# Lab 1: Introduction to Deep Learning Programming

CC-MEI (FIB) Laboratory Assignment

Mario Ventura Burgos

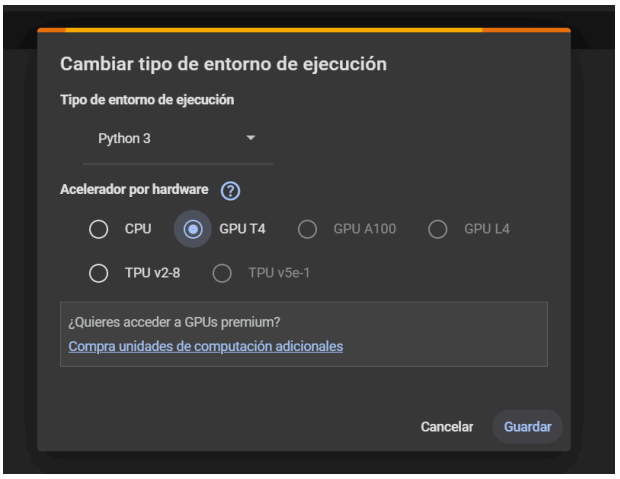

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

# Índice

# TASK 1

**Open a browser of your choice, go to colab.research.google.com, and sign in using your Google account. Click on a new notebook to create a new runtime instance or download a GitHub file. To create a GPU/TPU-enabled runtime, click on the "runtime" label in the toolbar menu, click “Change runtime type”, and then select GPU or TPU.**

Tal y como pide el enunciado, en primer lugar se crea un notebook vacío y se cambia el tipo de entorno de ejecución. Recomendado por la IA incorporada de google colab (Gemini), se selecciona la opción GPU T4.




Posteriormente, se ejecuta el código del enunciado, que es similar al visto en clase. En pocas palabras, este código permite entrenar un modelo a través de varias etapas o **épocas** y se muestra la **matriz de confusión**.

En este caso, se entrenó el modelo en 5 épocas y posteriormente se decidió hacerlo en 8. El hecho de haber entrenado en 5 épocas antes de entrenar en otras 8 (13 en total) provocó que esta segunda fase de 8 épocas iniciase con una precisión (accuracy) considerablemente alta para tratarse de una “primera etapa”. Si se mostrase un gráfico con la precisión obtenida en distintas épocas, se podría observar que, con el paso del tiempo, para una etapa **n**, la diferencia en cuanto a precisión con respecto a la etapa **n-1** (la anterior) cada vez es menor.

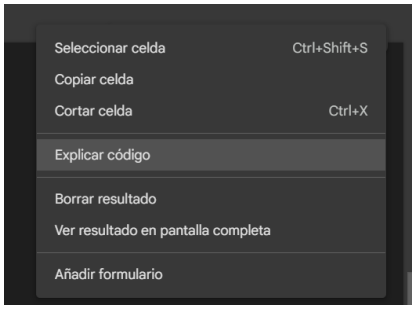
Se puede observar, por tanto, que la precisión aumenta siguiendo una tendencia típicamente logarítmica.

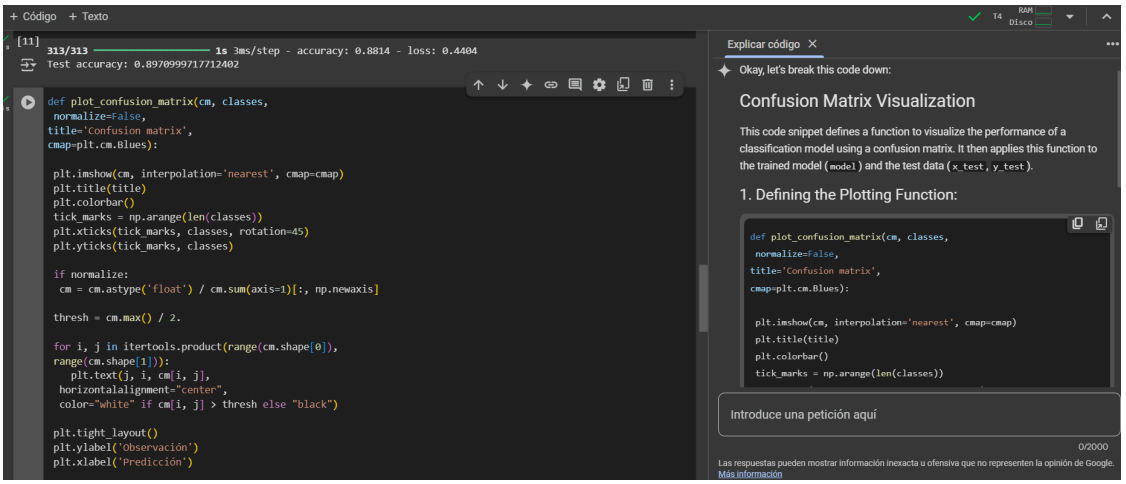
```
[10] model.compile(loss="categorical_crossentropy",
      optimizer="sgd",
      metrics = ['accuracy'])
     model.fit(x_train, y_train, epochs=8)

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>
```

Por último, como último aspecto a destacar para esta primera tarea, cabe mencionar que colab ofrece funciones avanzadas que permiten explicar el código a través de IA. En caso de no entender un bloque de código, resulta una funcionalidad de gran utilidad.

A continuación, se muestra un ejemplo de esta funcionalidad para el bloque de código que genera la matriz de confusión, que resulta ser el bloque de código más grande del presente apartado, y por tanto, el más susceptible a generar confusión.




# TASK 2

**In your Colab, download the GitHub file `https://github.com/jorditorresBCN/Marenostrum5/blob/main/PracticalIntro ductionToDeepLearningBasics.ipynb`. Define, train, and evaluate a new model that improves the Accuracy obtained by the basic model (teacher's model). (a) Explain the improvement applied: More epochs? More layers? More neurons per layer? Print and analyze the confusion matrix. (b) Compare the optimizers SGD and Adam for your neural networks. Describe the results of this task in the lab report.**

(a) Se procede a definir, entrenar y evaluar un nuevo modelo que mejora la precisión obtenida por el modelo básico presentado inicialmente por el profesor.

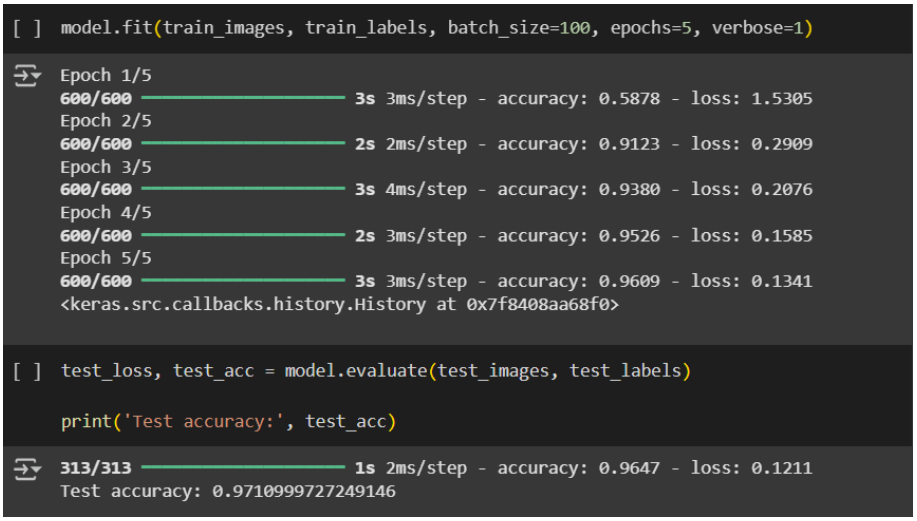
La mejora del modelo inicial se ha llevado a cabo aplicando las técnicas mostradas en la solución del enunciado:

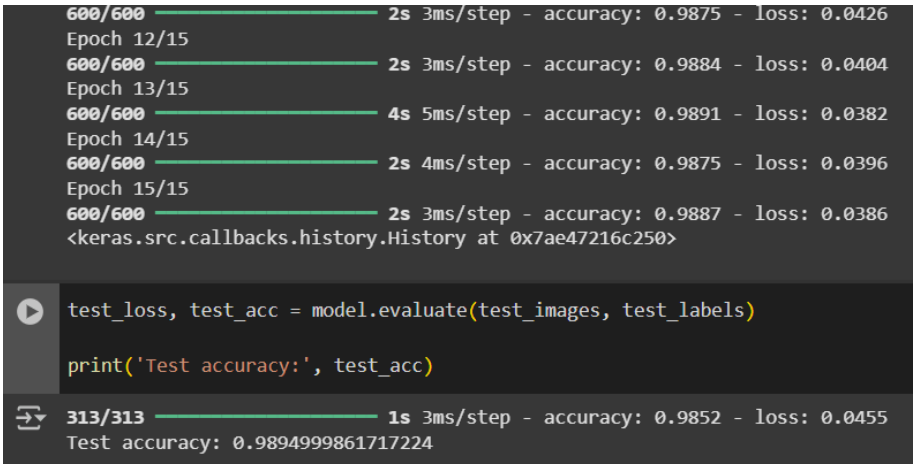
1. **Añadir más capas:** Se añaden capas adicionales con neuronas al modelo original.

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.

Con todo esto, se pueden observar las siguientes diferencias respecto al modelo original:

Original (1) vs Mejorado (2)



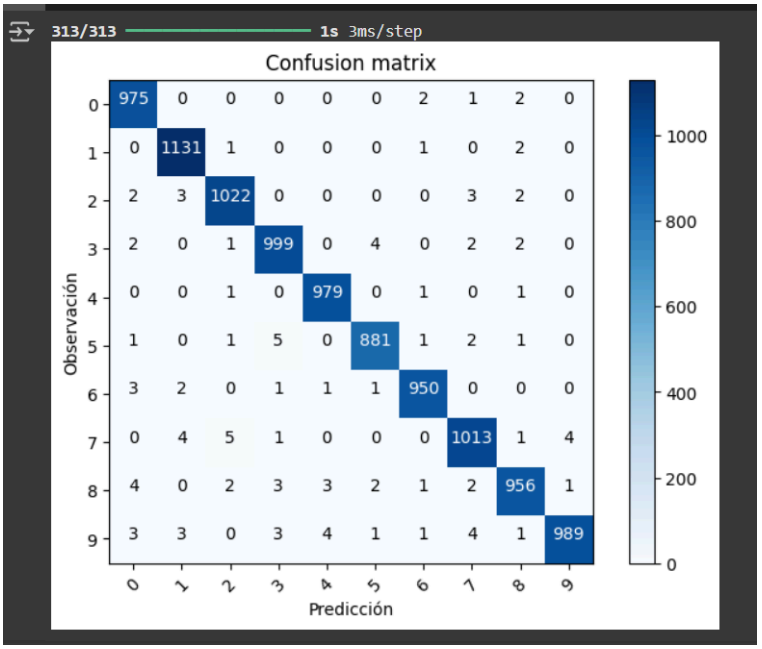




Se observa que el modelo original tiene una precisión de 97'1%, mientras que el modelo mejorado presenta un 98’9% de precisión. Esto no supone una gran mejora teniendo en cuenta que se ha usado el triple (300%) de épocas con respecto al modelo original.

Sin embargo, cabe mencionar que el modelo original ya contaba con un porcentaje de precisión considerablemente alto, haciendo que el margen de mejora fuera menor.

Muy probablemente, con modelos más complejos que contengan una cantidad mayor de datos sí que se aprecie una diferencia considerable al aplicar alguna de las 3 técnicas descritas previamente.

A continuación se muestra la matriz de confusión obtenida para el modelo mejorado.




Se aprecia una disminución de los fallos respecto a la matriz de confusión del modelo original, donde las discrepancias entre observación y predicción eran levemente mayores, y por tanto, los valores obtenidos en la diagonal eran ínfimamente menores. Esto refleja el pequeño aumento de precisión obtenida en el modelo tras las mejoras implementadas.

(b) A continuación, se cambia el optimizer de SGD a Adam, permitiendo obtener las siguientes diferencias:

**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
```

Puede notarse una mayor precisión obtenida en el modelo final entrenado con Adam, aunque la diferencia es muy pequeña. Donde sí que puede apreciarse una diferencia considerable es en la precisión obtenida en cada época del entrenamiento, donde Adam presenta mejores resultados.

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>
```

Tal y como muestran las imágenes anteriores, pese a que la precisión final es ciertamente parecida, en Adam se aprecia una mayor precisión en las primeras épocas. Por ejemplo, en la primera época el modelo original (SGD) obtiene una precisión del 58’78%, mientras que el nuevo modelo que hace uso de Adam muestra un 85’24% de precisión.

Al llegar a la quinta etapa (el modelo mejorado realiza 15 pero se ha optado por mostrar únicamente las primeras 5 para que la comparación sea más realista), sin embargo, la precisión obtenida con Adam es del 99’23%, frente al 96’09% del modelo original. Pese a que sigue habiendo una diferencia, esta es menor.

Puede concluirse, por tanto, que normalmente **Adam** suele dar mejores resultados y converger más rápido.

# TASK 3

**Define, train, and evaluate a new model that improves the Accuracy obtained by the basic convolutional model (teacher's model). Explain the improvement applied. Describe the results of doing this task in the lab report.**

Para mejorar el modelo original presentado en clase, se han realizado los siguientes cambios.

1. **Más capas convolucionales (3 en lugar de 2)** → Permite aprender patrones más complejos.
2. **Más filtros (32 → 64 → 128)** → Captura mejor las características de los dígitos.

3. **Batch Normalization** → Acelera el entrenamiento y mejora la estabilidad.
4. **Dropout (0.4 en capa densa)** → Evita el sobreajuste.
5. **Más neuronas en la capa densa (256 en lugar de 128)** → Mejora la capacidad de representación.
6. **Entrenar por más épocas (20 en lugar de 10)** → Permite una mejor convergencia.
7. **Usar Adam en lugar de SGD** → Converge más rápido y mejora la precisión.

Algunos de estos cambios se han aplicado tras las conclusiones obtenidas en apartados previos (SGD vs Adam, uso de más épocas, etc.), mientras que otros cambios se han aplicado al ser sugeridos por Gemini AI.

Los resultados obtenidos son los siguientes:

```
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
```

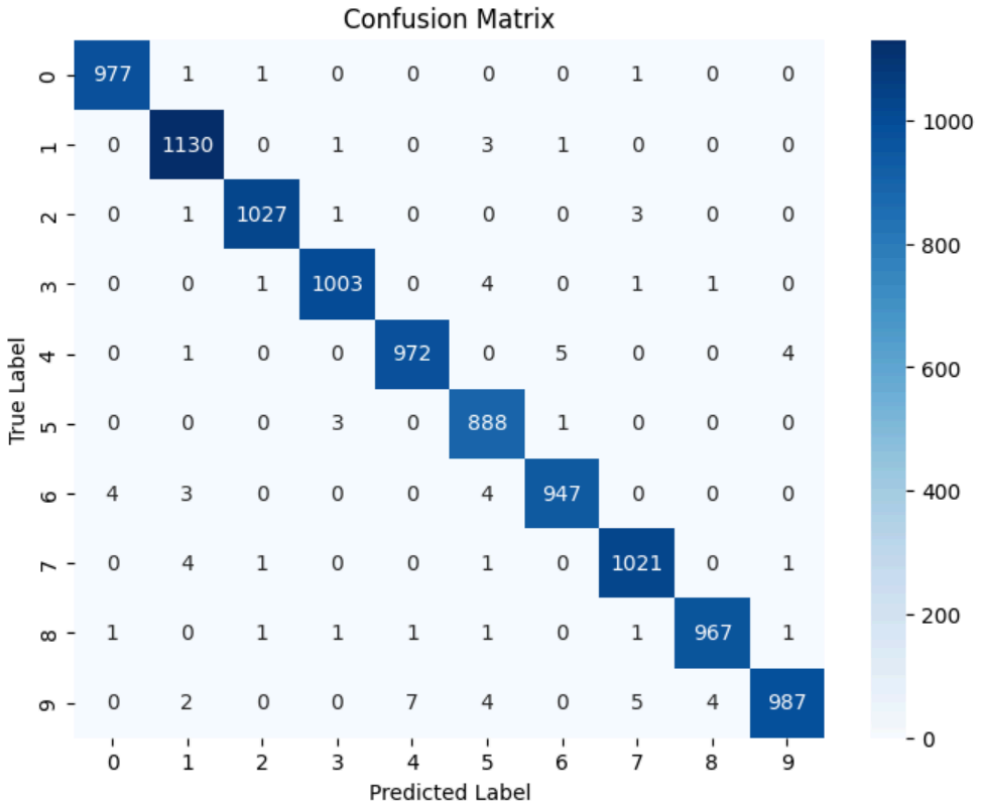
```
[40] test_loss, test_acc = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {test_acc:.4f}")

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

La precisión obtenida finalmente es superior al 99%, suponiendo una pérdida menor al 1% (0’8% aproximadamente). Además, al usar Adam y al añadir más filtros, más capas y más neuronas por capa, se obtiene una precisión alta desde las primeras épocas.

Puede apreciarse que desde la primera etapa se obtiene una precisión del 93’21%, que es una cifra considerablemente alta para tratarse de una primera aproximación.

A continuación, se muestra también la matriz de confusión de este modelo.




# TASK 4

**Create a new notebook in Colab and reproduce the above code to compare PyTorch and TensorFlow. Save your notebook as a PDF and add it to the lab report delivery with the name PTvsTF.pdf.**

Tras implementar el código y probar PyTorch y TensorFlow, se obtienen los resultados que se muestran a continuación.

```
[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
```

```
xy_testPT = torchvision.datasets.MNIST(root='./data', train=False,
download=True,

transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor()]))
xy_test_loaderPT = torch.utils.data.DataLoader(xy_testPT)
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
```

Las precisiones obtenidas para TensorFlow y PyTorch son 89'09% y 89'07% respectivamente. Siendo estrictos a nivel matemático, se podría decir que **TensorFlow es el que obtiene una mayor precisión**, pero la realidad es que parece ser que esto en realidad no importa. Ambas son buenas opciones y su uso depende en gran medida de las preferencias personales de cada desarrollador.

Se puede concluir, por tanto, que el framework que se elija no es lo más importante.

# TASK 5

**Briefly describe the main differences between TensorFlow and PyTorch that you consider relevant.**

En primer lugar, tal y como se ha concluido en el apartado anterior, el framework que se elija no es lo más relevante ya que ambos tienen recursos computacionales parecidos. Los conceptos de uno pueden trasladarse al otro, es decir, que el aprendizaje sobre Deep Learning es transferible.

Por otro lado, puede apreciarse una diferencia notable en cuanto a la forma de entrenamiento. En TensorFlow existe una función *fit()* que realiza el entrenamiento de forma automatizada con una simple llamada, mientras que en PyTorch el entrenamiento debe realizarse con bucles implementados manualmente (en este caso, bucles 'for'). Esto sí que supone una diferencia importante ya que el hecho de automatizar el entrenamiento como lo hace TensorFlow resulta ventajoso y más cómodo para cualquier programador.

Por último, el enfoque de cada framework también parece ser distinto. Mientras que PyTorch se suele usar en contextos de investigación, TensorFlow se suele usar más en contextos empresariales o en producción. Esto se debe a que PyTorch es muy intuitivo y ofrece gran flexibilidad, permitiendo hacer pruebas rápidas, mientras que TensorFlow ofrece herramientas avanzadas como TensorFlow Lite o TensorFlow Serving, que resultan ideales en contextos de producción a nivel empresarial.