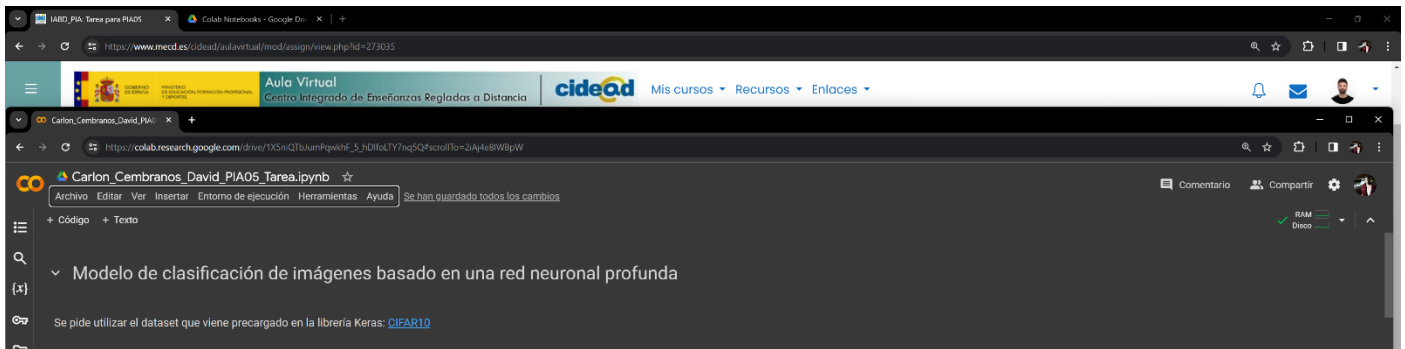
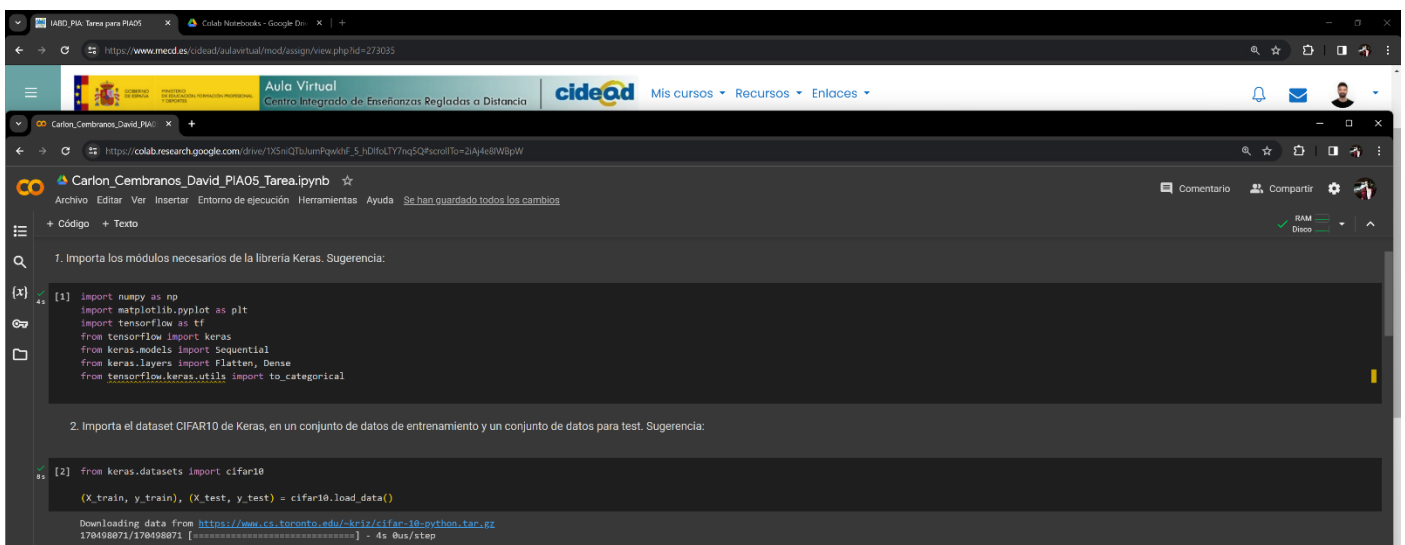


TAREA DE PROGRAMACION DE INTELIGENCIA ARTIFICIAL 05

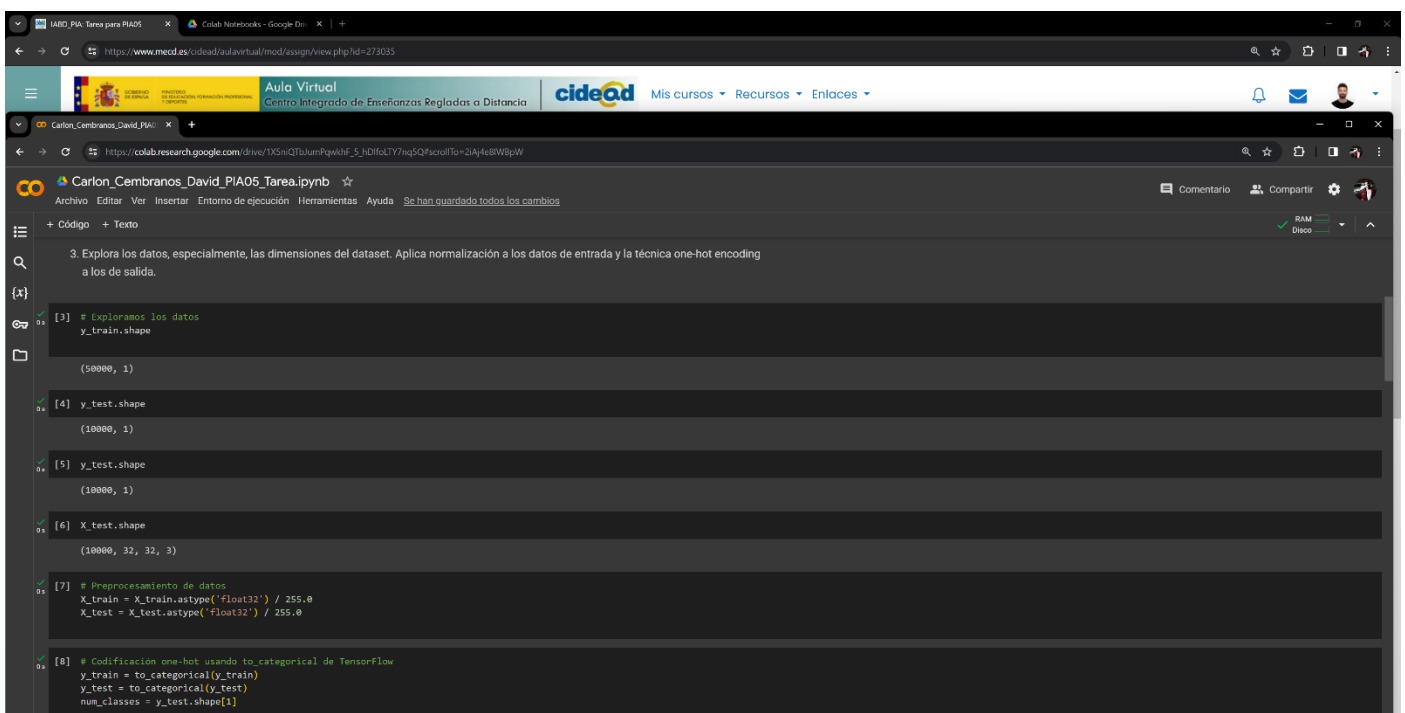
Apartado 1: Se crea un notebook en Colab, con su título y las celdas de código indicadas.



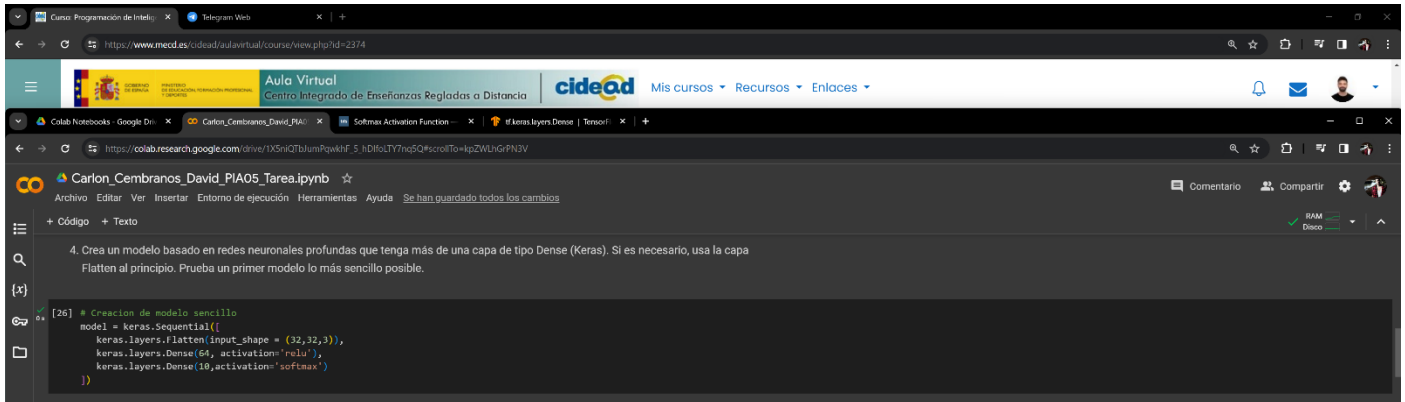
Apartado 2: Se importa el dataset y se quedan configurados los conjuntos de datos de entrenamiento y de test.



Apartado 3: Se muestran las dimensiones del dataset, se aplica la normalización de los datos de entrada y one-hot encoding a los de salida.



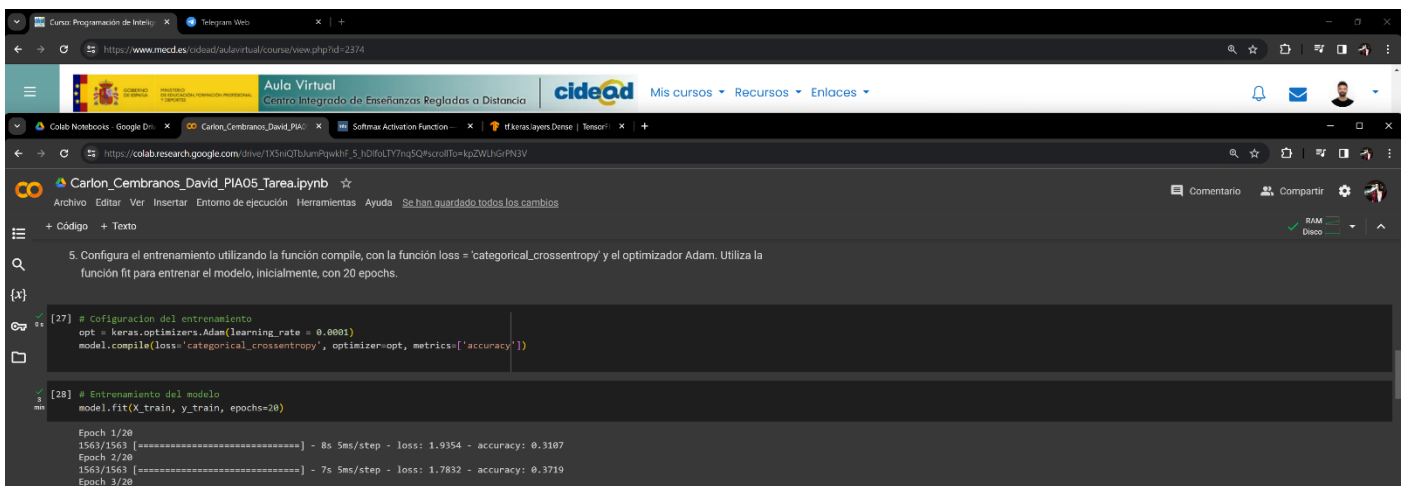
Apartado 4: Se crea el modelo de red neuronal profunda.



The screenshot shows a Google Colab notebook titled 'Carlón_Cembranos_David_PIA05_Tarea.ipynb'. The notebook is open to a cell containing Python code for creating a simple neural network model using Keras. The code defines a model with three layers: a Flatten layer, a Dense layer with 64 units and 'relu' activation, and another Dense layer with 10 units and 'softmax' activation. The notebook interface includes a top bar with the CIDEAD logo and navigation links, and a left sidebar with file and code tabs. The output of the cell is not visible.

```
[26] # Creación de modelo sencillo
model = keras.Sequential([
    keras.layers.Flatten(input_shape = (32,32,3)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

Apartado 5: Se configura y se lanza el entrenamiento con la función compile, fit y los parámetros indicados.

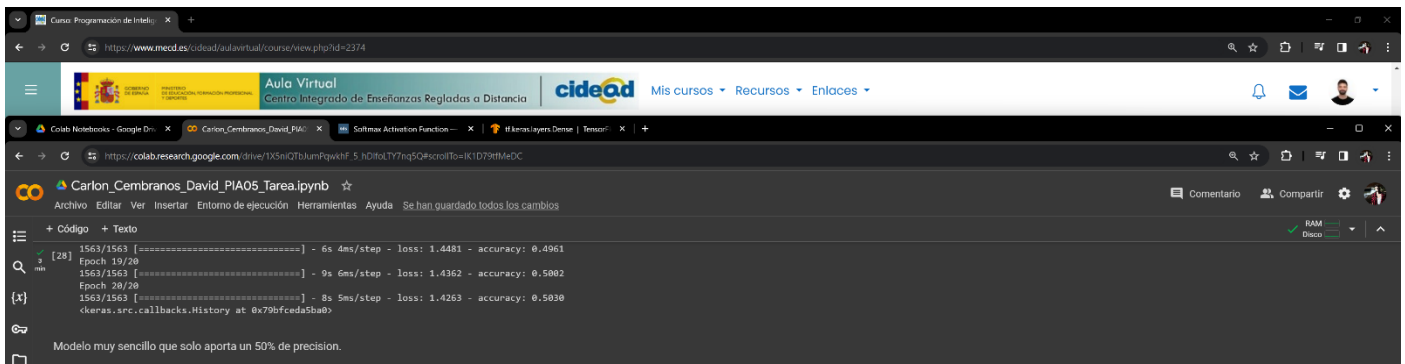


The screenshot shows the same Google Colab notebook, now with two additional cells. The first cell configures the training process by compiling the model with 'categorical_crossentropy' loss, 'Adam' optimizer, and 'accuracy' metric. The second cell trains the model for 20 epochs. The output of the training process is displayed, showing the progress of the model over 20 epochs, including loss and accuracy values.

```
[27] # Configuración del entrenamiento
opt = keras.optimizers.Adam(learning_rate = 0.0001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

[28] # Entrenamiento del modelo
model.fit(X_train, y_train, epochs=20)

Epoch 1/20
1563/1563 [=====] - 8s 5ms/step - loss: 1.9354 - accuracy: 0.3107
Epoch 2/20
1563/1563 [=====] - 7s 5ms/step - loss: 1.7832 - accuracy: 0.3719
Epoch 3/20
```



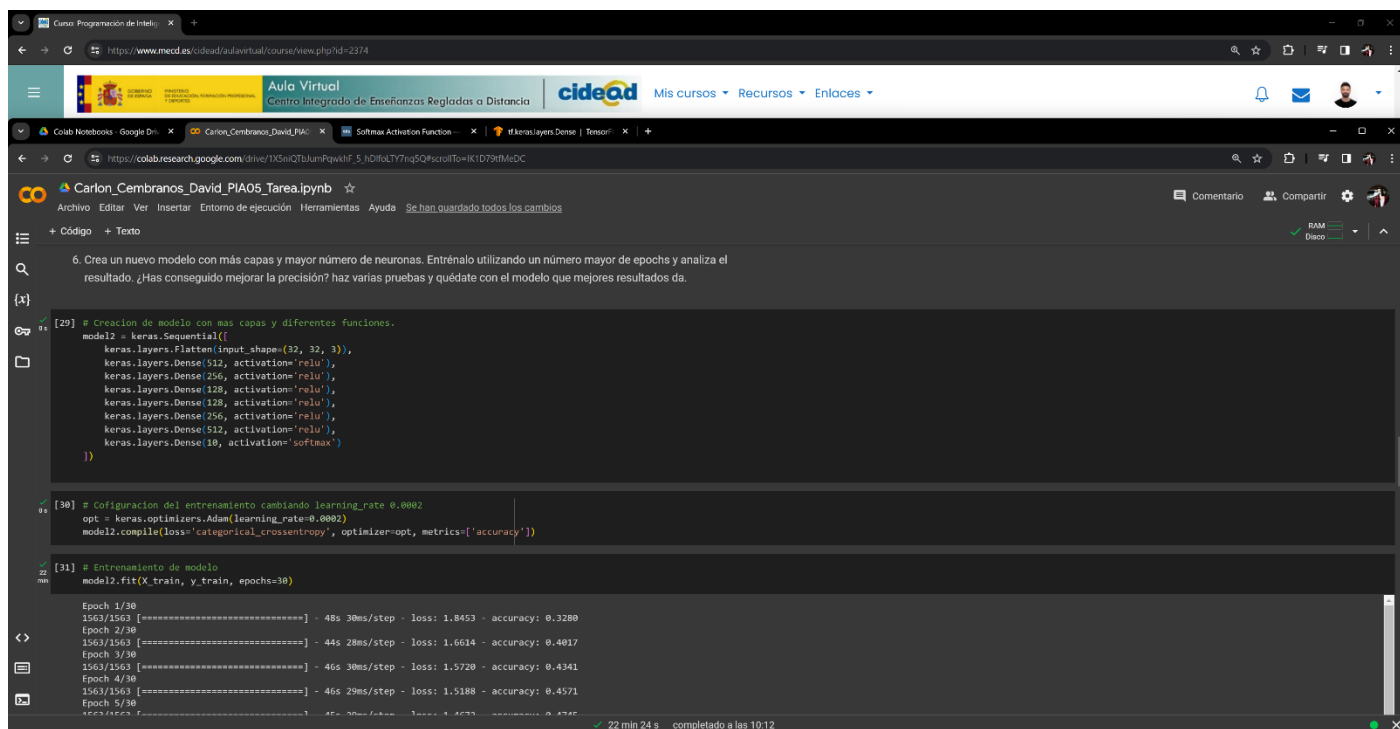
The screenshot shows the same Google Colab notebook, now with a third cell. This cell displays the final output of the training process, showing the progress of the model over 20 epochs, including loss and accuracy values. The output indicates that the model achieved an accuracy of 0.5038, which is approximately 50%.

```
[28] 1563/1563 [=====] - 6s 4ms/step - loss: 1.4481 - accuracy: 0.4961
Epoch 19/20
1563/1563 [=====] - 9s 6ms/step - loss: 1.4362 - accuracy: 0.5002
Epoch 20/20
1563/1563 [=====] - 8s 5ms/step - loss: 1.4263 - accuracy: 0.5038
<keras.src.callbacks.History at 0x79bfcda5ba0>

Modelo muy sencillo que solo aporta un 50% de precisión.
```

Modelo muy sencillo que solo aporta un 50% de precisión.

Apartado 6: Se crea un segundo modelo con más capas y neuronas.



The screenshot shows a Google Colab notebook titled 'Carlón_Cembranos_David_PIA05_Tarea.ipynb'. The notebook contains three code cells. The first cell creates a Keras model with the following architecture: a Flatten layer (32, 32, 3), followed by three Dense layers (512, 128, 128) with 'relu' activation, then another Dense layer (256) with 'relu' activation, followed by a Dense layer (512) with 'relu' activation, and finally a Dense layer (10) with 'softmax' activation. The second cell configures the training with Adam optimizer and a learning rate of 0.0002. The third cell trains the model for 30 epochs. The output shows the training progress for the first 5 epochs, with accuracy increasing from 0.3288 to 0.4571.

```
[29] # Creacion de modelo con mas capas y diferentes funciones.
model2 = keras.Sequential([
    keras.layers.Flatten(input_shape=(32, 32, 3)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

[30] # Configuración del entrenamiento cambiando learning_rate 0.0002
opt = keras.optimizers.Adam(learning_rate=0.0002)
model2.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

[31] # Entrenamiento de modelo
model2.fit(X_train, y_train, epochs=30)

Epoch 1/30
1563/1563 [=====] - 48s 30ms/step - loss: 1.8453 - accuracy: 0.3288
Epoch 2/30
1563/1563 [=====] - 44s 28ms/step - loss: 1.6614 - accuracy: 0.4017
Epoch 3/30
1563/1563 [=====] - 46s 30ms/step - loss: 1.5720 - accuracy: 0.4341
Epoch 4/30
1563/1563 [=====] - 46s 29ms/step - loss: 1.5188 - accuracy: 0.4571
Epoch 5/30
1563/1563 [=====] - 46s 29ms/step - loss: 1.4776 - accuracy: 0.4744
```



The screenshot shows the continuation of the Keras model training from the previous notebook. The output shows the training progress for epochs 26 to 30, with accuracy increasing from 0.6984 to 0.7368. The final output shows the training history for the last epoch.

```
[31] Epoch 26/30
1563/1563 [=====] - 44s 28ms/step - loss: 0.8320 - accuracy: 0.6984
Epoch 27/30
1563/1563 [=====] - 43s 28ms/step - loss: 0.8065 - accuracy: 0.7080
Epoch 28/30
1563/1563 [=====] - 44s 28ms/step - loss: 0.7810 - accuracy: 0.7177
Epoch 29/30
1563/1563 [=====] - 44s 28ms/step - loss: 0.7534 - accuracy: 0.7245
Epoch 30/30
1563/1563 [=====] - 43s 28ms/step - loss: 0.7276 - accuracy: 0.7368
<keras.src.callbacks.history at 0x79bfcddad10>
```

Después de probar con varias configuraciones. Cambiando las funciones, he llegado a la conclusión que ReLu es la que mejor se adapta en las capas intermedias. Softmax en la última capa, ya que es una salida múltiple. También modifiqué el learnig_rate, añadí neuronas y capas he conseguido una precisión de 73%.

Después de probar con varias configuraciones. Cambiando las funciones, he llegado a la conclusión que ReLu es la que mejor se adapta en las capas intermedias. Softmax en la última capa, ya que es una salida múltiple. También modifiqué el learnig_rate, añadí neuronas y capas he conseguido una precisión de 73%.

Apartado 7: Se evalúa el modelo, obteniendo la precisión con los datos de test.



```
7. Utiliza el método evaluate para ver la precisión que se alcanzaría con datos nuevos, aplicándolo al conjunto de datos de test. ¿Es muy diferente a la precisión alcanzada en el entrenamiento?
```

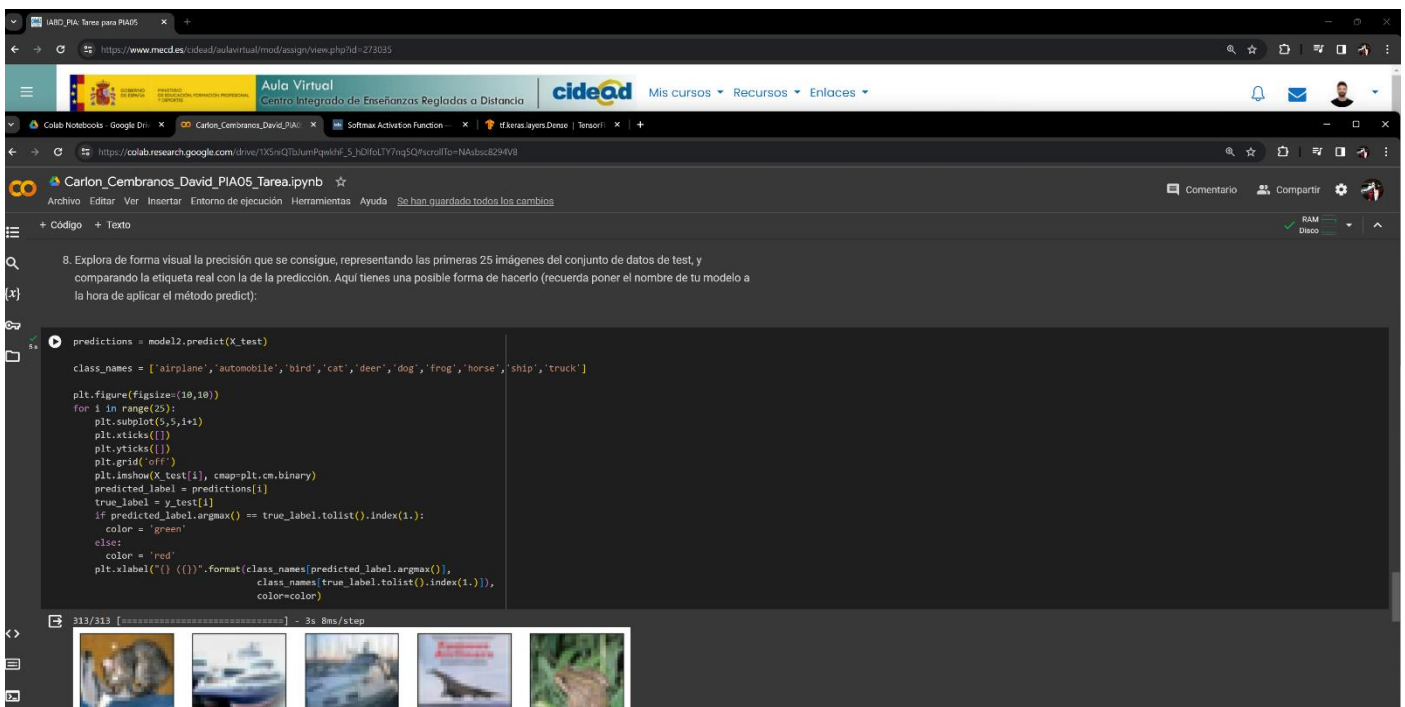
```
[32]: result2 = model.evaluate(X_test, y_test)
      print(result2)
```

```
313/313 [=====] - 2s 6ms/step - loss: 1.6267 - accuracy: 0.5192
[1.6267498386289, 0.5192800269889832]
```

Si, es muy diferente solo consigue el 52%.

Si, es muy diferente solo consigue el 52%.

Apartado 8: Se representan imágenes del conjunto de datos de test, con su etiqueta real y la proporcionada por el modelo.



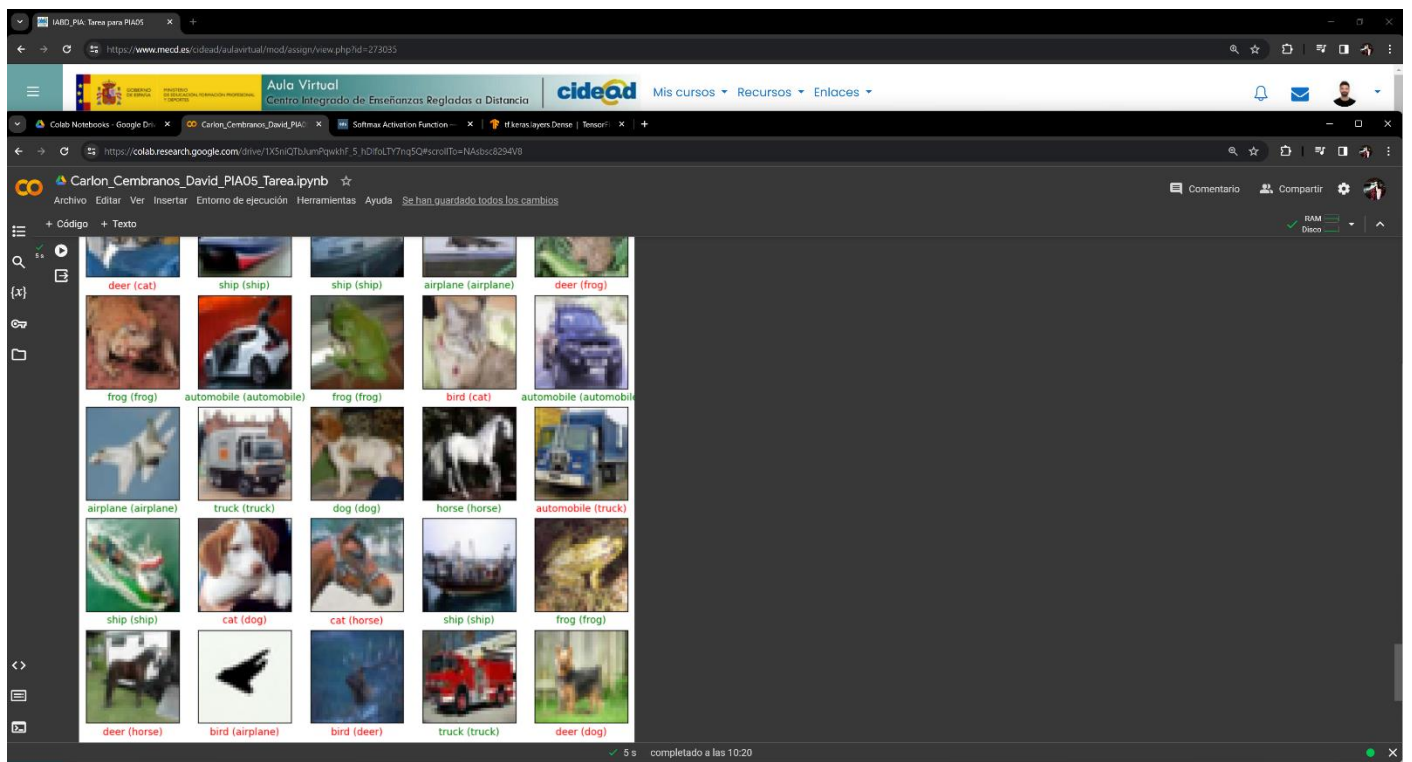
```
8. Explora de forma visual la precisión que se consigue, representando las primeras 25 imágenes del conjunto de datos de test, y comparando la etiqueta real con la de la predicción. Aquí tienes una posible forma de hacerlo (recuerda poner el nombre de tu modelo a la hora de aplicar el método predict):
```

```
predictions = model.predict(X_test)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_test[i], cmap=plt.cm.binary)
    predicted_label = predictions[i]
    true_label = y_test[i]
    if predicted_label.argmax() == true_label.tolist().index(1.):
        color = 'green'
    else:
        color = 'red'
    plt.xlabel("{} ({})".format(class_names[predicted_label.argmax()],
                                class_names[true_label.tolist().index(1.)]),
                color=color)
```

313/313 [=====] - 3s 8ms/step



Encalce al cuaderno de Google colab:

https://colab.research.google.com/drive/1X5niQTbJumPqwkHf_5_hDIfoLTY7ng5Q?usp=sharing