

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

```
# وارد کردن کتابخانه‌های لازم
import tensorflow as tf # کتابخانه TensorFlow برای یادگیری عمیق
import tensorflow_quantum as tfq # کتابخانه TensorFlow Quantum برای یادگیری ماشین کوانتومی
from tensorflow.keras.datasets import fashion_mnist # مجموعه داده Fashion MNIST برای آزمایش مدل
from tensorflow.keras import regularizers # ابزارهای منظم‌سازی در Keras
import cirq # کتابخانه برای کار با مدارهای کوانتومی
import sympy # کتابخانه برای محاسبات نمادین
import numpy as np # کتابخانه برای کار با آرایه‌ها و محاسبات عددی
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # معیارهای ارزیابی مدل


# برای نمایش نمودارها به صورت آنلاین Jupyter Notebook تنظیمات
%matplotlib inline
import matplotlib.pyplot as plt # کتابخانه برای رسم نمودارها
from cirq.contrib.svg import SVGCircuit # برای نمایش مدارهای کوانتومی به صورت SVG
```

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

```
# نرمال‌سازی تصاویر
train_images = train_images / 255.0 # [1, 0] به بازه
test_images = test_images / 255.0 # انجام همان نرمال‌سازی برای داده‌های آزمایشی
```

```
# (تصاویر خاکستری: یک کانال) QCNN افزودن بعد کانال برای سازگاری با خط لوله
train_images = train_images[..., np.newaxis] # انتهای آرایه تصاویر آموزشی
test_images = test_images[..., np.newaxis] # انجام همان کار برای تصاویر آزمایشی
```

```
# چاپ شکل داده‌های آموزشی و آزمایشی قبل از ایجاد زیرمجموعه
print("Training Data Shape (Before Subset):", train_images.shape)
print("Testing Data Shape (Before Subset):", test_images.shape)
```

 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 (Before Subset): (60000, 28, 28, 1)
Testing Data Shape (Before Subset): (10000, 28, 28, 1)

```
# تعداد نمونه‌هایی که در زیرمجموعه‌ها خواهند 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 نمونه‌ها در هر کلاس آموزشی
num_test_per_class = num_test // 10 # تعداد نمونه‌ها در هر کلاس آزمایشی
```

```
# لیست برای زیرمجموعه‌ها
train_images_subset = []
train_labels_subset = []
test_images_subset = []
test_labels_subset = []
```

```
# تکرار بر روی هر کلاس و نمونه‌گیری برابر
for i in range(10):
    class_indices_train = np.where(train_labels == i)[0] # پیدا کردن ایندکس‌های کلاس
    class_indices_test = np.where(test_labels == i)[0] # پیدا کردن ایندکس‌های کلاس
```

```
# نمونه‌گیری تصادفی از تصاویر در هر کلاس
sampled_train_indices = np.random.choice(class_indices_train, num_train_per_class, replace=False) # نمونه‌گیری بدون جایگزینی
```

```

sampled_test_indices = np.random.choice(class_indices_test, num_test_per_class, replace=False) # نمونه‌گیری بدون جایگزینی

# افزودن به لیست
train_images_subset.append(train_images[sampled_train_indices]) # افزودن تصاویر آموزشی نمونه‌گیری شده
train_labels_subset.append(train_labels[sampled_train_indices]) # افزودن برچسب‌های آموزشی نمونه‌گیری شده
test_images_subset.append(test_images[sampled_test_indices]) # افزودن تصاویر آزمایشی نمونه‌گیری شده
test_labels_subset.append(test_labels[sampled_test_indices]) # افزودن برچسب‌های آزمایشی نمونه‌گیری شده

# تبدیل لیست‌ها به آرایه‌ها
train_images_subset = np.concatenate(train_images_subset) # ترکیب لیست تصاویر آموزشی به یک آرایه
train_labels_subset = np.concatenate(train_labels_subset) # ترکیب لیست برچسب‌های آموزشی به یک آرایه
test_images_subset = np.concatenate(test_images_subset) # ترکیب لیست تصاویر آزمایشی به یک آرایه
test_labels_subset = np.concatenate(test_labels_subset) # ترکیب لیست برچسب‌های آزمایشی به یک آرایه

# چاپ شکل داده‌های آموزشی و آزمایشی پس از ایجاد زیرمجموعه
print("Training Data Shape (After Subset):", train_images_subset.shape)
print("Testing Data Shape (After Subset):", test_images_subset.shape)

# چاپ کلاس‌های منحصر به فرد برای تولید اینکه همه 10 کلاس شامل شده‌اند
print("Unique Train Labels:", np.unique(train_labels_subset))
print("Unique Test Labels:", np.unique(test_labels_subset))

↩ Training Data Shape (After Subset): (6000, 28, 28, 1)
Testing Data Shape (After Subset): (1000, 28, 28, 1)
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]

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

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

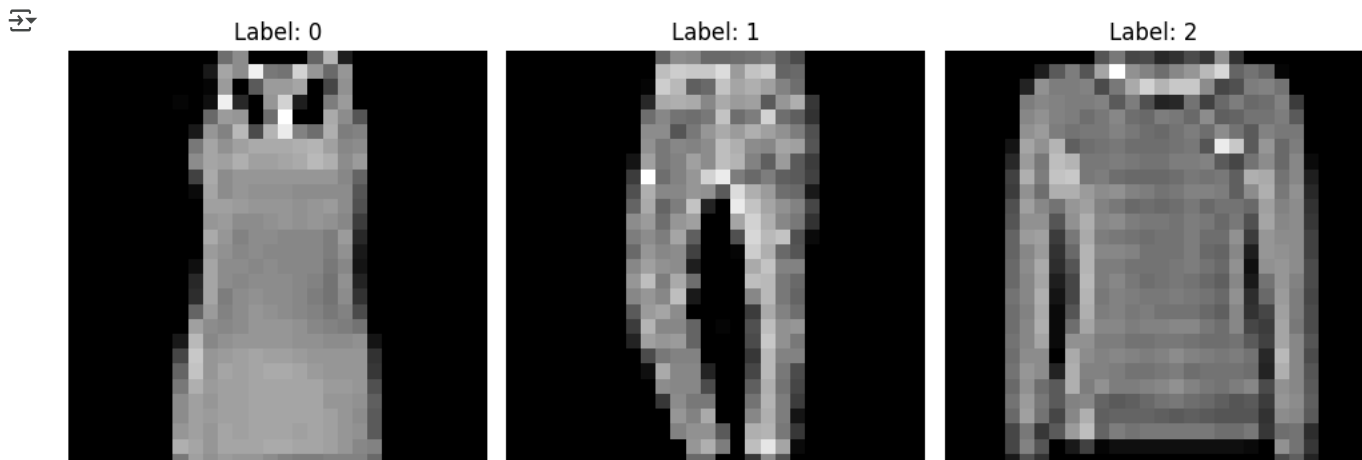
    # انتخاب یک تصویر از هر کلاس منتخب
    for class_label in selected_classes:
        class_indices = np.where(labels == class_label)[0] # پیدا کردن ایندکس‌های کلاس
        selected_images.append(images[class_indices[0]]) # انتخاب اولین تصویر از کلاس
        selected_labels.append(labels[class_indices[0]])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

    # تبدیل لیست‌ها به آرایه‌ها
    selected_images = np.array(selected_images)
    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') # نمایش تصویر با حذف بعد کانال
        ax.set_title(f'Label: {selected_labels[i]}') # عنوان بر اساس برچسب
        ax.axis('off') # خاموش کردن محور
    plt.tight_layout() # تنظیمات نهایی
    plt.show() # نمایش تصاویر

# نمایش داده‌های نمونه از مجموعه داده پاک
display_images(train_images_subset, train_labels_subset, num_images=3)

```



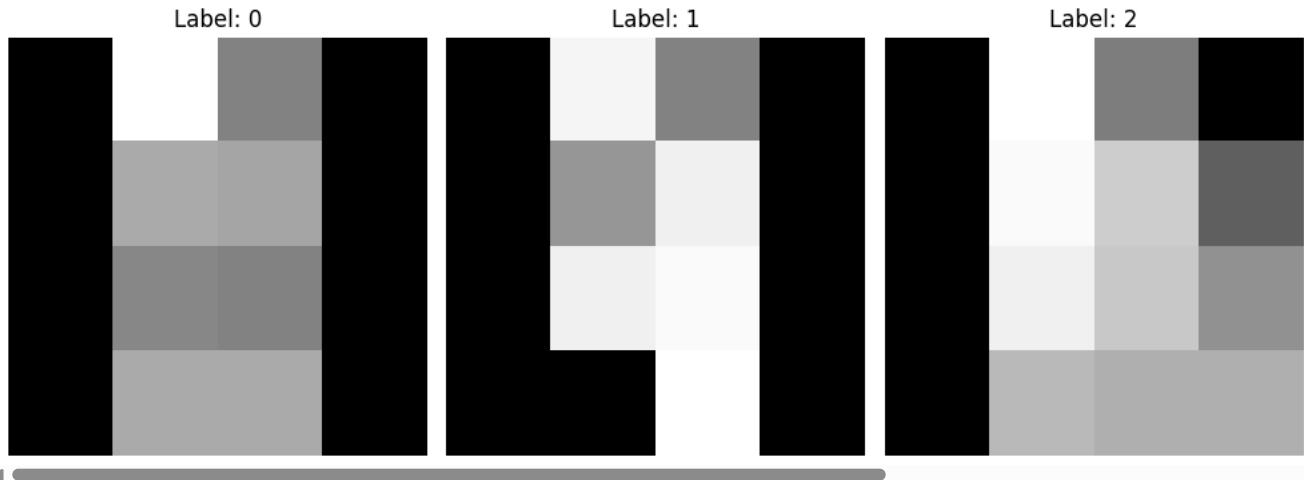
```
# تابعی برای تغییر اندازه تصاویر
def resize_images(images, target_size=(4, 4)):
    return tf.image.resize(images, target_size).numpy() # تغییر اندازه تصاویر به اندازه هدف و تبدیل به آرایه NumPy

# تغییر اندازه داده‌های آموزشی و آزمایشی
train_data_clean = resize_images(train_images_subset) # تغییر اندازه تصاویر آموزشی
test_data_clean = resize_images(test_images_subset) # تغییر اندازه تصاویر آزمایشی

# چاپ شکل داده‌های آموزشی و آزمایشی پس از تغییر اندازه
print("Resized Training Data Shape:", train_data_clean.shape)
print("Resized Testing Data Shape:", test_data_clean.shape)

# نمایش داده‌های نمونه از داده‌های آموزشی تغییر اندازه شده
display_images(train_data_clean, train_labels_subset, num_images=3)
```

Resized Training Data Shape: (6000, 4, 4, 1)
Resized Testing Data Shape: (1000, 4, 4, 1)



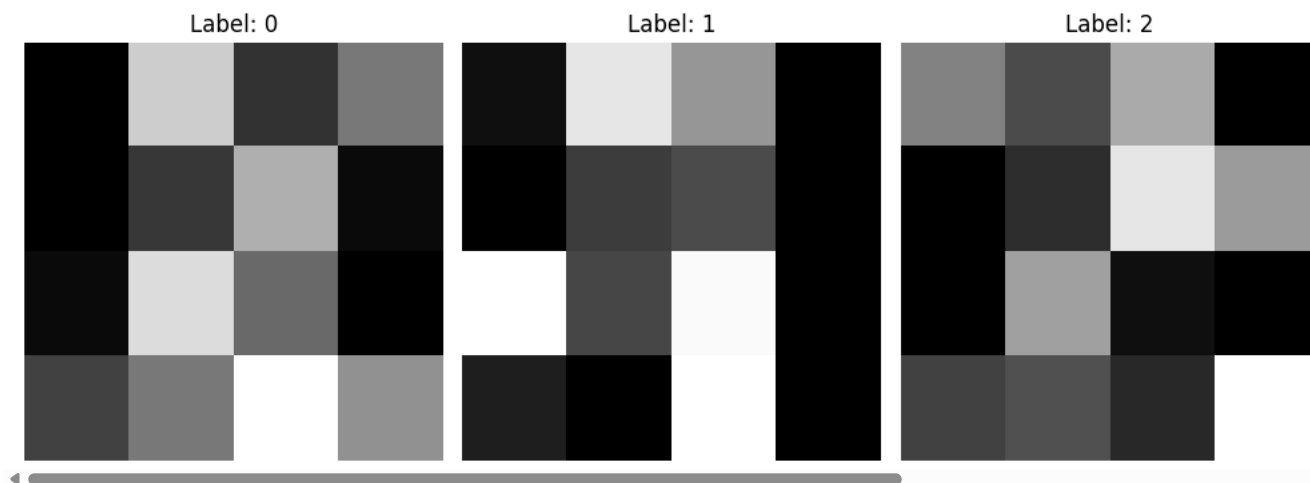
```
# تابعی برای افزودن نویز گوسی به تصاویر
def add_gaussian_noise(images, mean=0.0, severity=1):
    severity_levels = [0.08, 0.12, 0.18, 0.26, 0.38] # سطوح شدت نویز
    stddev = severity_levels[severity - 1] # دریافت انحراف استاندارد بر اساس شدت (1 تا 5)
    noise = np.random.normal(mean, stddev, images.shape) # 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

# 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

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

# نمایش داده‌های نمونه از داده‌های آموزشی با نویز
display_images(train_data_noisy, train_labels_subset, num_images=3)
```

Noisy Training Data Shape: (6000, 4, 4, 1)
Noisy Testing Data Shape: (1000, 4, 4, 1)



```
# تعریف آستانه برای فعال‌سازی حالت کویت‌ها  
THRESHOLD = 0.5
```

```
# ایجاد کدگذاری بلیزی برای داده‌های آموزشی و آزمایشی  
x_train_bin = np.array(train_data_clean > THRESHOLD, dtype=np.float32) # کدگذاری بلیزی برای داده‌های آموزشی پاک  
x_test_bin = np.array(test_data_clean > THRESHOLD, dtype=np.float32) # کدگذاری بلیزی برای داده‌های آزمایشی پاک  
  
x_train_noisy_bin = np.array(train_data_noisy > THRESHOLD, dtype=np.float32) # کدگذاری بلیزی برای داده‌های آموزشی با نویز  
x_test_noisy_bin = np.array(test_data_noisy > THRESHOLD, dtype=np.float32) # کدگذاری بلیزی برای داده‌های آزمایشی با نویز
```

```
# تابعی برای تبدیل تصاویر کلاسیک به داده‌های کوانتومی  
def convert_to_circuit(image):  
    values = np.ndarray.flatten(image) # تبدیل تصویر به یک آرایه یک بعدی  
    qubits = cirq.GridQubit.rect(4, 4) # ایجاد یک شبکه از کویت‌ها به ابعاد 4 در 4  
    circuit = cirq.Circuit() # ایجاد یک مدار کوانتومی خالی  
    for i, value in enumerate(values):  
        if value: # اگر مقدار پیکسل برابر با 1 باشد (فعال)  
            circuit.append(cirq.X(qubits[i])) # به کویت مربوطه X اضافه کردن گیت  
    return circuit # بازگشت به مدار کوانتومی
```

```
# تبدیل تصاویر بلیزی تمیز به مدارهای کوانتومی  
train_clean_circ = [convert_to_circuit(x) for x in x_train_bin] # برای داده‌های آموزشی تمیز  
test_clean_circ = [convert_to_circuit(x) for x in x_test_bin] # برای داده‌های آزمایشی تمیز
```

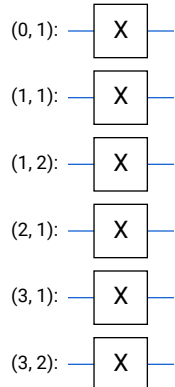
```
# تبدیل تصاویر بلیزی با نویز به مدارهای کوانتومی  
train_noisy_circ = [convert_to_circuit(x) for x in x_train_noisy_bin] # برای داده‌های آموزشی با نویز  
test_noisy_circ = [convert_to_circuit(x) for x in x_test_noisy_bin] # برای داده‌های آزمایشی با نویز
```

```
# نمایش کویت‌هایی که روی آن‌ها گیت قرار گرفته است  
SVGCircuit(train_clean_circ[0]) # نمایش مدار کوانتومی اولین تصویر تمیز
```

```

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.

```



تبدیل مدارهای cirq به TensorFlow Quantum

```

train_clean_tfqcirc = tfq.convert_to_tensor(train_clean_circ) # تبدیل مدارهای آموزشی تمیز به تانسور
test_clean_tfqcirc = tfq.convert_to_tensor(test_clean_circ) # تبدیل مدارهای آزمایشی تمیز به تانسور

```

```

train_noisy_tfqcirc = tfq.convert_to_tensor(train_noisy_circ) # تبدیل مدارهای آموزشی با نویز به تانسور
test_noisy_tfqcirc = tfq.convert_to_tensor(test_noisy_circ) # تبدیل مدارهای آزمایشی با نویز به تانسور

```

کلاس برای ایجاد لایه‌های مدار کوانتومی

```
class CircuitLayerBuilder:
```

```

    def __init__(self, data_qubits, readout):
        self.data_qubits = data_qubits # کویت‌های داده
        self.readout = readout # روش خوانش

```

```
    def add_layer(self, circuit, prefix):
```

بلایری QCNN مشابه مدل ZZ و XX اعمال درهم‌تنیدگی

```

    for i in range(len(self.data_qubits) - 1):
        circuit.append(cirq.XX(self.data_qubits[i], self.data_qubits[i + 1])) # درهم‌تنیدگی XX
        circuit.append(cirq.ZZ(self.data_qubits[i], self.data_qubits[i + 1])) # درهم‌تنیدگی ZZ

```

اعمال گیت‌های کوانتومی پارامتری به هر کویت

```

    for i, qubit in enumerate(self.data_qubits):
        # تعریف پارامترها و افزودن تنظیم‌کننده
        #lambda_ = sympy.Symbol(f'{prefix}_lambda_{i}', regularizer=regularizers.l2(0.01)) # برای استفاده‌های آینده
        theta = sympy.Symbol(f'{prefix}_theta_{i}', regularizer=regularizers.l2(0.01)) # پارامتر theta
        phi = sympy.Symbol(f'{prefix}_phi_{i}', regularizer=regularizers.l2(0.01)) # پارامتر phi

```

افزودن چرخشی پارامتری

```

    #circuit.append(cirq.rx(lambda_).on(qubit)) # گیت RX (غیرفعال)
    circuit.append(cirq.ry(theta).on(qubit)) # گیت RY با پارامتر theta
    circuit.append(cirq.rz(phi).on(qubit)) # گیت RZ با پارامتر phi
    circuit.append(cirq.ry(theta).on(qubit)) # گیت RY دوباره با پارامتر theta
    circuit.append(cirq.rz(phi).on(qubit)) # گیت RZ دوباره با پارامتر phi

```

جمع مبتنی بر اندازه‌گیری (غیرفعال)

استفاده نمی‌شود زیرا دقت را کاهش می‌دهد زیرا اندازه داده‌ها کوچک است

```

    for i in range(0, len(self.data_qubits), 4):
        #circuit.append(cirq.CNOT(self.data_qubits[i], self.data_qubits[i + 1]))
        #circuit.append(cirq.measure(self.data_qubits[i + 1]))

```

به‌روزرسانی کویت‌ها برای نشان دادن تجمع (غیرفعال)

```
#self.data_qubits = [self.data_qubits[i] for i in range(0, len(self.data_qubits), 2)]
```

تعریف کویت‌ها و کویت خوانش

```

data_qubits = cirq.GridQubit.rect(2, 2) # از کویت‌ها 2x2 ایجاد یک شبکه 2
readout = cirq.GridQubit(-1, -1) # تعریف کویت خوانش

```

ایجاد یک مدار نمونه


```

tfq.layers.PQC(model_circuit, model_readout), # با مدار کوانتومی و خوانش PQC لایه #
tf.keras.layers.Dense(8, activation='relu', kernel_regularizer=regularizers.l2(0.01)), # Dense لایه # با 8 نورون و تابع فعال‌سازی ReLU
tf.keras.layers.Dense(10, activation='softmax', kernel_regularizer=regularizers.l2(0.01)) # Dense لایه # (10 کلاس)
])

```

```

# کامپایل کردن مدل
clean_model.compile(optimizer='adam', # استفاده از بهینه‌ساز Adam
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(), # تابع هزینه برای طبقه‌بندی چندکلاسی
                    metrics=['accuracy']) # معیار دقت

```

```

# چاپ خلاصه مدل
print(clean_model.summary()) # نمایش ساختار مدل

```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
pqc (PQC)	(None, 16)	64
dense (Dense)	(None, 8)	136
dense_1 (Dense)	(None, 10)	90
=====		
Total params: 290 (1.13 KB)		
Trainable params: 290 (1.13 KB)		
Non-trainable params: 0 (0.00 Byte)		
None		

```

# در اینجا نمایش بصری مدل مدار کوانتومی آمده است:
SVGCircuit(model_circuit)

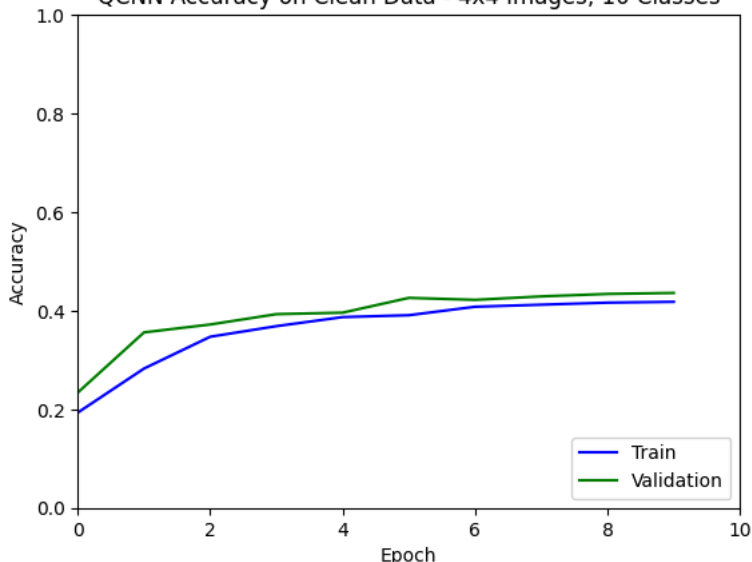
```



```
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.3870 - val_loss: 1.7850 - val_accuracy: 0.3966
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.3908 - val_loss: 1.7546 - val_accuracy: 0.4266
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.4080 - val_loss: 1.7307 - val_accuracy: 0.4226
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.4122 - val_loss: 1.7152 - val_accuracy: 0.4296
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.4163 - val_loss: 1.7041 - val_accuracy: 0.4346
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager.findfont: Font family 'Arial' not found. accuracy: 0.4180 - val_loss: 1.6983 - val_accuracy: 0.4366
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.
```

```
def plot_metrics(history, metric_name, title, ylim=1, xlim=10):
    plt.title(title) # عنوان نمودار
    plt.ylim(0, ylim) # تنظیم محدویت محور عمودی
    plt.xlim(0, xlim) # تنظیم محدویت محور افقی
    plt.plot(history.history[metric_name], color='blue', label='Train') # نمودار دقت آموزش
    plt.plot(history.history['val_' + metric_name], color='green', label='Validation') # نمودار دقت اعتبارسنجی
    plt.legend(loc='lower right') # مکان قرارگیری افسانه
    plt.xlabel("Epoch") # برچسب محور افقی
    plt.ylabel("Accuracy") # برچسب محور عمودی
```

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



ساخت و کامپایل یک مدل دیگر برای داده‌های نویزی

کامپایل کردن مدل

جاپ خلاصه مدل

[illegible]



QCNN Accuracy on Noisy Data - 4x4 Images, 10 Classes

1.0

حالا برای مقایسه دیگر معیارها بین هر دو مدل

استخراج برچسب‌های واقعی

y_true = test_labels_subset

پیش‌بینی‌های مدل تمیز

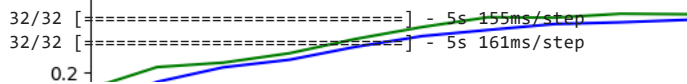
y_pred_clean = clean_model.predict(test_clean_tfirc) # پیش‌بینی بر اساس داده‌های تمیز

y_pred_clean = np.argmax(y_pred_clean, axis=-1) # تبدیل احتمالات به ایندکس‌های کلاس

پیش‌بینی‌های مدل نویزی

y_pred_noisy = noisy_model.predict(test_noisy_tfirc) # پیش‌بینی بر اساس داده‌های نویزی

y_pred_noisy = np.argmax(y_pred_noisy, axis=-1) # تبدیل احتمالات به ایندکس‌های کلاس



1_theta_0

Rz(layer1_phi_0)

محاسبه معیارهای مدل تمیز

accuracy_clean = accuracy_score(y_true, y_pred_clean) # دقت

precision_clean = precision_score(y_true, y_pred_clean, average='weighted') # دقت (Precision)

recall_clean = recall_score(y_true, y_pred_clean, average='weighted') # یادآوری (Recall)

f1_clean = f1_score(y_true, y_pred_clean, average='weighted') # نمره F1

چاپ معیارهای داده‌های تمیز

print("\nClean Data Metrics:")

print(f"Accuracy: {accuracy_clean:.4f}")

print(f"Precision: {precision_clean:.4f}")

print(f"Recall: {recall_clean:.4f}")

print(f"F1 Score: {f1_clean:.4f}")



(1,2):

Clean Data Metrics:

Accuracy: 0.4360

Precision: 0.4498

Recall: 0.4360

F1 Score: 0.4054

(2,0):

محاسبه معیارهای مدل نویزی

accuracy_noisy = accuracy_score(y_true, y_pred_noisy) # دقت

precision_noisy = precision_score(y_true, y_pred_noisy, average='weighted') # دقت (Precision)

recall_noisy = recall_score(y_true, y_pred_noisy, average='weighted') # یادآوری (Recall)

f1_noisy = f1_score(y_true, y_pred_noisy, average='weighted') # نمره F1

چاپ معیارهای داده‌های نویزی

print("\nNoisy Data Metrics:")

print(f"Accuracy: {accuracy_noisy:.4f}")

print(f"Precision: {precision_noisy:.4f}")

print(f"Recall: {recall_noisy:.4f}")

print(f"F1 Score: {f1_noisy:.4f}")