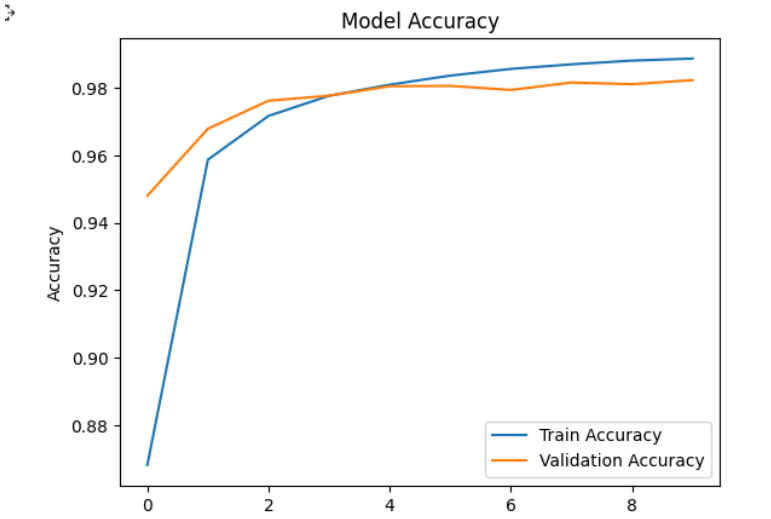
模型訓練 成果:

```
y_test = to_categorical(y_test)

#建構網路
model = Sequential()
#CNN輸入為28*28*1
model.add(Conv2D(32, kernel_size = 3, input_shape = (28,28,1),padding="same", activation = 'relu'))
#池化層
model.add(MaxPooling2D(pool_size = 2))
#攤平
model.add(Flatten())
#全連接層
model.add(Dense(16, activation = 'relu'))
#輸出層
model.add(Dense(10, activation = 'softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train,
                                        batch_size=128,
                                        epochs=10,
                                        verbose=1,
                                        validation_data=(x_test, y_test))
```

```
Epoch 1/10
469/469 [==============================] - 12s 5ms/step - loss: 0.4581 - accuracy: 0.8682 - val_loss: 0.1888 - val_accuracy: 0.9480
Epoch 2/10
469/469 [==============================] - 2s 5ms/step - loss: 0.1472 - accuracy: 0.9587 - val_loss: 0.1065 - val_accuracy: 0.9679
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0981 - accuracy: 0.9717 - val_loss: 0.0816 - val_accuracy: 0.9762
Epoch 4/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0764 - accuracy: 0.9777 - val_loss: 0.0719 - val_accuracy: 0.9777
Epoch 5/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0645 - accuracy: 0.9809 - val_loss: 0.0625 - val_accuracy: 0.9805
Epoch 6/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0555 - accuracy: 0.9837 - val_loss: 0.0620 - val_accuracy: 0.9806
Epoch 7/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0492 - accuracy: 0.9856 - val_loss: 0.0633 - val_accuracy: 0.9794
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0437 - accuracy: 0.9870 - val_loss: 0.0568 - val_accuracy: 0.9816
Epoch 9/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0397 - accuracy: 0.9881 - val_loss: 0.0600 - val_accuracy: 0.9811
Epoch 10/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0362 - accuracy: 0.9887 - val_loss: 0.0532 - val_accuracy: 0.9823
```

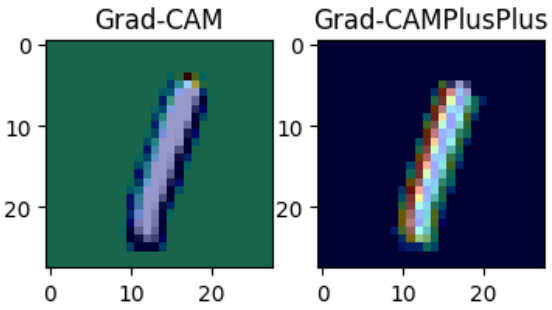
準確率

```
# 绘制训练和验证准确率
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# 绘制训练和验证损失
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

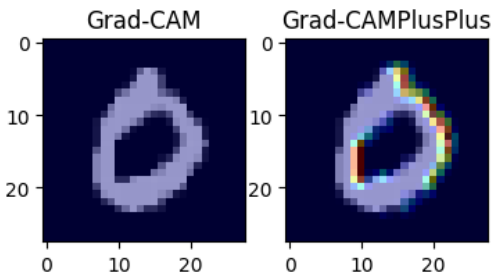


Grad-cam(還有 grad-cam++)圖像

```
[62]  #  选择一个测试图像的索引和文件名
      file_index  =  5     #  替换为你想要的测试图像的索引
      save_name  =  "grad_cam_result.png"     #  替换为你想要的保存文件名

      #  调用  Grad_CAM_savepictures  函数生成结果
      Grad_CAM_savepictures(file_index,  model,  save_name)
```




```
#  选择一个测试图像的索引和文件名
file_index  =  3     #  替换为你想要的测试图像的索引
save_name  =  "grad_cam_result.png"     #  替换为你想要的保存文件名

#  调用  Grad_CAM_savepictures  函数生成结果
Grad_CAM_savepictures(file_index,  model,  save_name)
```




```
#  选择一个测试图像的索引和文件名
file_index  =  4     #  替换为你想要的测试图像的索引
save_name  =  "grad_cam_result.png"     #  替换为你想要的保存文件名

#  调用  Grad_CAM_savepictures  函数生成结果
Grad_CAM_savepictures(file_index,  model,  save_name)
```