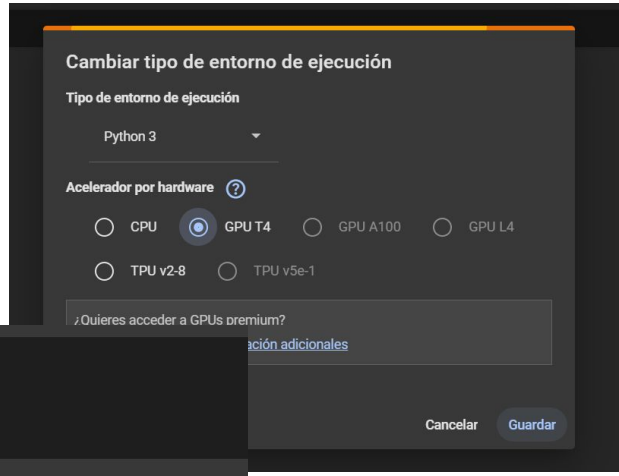
Lab 1: DeepLearning

Mario Ventura

Índice

Crear notebook y seleccionar GPU

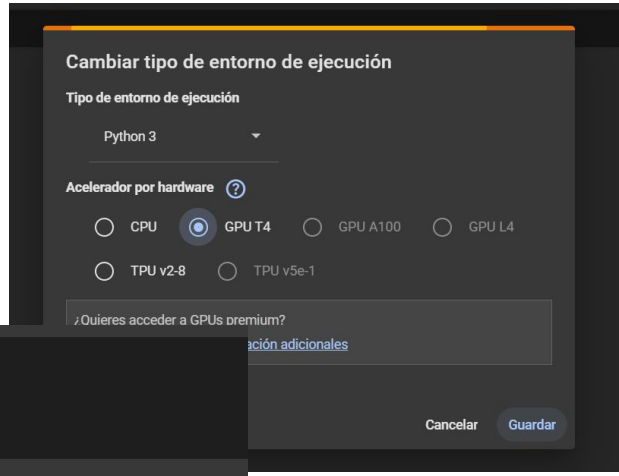
- Colab de Google
- Se ejecuta el modelo inicial




```
[10] model.compile(loss="categorical_crossentropy",
      optimizer="sgd",
      metrics = ['accuracy'])
     model.fit(x_train, y_train, epochs=8)

Epoch 1/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8637 - loss: 0.6158
Epoch 2/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8726 - loss: 0.5582
Epoch 3/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8795 - loss: 0.5098
Epoch 4/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8814 - loss: 0.4838
Epoch 5/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8863 - loss: 0.4570
Epoch 6/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8893 - loss: 0.4392
Epoch 7/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8908 - loss: 0.4213
Epoch 8/8
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8924 - loss: 0.4086
<keras.src.callbacks.history.History at 0x7c1459b0aed0>
```

Crear notebook y seleccionar GPU

- Colab de Google
- Se ejecuta el modelo inicial (8 etapas)

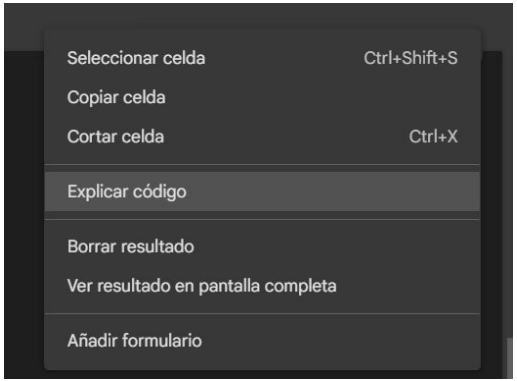


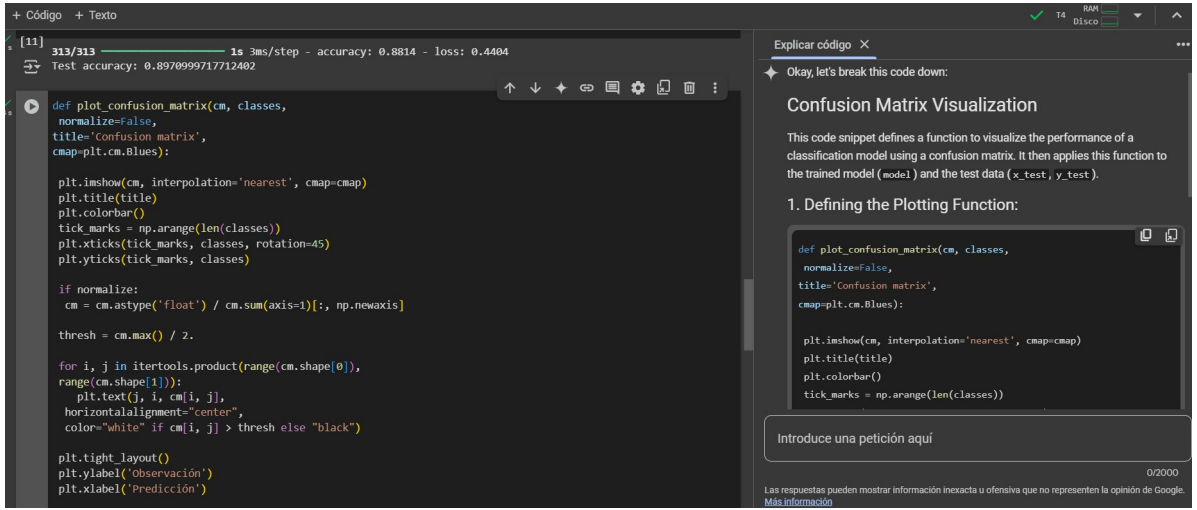

```
[10] model.compile(loss="categorical_crossentropy",
      optimizer="sgd",
      metrics = ['accuracy'])
     model.fit(x_train, y_train, epochs=8)

     Epoch 1/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8637 - loss: 0.6158
     Epoch 2/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8726 - loss: 0.5582
     Epoch 3/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8795 - loss: 0.5098
     Epoch 4/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8814 - loss: 0.4838
     Epoch 5/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8863 - loss: 0.4570
     Epoch 6/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8893 - loss: 0.4392
     Epoch 7/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8908 - loss: 0.4213
     Epoch 8/8
     1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8924 - loss: 0.4086
     <keras.src.callbacks.history.History at 0x7c1459b0aed0>
```

Crear notebook y seleccionar GPU

Utilidades

- Explicar código
- Gemini AI





Nuevo modelo mejorado

Definir, entrenar y evaluar un nuevo modelo que mejore la **accuracy**

ESTRATEGIAS DE MEJORA

1. **Añadir más capas:** Se añaden capas adicionales con neuronas al modelo original.
2. **Añadir más neuronas:** Se añaden más neuronas a las capas existentes.
3. **Añadir más épocas:** En lugar de realizar 5 etapas se realizan 15.

Nuevo modelo mejorado

ORIGINAL

```
[ ]  model.fit(train_images, train_labels, batch_size=100, epochs=5, verbose=1)
```

```
Epoch 1/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5878 - loss: 1.5305
Epoch 2/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9123 - loss: 0.2909
Epoch 3/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9380 - loss: 0.2076
Epoch 4/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9526 - loss: 0.1585
Epoch 5/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9609 - loss: 0.1341
<keras.src.callbacks.history.History at 0x7f8408aa68f0>
```

```
[ ]  test_loss, test_acc = model.evaluate(test_images, test_labels)

     print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9647 - loss: 0.1211
Test accuracy: 0.9710999727249146
```

Nuevo modelo mejorado

MEJORADO

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9875 - loss: 0.0426
Epoch 12/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9884 - loss: 0.0404
Epoch 13/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9891 - loss: 0.0382
Epoch 14/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9875 - loss: 0.0396
Epoch 15/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9887 - loss: 0.0386
<keras.src.callbacks.history.History at 0x7ae47216c250>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9852 - loss: 0.0455
Test accuracy: 0.9894999861717224
```

Original vs Mejorado

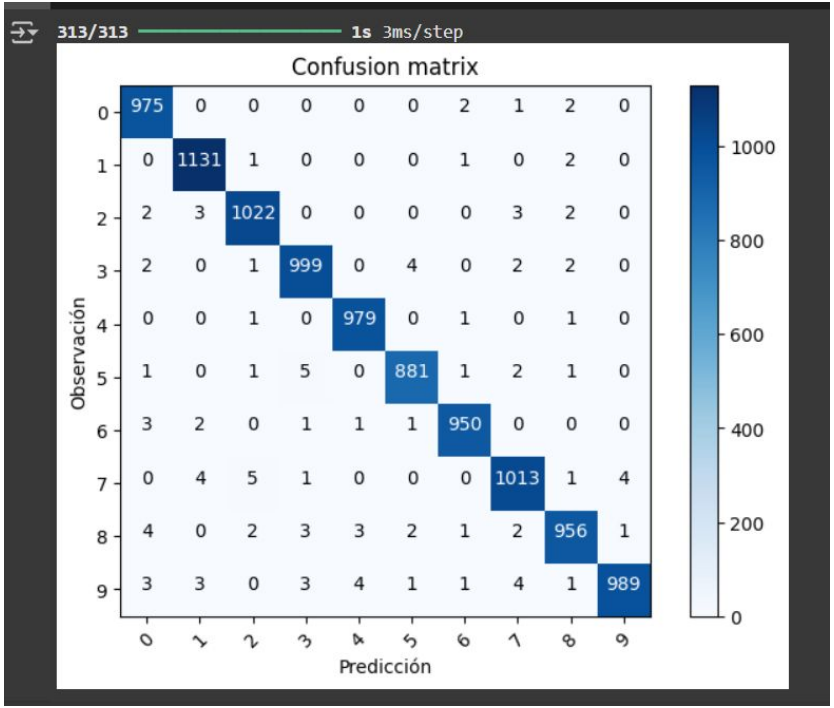
El modelo original tiene una precisión de **97'1%**

El modelo mejorado presenta una precisión de **98’9%**

- **1’8% de diferencia**
- **300% más epocas (15)**
- **Margen de mejora reducido**

Original vs Mejorado

Matriz de confusión del modelo generado



SGD vs Adam

Adam

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9882 - loss: 0.0538
Test accuracy: 0.9907000064849854
```

SGD

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9852 - loss: 0.0455
Test accuracy: 0.9894999861717224
```

SGD vs Adam

Adam

```
Epoch 1/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8524 - loss: 0.4963
Epoch 2/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9817 - loss: 0.0634
Epoch 3/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9878 - loss: 0.0398
Epoch 4/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9913 - loss: 0.0294
Epoch 5/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9923 - loss: 0.0252
```

Original

```
Epoch 1/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5878 - loss: 1.5305
Epoch 2/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9123 - loss: 0.2909
Epoch 3/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9380 - loss: 0.2076
Epoch 4/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9526 - loss: 0.1585
Epoch 5/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9609 - loss: 0.1341
<keras.src.callbacks.history.History at 0x7f8408aa68f0>
```

SGD vs Adam

Adam suele dar mejores resultados y converger más rápido.

Mejorar modelo convolucional

Técnicas:

- Más capas
- Más filtros
- Más neuronas por capa
- Más épocas
- Uso de Adam
- etc.

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 5ms/step - accuracy: 0.9321 - loss: 0.2265 - val_accuracy: 0.9818 - val_loss: 0.0520
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 16s 4ms/step - accuracy: 0.9835 - loss: 0.0553 - val_accuracy: 0.9829 - val_loss: 0.0594
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9875 - loss: 0.0442 - val_accuracy: 0.9881 - val_loss: 0.0414
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9903 - loss: 0.0330 - val_accuracy: 0.9911 - val_loss: 0.0320
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9913 - loss: 0.0306 - val_accuracy: 0.9906 - val_loss: 0.0340
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9922 - loss: 0.0282 - val_accuracy: 0.9860 - val_loss: 0.0597
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9932 - loss: 0.0223 - val_accuracy: 0.9919 - val_loss: 0.0345
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9935 - loss: 0.0221 - val_accuracy: 0.9892 - val_loss: 0.0446
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9923 - val_loss: 0.0397
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9941 - loss: 0.0214 - val_accuracy: 0.9912 - val_loss: 0.0442
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9960 - loss: 0.0142 - val_accuracy: 0.9910 - val_loss: 0.0377
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9955 - loss: 0.0165 - val_accuracy: 0.9917 - val_loss: 0.0396
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9956 - loss: 0.0160 - val_accuracy: 0.9915 - val_loss: 0.0425
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9962 - loss: 0.0120 - val_accuracy: 0.9934 - val_loss: 0.0344
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9974 - loss: 0.0094 - val_accuracy: 0.9914 - val_loss: 0.0398
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9970 - loss: 0.0104 - val_accuracy: 0.9924 - val_loss: 0.0530
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9972 - loss: 0.0089 - val_accuracy: 0.9915 - val_loss: 0.0455
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9969 - loss: 0.0124 - val_accuracy: 0.9926 - val_loss: 0.0357
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9971 - loss: 0.0101 - val_accuracy: 0.9925 - val_loss: 0.0387
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9982 - loss: 0.0066 - val_accuracy: 0.9919 - val_loss: 0.0614
```

Mejorar modelo convolucional

```
[40] test_loss, test_acc = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {test_acc:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9887 - loss: 0.0797
Test Accuracy: 0.9919
```

PT vs TF

```
xy_testPT = torchvision.datasets.MNIST(roo
download=True,

transform=torchvision.transforms.Compose([
xy_test_loaderPT = torch.utils.data.DataLo
correct_count, all_count = 0, 0
for images,labels in xy_test_loaderPT:
 for i in range(len(labels)):
  img = images[i].view(1, 784)
  logps = modelPT(img)
  ps = torch.exp(logps)
  probab = list(ps.detach().numpy()[0])
  pred_label = probab.index(max(probab))
  true_label = labels.numpy()[i]
  if(true_label == pred_label):
    correct_count += 1
  all_count += 1
print("\n PyTorch model Accuracy =", (correct_count/all_count))
```

```
 PyTorch model Accuracy = 0.8907
```

```
[16] _, (x_testTF, y_testTF)= tf.keras.datasets.mnist.load_data()
     x_testTF = x_testTF.reshape(10000, 784).astype('float32')/255
     y_testTF = tf.keras.utils.to_categorical(y_testTF, num_classes=10)
     _ , test_accTF = modelTF.evaluate(x_testTF, y_testTF)
     print('\n TensorFlow model Accuracy =', test_accTF)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8734 - loss: 0.4783

 TensorFlow model Accuracy = 0.8909000158309937
```

PT vs TF

El framework que se elija no es lo más importante.

PT vs TF

- **No importa el framework**
- Método de **entrenamiento**: fit() vs manual
- **Enfoques** distintos

Gracias