های آموزشی و برچسب‌ها
    batch_size=128, # اندازه دسته برای آموزش
    epochs=10, # تعداد دوره‌های آموزش
    verbose=1, # نمایش پیشرفت در حین آموزش
    validation_data=(x_test_tfirc_clean, test_labels) # داده‌های اعتبارسنجی
)
```

# ارزیابی مدل روی داده‌های آزمایشی

test\_loss\_clean, test\_acc\_clean = clean\_model.evaluate(x\_test\_tfirc\_clean, test\_labels, verbose=2)

# چاپ دقت و ضرر آزمون

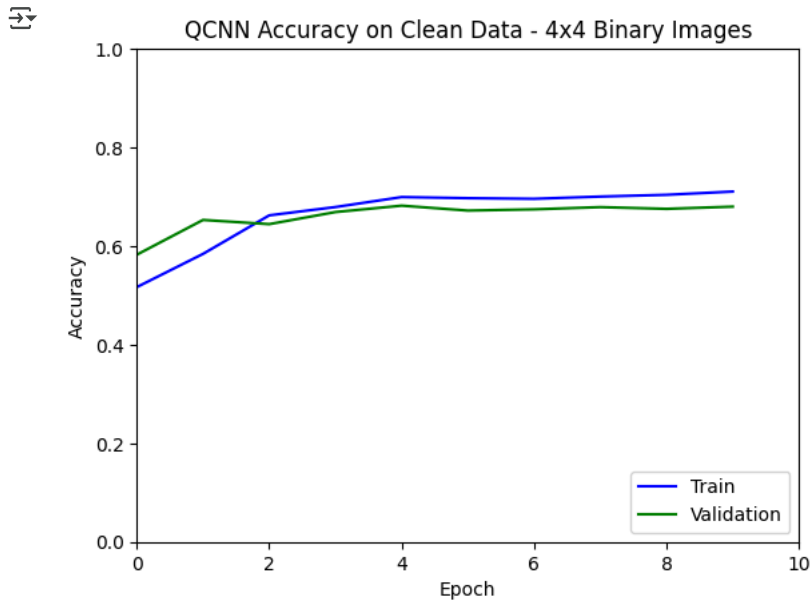
```
print(f"Clean Data - Test Accuracy: {test_acc_clean:.4f}, Test Loss: {test_loss_clean:.4f}")
print(f"Clean Data - Training Accuracy: {history_clean.history['binary_accuracy'][-1]:.4f}")
```

```
Epoch 1/10
94/94 [=====] - 262s 3s/step - loss: 0.6926 - binary_accuracy: 0.5167 - val_loss: 0.6897 - val_binary_acc: 0.5167
Epoch 2/10
94/94 [=====] - 261s 3s/step - loss: 0.6717 - binary_accuracy: 0.5842 - val_loss: 0.6429 - val_binary_acc: 0.5842
Epoch 3/10
94/94 [=====] - 257s 3s/step - loss: 0.6245 - binary_accuracy: 0.6623 - val_loss: 0.6170 - val_binary_acc: 0.6623
Epoch 4/10
94/94 [=====] - 265s 3s/step - loss: 0.6081 - binary_accuracy: 0.6792 - val_loss: 0.6109 - val_binary_acc: 0.6792
Epoch 5/10
94/94 [=====] - 262s 3s/step - loss: 0.6010 - binary_accuracy: 0.6994 - val_loss: 0.6072 - val_binary_acc: 0.6994
Epoch 6/10
94/94 [=====] - 258s 3s/step - loss: 0.5969 - binary_accuracy: 0.6972 - val_loss: 0.6061 - val_binary_acc: 0.6972
Epoch 7/10
94/94 [=====] - 230s 2s/step - loss: 0.5941 - binary_accuracy: 0.6959 - val_loss: 0.6044 - val_binary_acc: 0.6959
Epoch 8/10
94/94 [=====] - 229s 2s/step - loss: 0.5920 - binary_accuracy: 0.7004 - val_loss: 0.6043 - val_binary_acc: 0.7004
Epoch 9/10
94/94 [=====] - 230s 2s/step - loss: 0.5874 - binary_accuracy: 0.7040 - val_loss: 0.5979 - val_binary_acc: 0.7040
Epoch 10/10
94/94 [=====] - 231s 2s/step - loss: 0.5807 - binary_accuracy: 0.7106 - val_loss: 0.5945 - val_binary_acc: 0.7106
63/63 - 5s - loss: 0.5945 - binary_accuracy: 0.6800 - 5s/epoch - 85ms/step
Clean Data - Test Accuracy: 0.6800, Test Loss: 0.5945
Clean Data - Training Accuracy: 0.7106
```

# تابعی برای رسم معیارها

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

```
# فراخوانی تابع برای رسم دقت مدل پاک
plot_metrics(history_clean, 'binary_accuracy', 'QCNN Accuracy on Clean Data - 4x4 Binary Images')
```



```
# ساخت مدل نویز
noisy_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(), dtype=tf.string), # ورودی داده‌ها به عنوان یک رشته
    # انتظار مقدار گیت خواندن را باز می‌گرداند، در بازه [1, -1]
    tf.keras.layers.PQC(model_circuit, model_readout),
])

# کامپایل مدل نویزی
noisy_model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), # تعیین تابع ضرر به عنوان تقاطع بلژیکی
    optimizer=tf.keras.optimizers.Adam(), # استفاده از بهینه‌ساز آدام
    metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0.0)] # تعیین معیار دقت بلژیکی با آستانه 0.0
)
```

```
print(noisy_model.summary()) # چاپ خلاصه مدل شامل تعداد لایه‌ها و پارامترها
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
pqc_1 (PQC)	(None, 1)	64

```

Total params: 64 (256.00 Byte)
Trainable params: 64 (256.00 Byte)
Non-trainable params: 0 (0.00 Byte)

None
/usr/local/lib/python3.11/dist-packages/keras/src/initializers/initializers.py:120: UserWarning: The initializer RandomUniform is u
warnings.warn(

```

```
# آموزش مدل نویزی با داده‌های نویزی
history_noisy = noisy_model.fit(
    x_train_tfcirc_noisy, train_labels, # داده‌های آموزشی و برچسب‌ها
    batch_size=128, # اندازه دسته برای آموزش
    epochs=10, # تعداد دوره‌های آموزش
    verbose=1, # نمایش پیشرفت در حین آموزش
    validation_data=(x_test_tfcirc_noisy, test_labels) # داده‌های اعتبارسنجی
)

# ارزیابی مدل روی داده‌های آزمایشی
test_loss_noisy, test_acc_noisy = noisy_model.evaluate(x_test_tfcirc_noisy, test_labels, verbose=2)

# چاپ دقت آزمون
print(f"Noisy Data - Test Accuracy: {test_acc_noisy:.4f}, Test Loss: {test_loss_noisy:.4f}")
print(f"Noisy Data - Training Accuracy: {history_noisy.history['binary_accuracy'][-1]:.4f}")
```

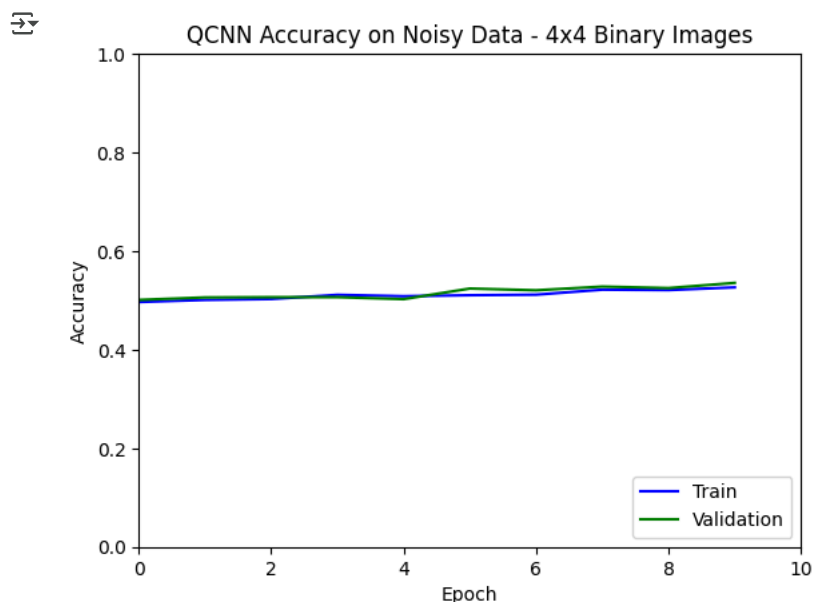


```

Epoch 1/10
94/94 [=====] - 232s 2s/step - loss: 0.6932 - binary_accuracy: 0.4967 - val_loss: 0.6931 - val_binary_accu
Epoch 2/10
94/94 [=====] - 232s 2s/step - loss: 0.6932 - binary_accuracy: 0.5009 - val_loss: 0.6931 - val_binary_accu
Epoch 3/10
94/94 [=====] - 230s 2s/step - loss: 0.6931 - binary_accuracy: 0.5027 - val_loss: 0.6931 - val_binary_accu
Epoch 4/10
94/94 [=====] - 229s 2s/step - loss: 0.6931 - binary_accuracy: 0.5113 - val_loss: 0.6930 - val_binary_accu
Epoch 5/10
94/94 [=====] - 229s 2s/step - loss: 0.6930 - binary_accuracy: 0.5085 - val_loss: 0.6930 - val_binary_accu
Epoch 6/10
94/94 [=====] - 233s 2s/step - loss: 0.6929 - binary_accuracy: 0.5105 - val_loss: 0.6926 - val_binary_accu
Epoch 7/10
94/94 [=====] - 231s 2s/step - loss: 0.6926 - binary_accuracy: 0.5116 - val_loss: 0.6924 - val_binary_accu
Epoch 8/10
94/94 [=====] - 229s 2s/step - loss: 0.6924 - binary_accuracy: 0.5217 - val_loss: 0.6923 - val_binary_accu
Epoch 9/10
94/94 [=====] - 230s 2s/step - loss: 0.6922 - binary_accuracy: 0.5209 - val_loss: 0.6919 - val_binary_accu
Epoch 10/10
94/94 [=====] - 230s 2s/step - loss: 0.6920 - binary_accuracy: 0.5264 - val_loss: 0.6917 - val_binary_accu
63/63 - 6s - loss: 0.6917 - binary_accuracy: 0.5355 - 6s/epoch - 89ms/step
Noisy Data - Test Accuracy: 0.5355, Test Loss: 0.6917
Noisy Data - Training Accuracy: 0.5264

```

```
plot_metrics(history_noisy, 'binary_accuracy', 'QCNN Accuracy on Noisy Data - 4x4 Binary Images')
```



# مقایسه سایر معیارهای عملکرد

```
y_true = test_labels # برچسب‌های واقعی داده‌های آزمایشی
```

# پیش‌بینی‌های مدل پاک

```
y_pred_clean_bin = clean_model.predict(x_test_tfcirc_clean) # پیش‌بینی بر اساس مدل پاک
```

```
y_pred_clean_bin = (y_pred_clean_bin > 0.5).astype(int) # تبدیل احتمال‌ها به کلاس‌های بلندی
```

# پیش‌بینی‌های مدل نویزی

```
y_pred_noisy_bin = noisy_model.predict(x_test_tfcirc_noisy) # پیش‌بینی بر اساس مدل نویزی
```

```
y_pred_noisy_bin = (y_pred_noisy_bin > 0.5).astype(int) # تبدیل احتمال‌ها به کلاس‌های بلندی
```

```

63/63 [=====] - 5s 86ms/step
63/63 [=====] - 6s 90ms/step

```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # وارد کردن متریک‌های مورد نیاز
```

# محاسبه معیارهای مدل پاک

```
accuracy_clean_bin = accuracy_score(y_true, y_pred_clean_bin) # دقت
```

```
precision_clean_bin = precision_score(y_true, y_pred_clean_bin, average='weighted') # دقت مثبت
```

```
recall_clean_bin = recall_score(y_true, y_pred_clean_bin, average='weighted') # یادآوری
```

```
f1_clean_bin = f1_score(y_true, y_pred_clean_bin, average='weighted') # امتیاز F1
```

# چاپ معیارهای داده‌های پاک

```
print("\nClean Data Metrics:")
```

```
print(f"Accuracy: {accuracy_clean_bin:.4f}") # چاپ دقت
```

```
print(f"Precision: {precision_clean_bin:.4f}") # چاپ دقت مثبت
print(f"Recall: {recall_clean_bin:.4f}") # چاپ یادآوری
print(f"F1 Score: {f1_clean_bin:.4f}") # چاپ امتیاز F1
```



```
Clean Data Metrics:
Accuracy: 0.7005
Precision: 0.7015
Recall: 0.7005
F1 Score: 0.7001
```

```
# محاسبه معیارهای مدل نویزی
accuracy_noisy_bin = accuracy_score(y_true, y_pred_noisy_bin) # دقت
precision_noisy_bin = precision_score(y_true, y_pred_noisy_bin, average='weighted') # دقت مثبت
recall_noisy_bin = recall_score(y_true, y_pred_noisy_bin, average='weighted') # یادآوری
f1_noisy_bin = f1_score(y_true, y_pred_noisy_bin, average='weighted') # امتیاز F1
```

```
# چاپ معیارهای داده‌های نویزی
print("\nNoisy Data Metrics:")
print(f"Accuracy: {accuracy_noisy_bin:.4f}") # چاپ دقت
print(f"Precision: {precision_noisy_bin:.4f}") # چاپ دقت مثبت
print(f"Recall: {recall_noisy_bin:.4f}") # چاپ یادآوری
print(f"F1 Score: {f1_noisy_bin:.4f}") # چاپ امتیاز F1
```



```
Noisy Data Metrics:
Accuracy: 0.5000
Precision: 0.2500
Recall: 0.5000
F1 Score: 0.3333
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ;
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
import numpy as np
import matplotlib.pyplot as plt # برای رسم نمودارها وارد شده است matplotlib اطمینان از اینکه
```

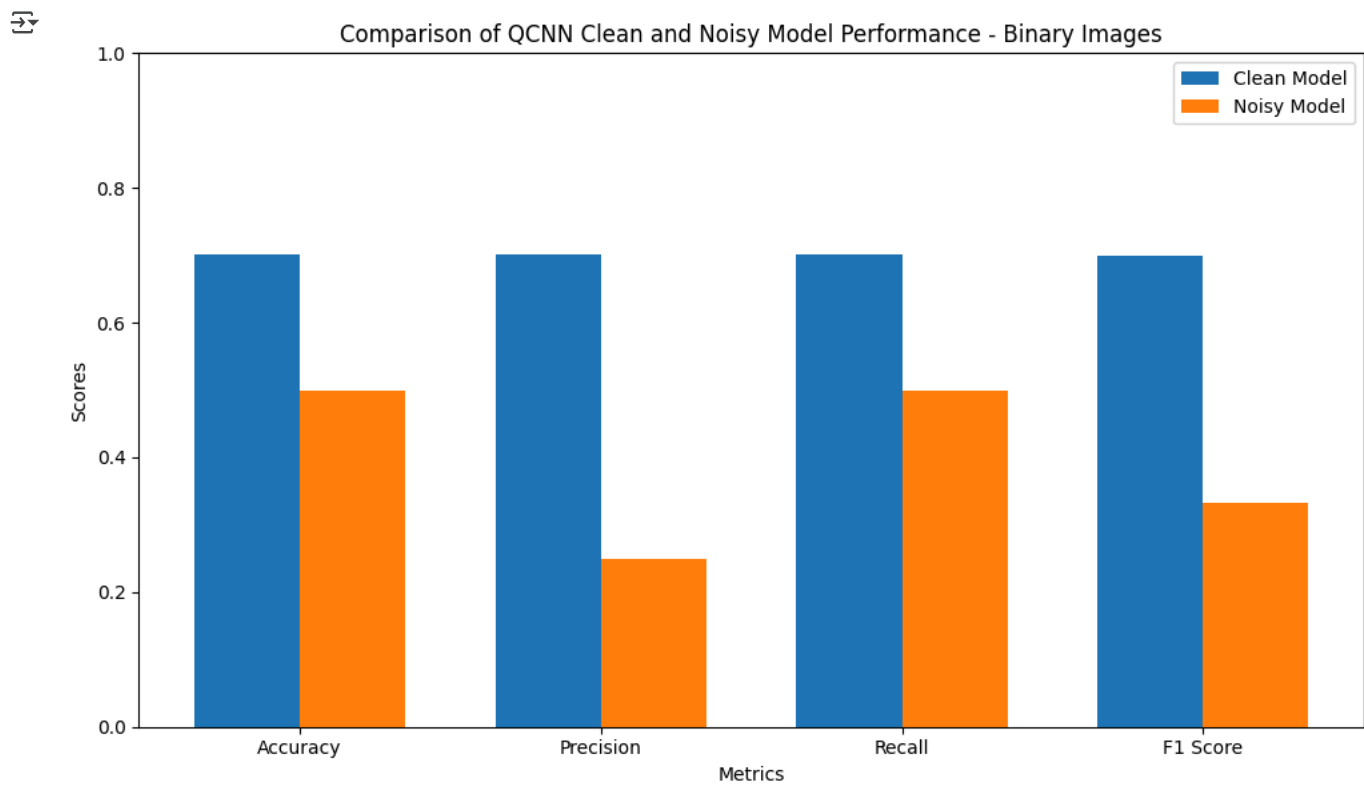
```
# تعریف معیارها و مقادیر
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score'] # نام معیارها
clean_values = [accuracy_clean_bin, precision_clean_bin, recall_clean_bin, f1_clean_bin] # مقادیر مدل پاک
noisy_values = [accuracy_noisy_bin, precision_noisy_bin, recall_noisy_bin, f1_noisy_bin] # مقادیر مدل نویزی
```

```
# موقعیت برجسبها
x = np.arange(len(metrics)) # موقعیت برجسبها
width = 0.35 # عرض میله‌ها
```

```
# ایجاد نمودار
fig, ax = plt.subplots(figsize=(10, 6)) # سایز نمودار
rects1 = ax.bar(x - width/2, clean_values, width, label='Clean Model') # میله‌های مدل پاک
rects2 = ax.bar(x + width/2, noisy_values, width, label='Noisy Model') # میله‌های مدل نویزی
```

```
# تنظیمات محورهای نمودار
ax.set_xlabel('Metrics') # x برجسب محور
ax.set_ylabel('Scores') # y برجسب محور
ax.set_title('Comparison of QCNN Clean and Noisy Model Performance - Binary Images') # عنوان نمودار
ax.set_xticks(x) # x تنظیمات برجسب‌های محور
ax.set_xticklabels(metrics) # برجسب‌های معیارها
ax.legend() # نمایش لژیند
ax.set_ylim(0, 1) # از 0 تا 1 y محدوده ثابت محور
```

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



In the bar plot above it is evident that this QCNN model had a higher robustness to noise compared to some classical CNNs