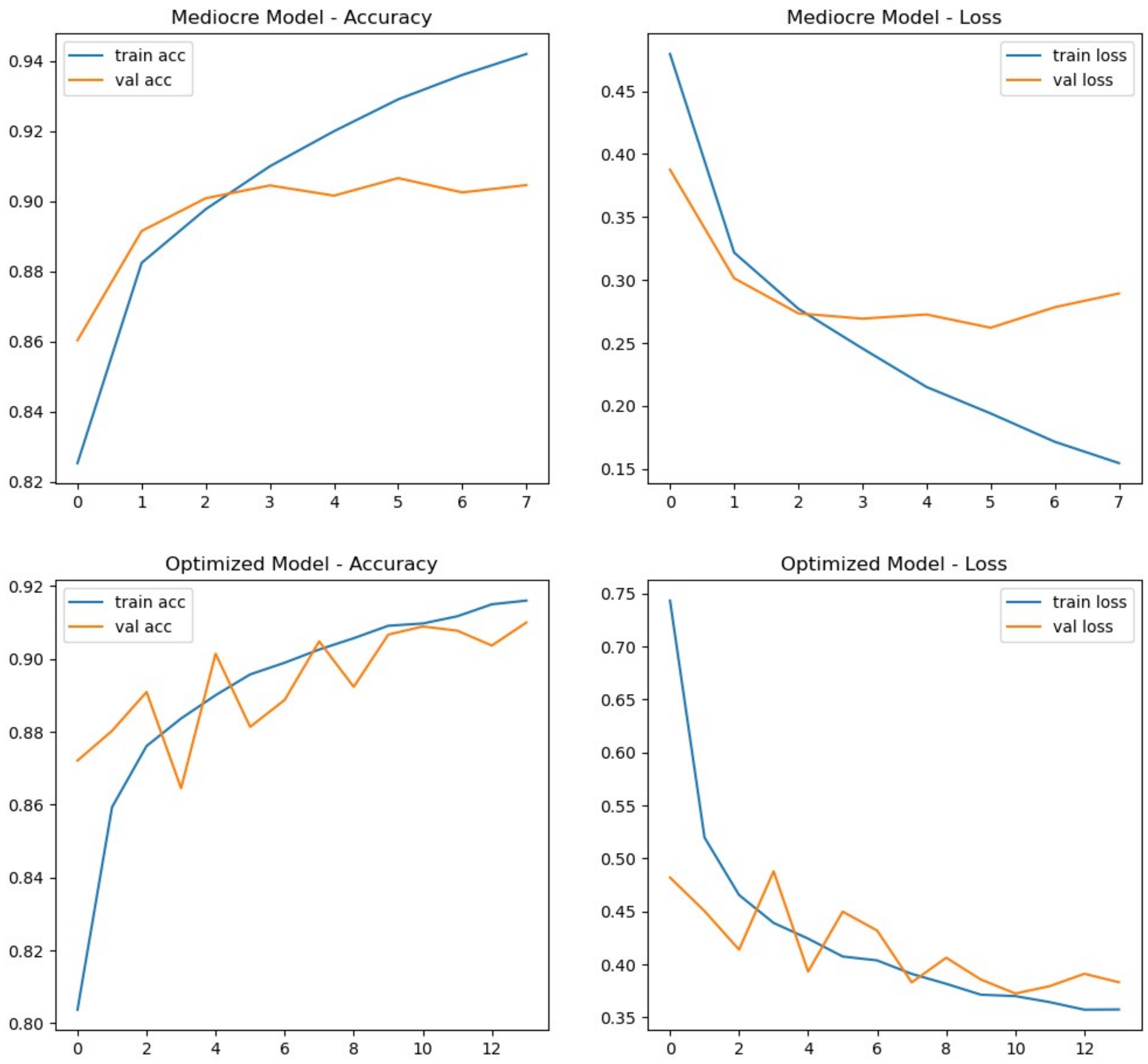
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def build_medicore_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
28, 1)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model


def build_optimized_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
28, 1)),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model


def train_test_evaluate_models():
    (x_train, y_train), (x_test, y_test) =
keras.datasets.fashion_mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0
    x_train, x_test = x_train[..., np.newaxis], x_test[...,
np.newaxis]
```

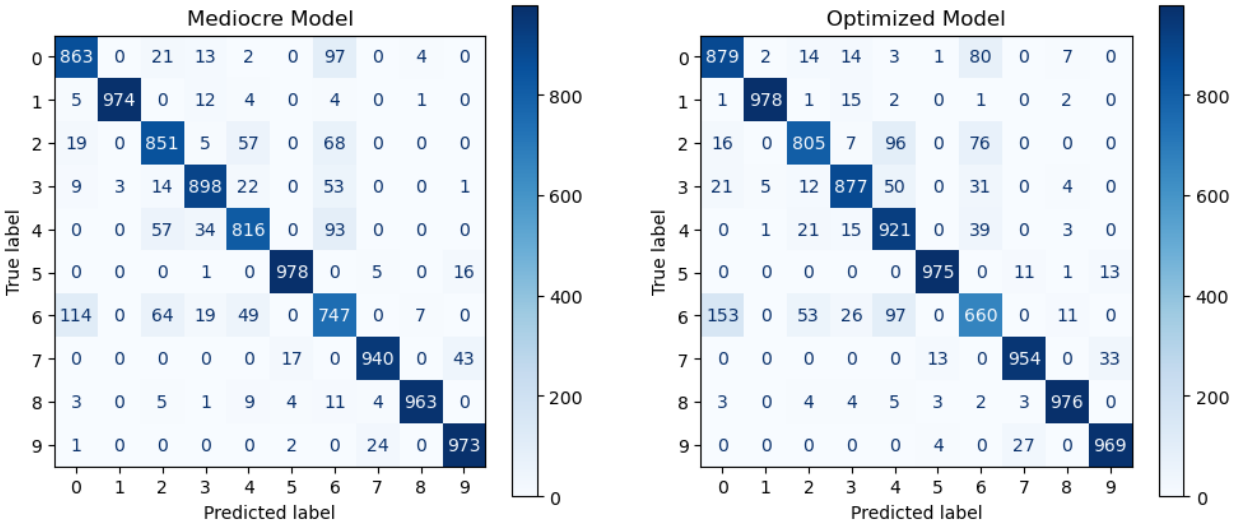
```
    model1 = build_medicore_model()
    model2 = build_optimized_model()
    early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

    history1 = model1.fit(x_train, y_train, epochs=8,
validation_split=0.2, callbacks=[early_stopping])
    history2 = model2.fit(x_train, y_train, epochs=16,
validation_split=0.2, callbacks=[early_stopping])

    test_loss1, test_acc1 = model1.evaluate(x_test, y_test)
    test_loss2, test_acc2 = model2.evaluate(x_test, y_test)
    print(f"Mediocre Model Accuracy: {test_acc1}, Optimized Model
Accuracy: {test_acc2}")

    test_loss1, test_acc1 = model1.evaluate(x_test, y_test)
    test_loss2, test_acc2 = model2.evaluate(x_test, y_test)
    print(f"Mediocre Model Accuracy: {test_acc1}, Optimized Model
Accuracy: {test_acc2}")


    def plot_curves(history, title):
        plt.figure(figsize=(12, 5))
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='train acc')
        plt.plot(history.history['val_accuracy'], label='val acc')
        plt.title(f'{title} - Accuracy')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='train loss')
        plt.plot(history.history['val_loss'], label='val loss')
        plt.title(f'{title} - Loss')
        plt.legend()
        plt.show()

    plot_curves(history1, "Mediocre Model")
    plot_curves(history2, "Optimized Model")

    y_pred1 = np.argmax(model1.predict(x_test), axis=1)
    y_pred2 = np.argmax(model2.predict(x_test), axis=1)

    cm1 = confusion_matrix(y_test, y_pred1)
    cm2 = confusion_matrix(y_test, y_pred2)

    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ConfusionMatrixDisplay(cm1).plot(ax=ax[0], cmap='Blues')
    ax[0].set_title('Mediocre Model')
    ConfusionMatrixDisplay(cm2).plot(ax=ax[1], cmap='Blues')
    ax[1].set_title('Optimized Model')
    plt.show()

train_test_evaluate_models()
```

```
c:\Users\sajee\anaconda3\Lib\site-packages\keras\src\layers\
convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.7593 -
loss: 0.6664 - val_accuracy: 0.8603 - val_loss: 0.3877
Epoch 2/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.8757 -
loss: 0.3367 - val_accuracy: 0.8915 - val_loss: 0.3015
Epoch 3/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.8973 -
loss: 0.2770 - val_accuracy: 0.9008 - val_loss: 0.2735
Epoch 4/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.9104 -
loss: 0.2415 - val_accuracy: 0.9045 - val_loss: 0.2693
Epoch 5/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 23s 16ms/step - accuracy: 0.9202 -
loss: 0.2123 - val_accuracy: 0.9016 - val_loss: 0.2726
Epoch 6/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 53s 23ms/step - accuracy: 0.9304 -
loss: 0.1898 - val_accuracy: 0.9066 - val_loss: 0.2621
Epoch 7/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.9379 -
loss: 0.1696 - val_accuracy: 0.9025 - val_loss: 0.2784
Epoch 8/8
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.9434 - loss:
0.1510 - val_accuracy: 0.9046 - val_loss: 0.2893
Epoch 1/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 30s 18ms/step - accuracy: 0.7511 -
loss: 0.9326 - val_accuracy: 0.8721 - val_loss: 0.4820
Epoch 2/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.8539 -
loss: 0.5381 - val_accuracy: 0.8802 - val_loss: 0.4504
Epoch 3/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 27s 18ms/step - accuracy: 0.8749 -
loss: 0.4691 - val_accuracy: 0.8909 - val_loss: 0.4140
Epoch 4/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.8864 -
loss: 0.4318 - val_accuracy: 0.8645 - val_loss: 0.4879
```

```
Epoch 5/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 24s 16ms/step - accuracy: 0.8907 -
loss: 0.4195 - val_accuracy: 0.9014 - val_loss: 0.3932
Epoch 6/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 29s 19ms/step - accuracy: 0.8987 -
loss: 0.3979 - val_accuracy: 0.8813 - val_loss: 0.4499
Epoch 7/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 27s 18ms/step - accuracy: 0.9025 -
loss: 0.4001 - val_accuracy: 0.8888 - val_loss: 0.4319
Epoch 8/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 29s 19ms/step - accuracy: 0.9076 -
loss: 0.3782 - val_accuracy: 0.9048 - val_loss: 0.3831
Epoch 9/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.9077 -
loss: 0.3745 - val_accuracy: 0.8923 - val_loss: 0.4063
Epoch 10/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.9093 -
loss: 0.3652 - val_accuracy: 0.9067 - val_loss: 0.3858
Epoch 11/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.9110 -
loss: 0.3636 - val_accuracy: 0.9089 - val_loss: 0.3726
Epoch 12/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.9145 -
loss: 0.3520 - val_accuracy: 0.9078 - val_loss: 0.3795
Epoch 13/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 32s 21ms/step - accuracy: 0.9185 -
loss: 0.3459 - val_accuracy: 0.9037 - val_loss: 0.3912
Epoch 14/16
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 28s 19ms/step - accuracy: 0.9200 -
loss: 0.3478 - val_accuracy: 0.9100 - val_loss: 0.3832
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9021 - loss:
0.2831
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8995 - loss:
0.4008
Mediocre Model Accuracy: 0.9003000259399414, Optimized Model Accuracy:
0.899399995803833
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9021 - loss:
0.2831
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8995 - loss:
0.4008
Mediocre Model Accuracy: 0.9003000259399414, Optimized Model Accuracy:
0.899399995803833
```



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
```



Mediocre Model

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	863	0	21	13	2	0	97	0	4	0
1	5	974	0	12	4	0	4	0	1	0
2	19	0	851	5	57	0	68	0	0	0
3	9	3	14	898	22	0	53	0	0	1
4	0	0	57	34	816	0	93	0	0	0
5	0	0	0	1	0	978	0	5	0	16
6	114	0	64	19	49	0	747	0	7	0
7	0	0	0	0	0	17	0	940	0	43
8	3	0	5	1	9	4	11	4	963	0
9	1	0	0	0	0	2	0	24	0	973

Optimized Model

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	879	2	14	14	3	1	80	0	7	0
1	1	978	1	15	2	0	1	0	2	0
2	16	0	805	7	96	0	76	0	0	0
3	21	5	12	877	50	0	31	0	4	0
4	0	1	21	15	921	0	39	0	3	0
5	0	0	0	0	0	975	0	11	1	13
6	153	0	53	26	97	0	660	0	11	0
7	0	0	0	0	0	13	0	954	0	33
8	3	0	4	4	5	3	2	3	976	0
9	0	0	0	0	0	4	0	27	0	969