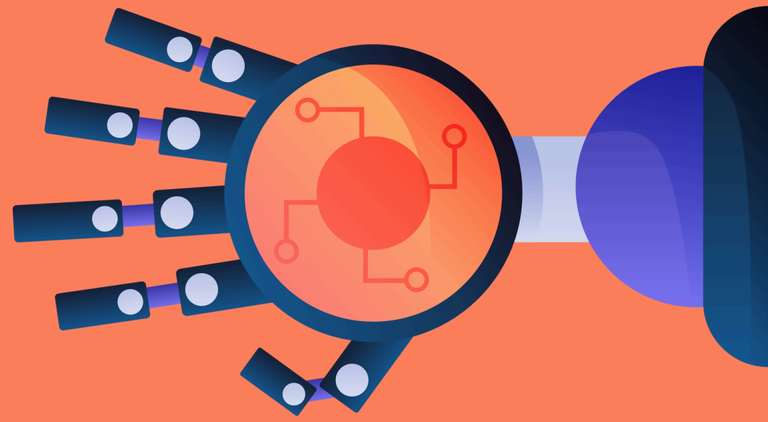
PROYECTO 3ER PACIAL

REDES NEURONALES CONVOLUCIONALES

HÉCTOR CAMACHO
ZAMORA 594557

IMPORTACION DE IMAGENES



```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image


# Preprocesamiento de las imágenes
datagen = ImageDataGenerator(rescale=1./255,validation_split = 0.2)

train_generator = datagen.flow_from_directory(
  '/content/drive/MyDrive/numeros_IA/Numbers/Train',   # Directorio de entrenamiento
  target_size=(280, 280),
  batch_size=32,
  class_mode='categorical',
  color_mode='rgb',  # 20% de los datos serán usados para validación
  subset='training'       # La parte de entrenamiento
)
val_generator = datagen.flow_from_directory(
  '/content/drive/MyDrive/numeros_IA/Numbers/Train',
  target_size=(280, 280),
  batch_size=32,
  class_mode='categorical',
  subset='validation',
  color_mode='rgb'
)
test_generator = datagen.flow_from_directory(
  '/content/drive/MyDrive/numeros_IA/Numbers/Test',
  target_size=(280, 280),
  batch_size=32,
  class_mode='categorical',
  color_mode='rgb'
)
```

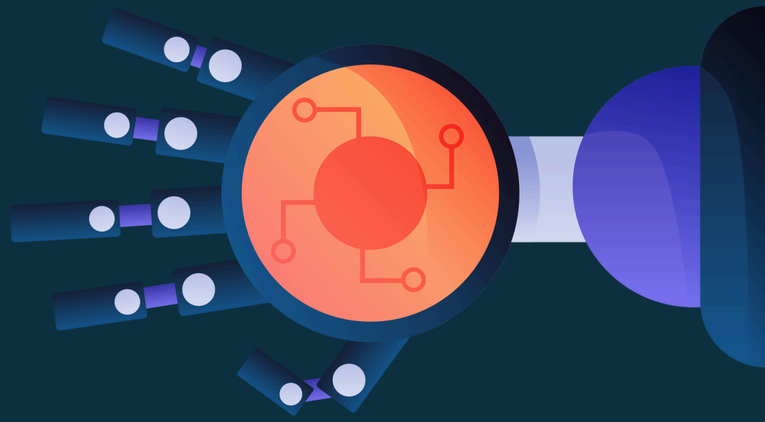
Utilice flow from directory para importar las imágenes directamente desde google collab, las importe en tamaño 280x280 y en RGB. Hice un subset para los datos de validacion. Solo en el modelo 4 y 5 cambie color_mode a ‘grayscale’.

MODELO 1

```
from tensorflow.keras import layers, models

model = models.Sequential()


model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(280, 280, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))


model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 278, 278, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 139, 139, 32)	0
conv2d_4 (Conv2D)	(None, 137, 137, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 68, 68, 64)	0
conv2d_5 (Conv2D)	(None, 66, 66, 64)	36,928
flatten_1 (Flatten)	(None, 278784)	0
dense_2 (Dense)	(None, 64)	17,842,240
dense_3 (Dense)	(None, 10)	650

Total params: 17,899,210 (68.28 MB)
Trainable params: 17,899,210 (68.28 MB)
Non-trainable params: 0 (0.00 B)



Para el primer modelo, utilice las imagenes en RGB y aplique un reshape a 280x280. Utilice un compilador Adam con el learning rate por default. Le pregunte a chatGPT cual era la mejor arquitectura para predecir numeros. Se le agrego un early stopping según el val loss.

MODELO 1:

RESULTADOS

```
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=[early_stopping])

Epoch 1/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58s 425ms/step - accuracy: 0.9123 - loss: 0.3104 - val_accuracy: 0.7371 - val_loss: 1.8564
Epoch 2/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54s 398ms/step - accuracy: 0.9593 - loss: 0.1379 - val_accuracy: 0.7306 - val_loss: 2.1819
Epoch 3/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54s 401ms/step - accuracy: 0.9822 - loss: 0.0699 - val_accuracy: 0.7540 - val_loss: 2.3390
Epoch 4/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55s 405ms/step - accuracy: 0.9920 - loss: 0.0459 - val_accuracy: 0.7297 - val_loss: 2.7222
Epoch 5/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55s 407ms/step - accuracy: 0.9938 - loss: 0.0325 - val_accuracy: 0.7353 - val_loss: 2.6533
Epoch 6/20 135/135 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81s 400ms/step - accuracy: 0.9839 - loss: 0.0580 - val_accuracy: 0.7502 - val_loss: 2.6948
```

MEJOR EPOCA

La mejor epoca fue la primera, ya que presento el val_loss mas pequeño.

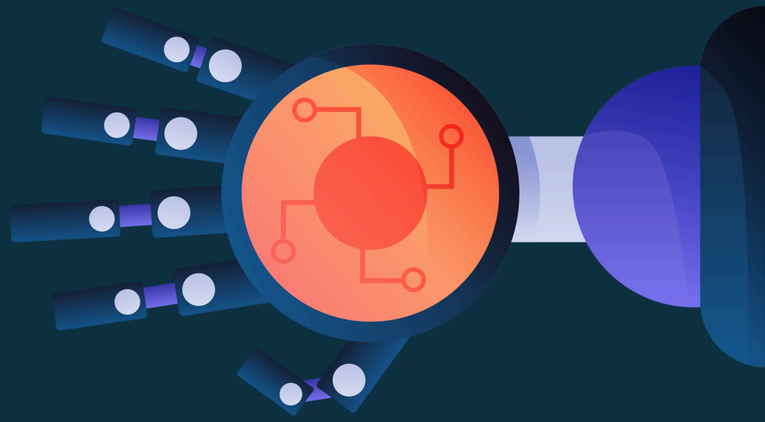
VAL ACCURACY

Presento un val accuracy de 0.7371

VAL LOSS

El val loss tuvo un valor de 1.8564

MODELO 2



```
from tensorflow.keras import layers, models

model = models.Sequential()


model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(280, 280, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))



model.compile(optimizer='adam',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
model.summary()


early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 278, 278, 32)	896
max_pooling2d (MaxPooling2D)	(None, 139, 139, 32)	0
conv2d_1 (Conv2D)	(None, 137, 137, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 68, 68, 64)	0
flatten (Flatten)	(None, 295936)	0
dense (Dense)	(None, 64)	18,939,968
dense_1 (Dense)	(None, 10)	650

Total params: 18,960,010 (72.33 MB)
Trainable params: 18,960,010 (72.33 MB)
Non-trainable params: 0 (0.00 B)

Para este segundo modelo, elimine una capa de convolución para observar sus resultados, además cambie el parámetro de paro, para que fuera el val_accuracy.

MODELO 2:

RESULTADOS

```
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=[early_stopping])
Epoch 1/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1568s 11s/step - accuracy: 0.2672 - loss: 5.2446 - val_accuracy: 0.5890 - val_loss: 1.5369
Epoch 2/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50s 356ms/step - accuracy: 0.8152 - loss: 0.6904 - val_accuracy: 0.7005 - val_loss: 1.4086
Epoch 3/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52s 372ms/step - accuracy: 0.9420 - loss: 0.2487 - val_accuracy: 0.6978 - val_loss: 1.7540
Epoch 4/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 353ms/step - accuracy: 0.9707 - loss: 0.1428 - val_accuracy: 0.7140 - val_loss: 2.0977
Epoch 5/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 350ms/step - accuracy: 0.9850 - loss: 0.0783 - val_accuracy: 0.6942 - val_loss: 2.6344
Epoch 6/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50s 355ms/step - accuracy: 0.9749 - loss: 0.1347 - val_accuracy: 0.7077 - val_loss: 2.4722
Epoch 7/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51s 367ms/step - accuracy: 0.9853 - loss: 0.0734 - val_accuracy: 0.7302 - val_loss: 2.8228
Epoch 8/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 348ms/step - accuracy: 0.9899 - loss: 0.0511 - val_accuracy: 0.7356 - val_loss: 2.9328
Epoch 9/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83s 355ms/step - accuracy: 0.9917 - loss: 0.0377 - val_accuracy: 0.7239 - val_loss: 3.2548
Epoch 10/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50s 360ms/step - accuracy: 0.9944 - loss: 0.0377 - val_accuracy: 0.7167 - val_loss: 3.4346
Epoch 11/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50s 354ms/step - accuracy: 0.9912 - loss: 0.0546 - val_accuracy: 0.7023 - val_loss: 3.2833
Epoch 12/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83s 360ms/step - accuracy: 0.9892 - loss: 0.0581 - val_accuracy: 0.7149 - val_loss: 3.6650
Epoch 13/20 140/140 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50s 359ms/step - accuracy: 0.9844 - loss: 0.0749 - val_accuracy: 0.7149 - val_loss: 3.6996
```

MEJOR EPOCA

La mejor época fue la ocho.

VAL ACCURACY

Presento un val accuracy de 0.7356 que es un poco mejor que el anterior.

VAL LOSS

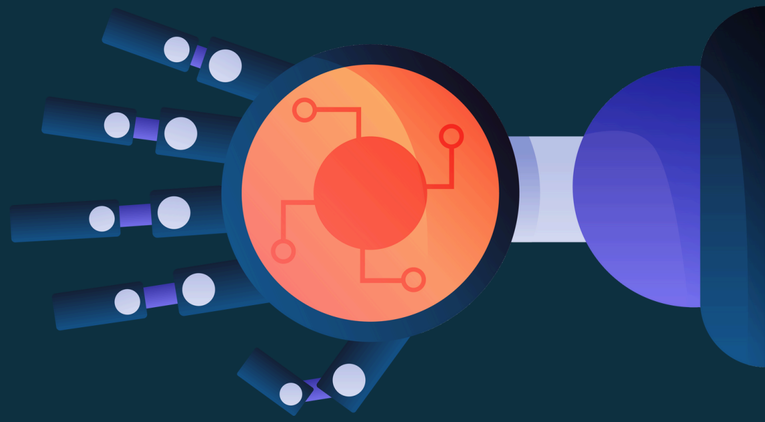
El val loss tuvo un valor de 2.9328. que es una unidad mas grande que el anterior.

MODELO 3

```
from tensorflow.keras import layers, models

model = models.Sequential()


model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(140, 140, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

opt = tf.keras.optimizers.Adam(learning_rate=0.002)
model.compile(optimizer=opt,
       loss='categorical_crossentropy',
       metrics=['accuracy'])
model.summary()


early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 138, 138, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 69, 69, 32)	0
conv2d_7 (Conv2D)	(None, 67, 67, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 33, 33, 64)	0
conv2d_8 (Conv2D)	(None, 31, 31, 64)	36,928
flatten_2 (Flatten)	(None, 61504)	0
dense_4 (Dense)	(None, 64)	3,936,320
dense_5 (Dense)	(None, 10)	650

Total params: 3,993,290 (15.23 MB)
Trainable params: 3,993,290 (15.23 MB)
Non-trainable params: 0 (0.00 B)



Para este tercer modelo, utilice la misma arquitectura del primero, sin embargo cambie las imágenes de entrada a tamaño 140x140, elegí este numero sin ninguna razón, simplemente dividí el 280 a la mitad. Además aumente el learning rate por 0.002

MODELO 3:

RESULTADOS

```
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=[early_stopping])
Epoch 1/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 55s 385ms/step - accuracy: 0.3138 - loss: 2.3825 - val_accuracy: 0.6380 - val_loss: 1.3866
Epoch 2/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 49s 366ms/step - accuracy: 0.7372 - loss: 0.8490 - val_accuracy: 0.7081 - val_loss: 1.3176
Epoch 3/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 48s 356ms/step - accuracy: 0.8553 - loss: 0.4870 - val_accuracy: 0.7353 - val_loss: 1.4437
Epoch 4/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 50s 370ms/step - accuracy: 0.9199 - loss: 0.2777 - val_accuracy: 0.7409 - val_loss: 1.5787
Epoch 5/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 50s 369ms/step - accuracy: 0.9448 - loss: 0.1897 - val_accuracy: 0.7465 - val_loss: 1.8928
Epoch 6/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 49s 365ms/step - accuracy: 0.9483 - loss: 0.1783 - val_accuracy: 0.7493 - val_loss: 2.2898
Epoch 7/20 135/135 ━━━━━━━━━━━━━━━━━━━━ 47s 351ms/step - accuracy: 0.9765 - loss: 0.0898 - val_accuracy: 0.7671 - val_loss: 2.2721
```

MEJOR EPOCA

La mejor época fue la numero 2, ya que presento el menor val_loss.

VAL ACCURACY

Presento un val accuracy de 0.7081

VAL LOSS

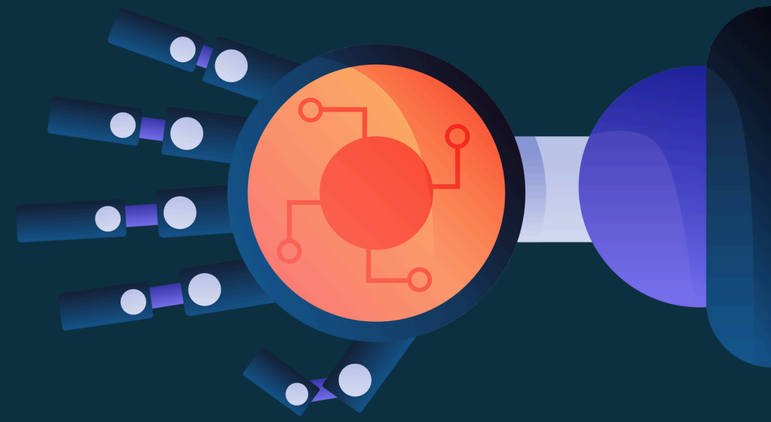
El val loss tuvo un valor de 1.3176

MODELO 4

```
from tensorflow.keras import layers, models

model = models.Sequential()


model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(120, 120, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))


model.compile(optimizer='adam',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
model.summary()

early_stopping          =          EarlyStopping(monitor='val_accuracy',          patience=5,
restore_best_weights=True)
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=
[early_stopping])
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 118, 118, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 59, 59, 32)	0
conv2d_3 (Conv2D)	(None, 57, 57, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 64)	0
flatten_1 (Flatten)	(None, 50176)	0
dense_2 (Dense)	(None, 64)	3,211,328
dense_3 (Dense)	(None, 10)	650

Total params: 3,230,794 (12.32 MB)
Trainable params: 3,230,794 (12.32 MB)
Non-trainable params: 0 (0.00 B)



Para este cuarto modelo, utilice la misma arquitectura del primero, sin embargo cambie las imágenes de entrada a tamaño 120x120, y las imágenes de entrada las cambie a escala de grises. Este modelo presento los resultados mucho mas rápido, que los anteriores, esto se debe gracias a que había menos capas que procesar.

MODELO 4:

RESULTADOS

```
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=[early_stopping])

Epoch 1/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 53s 359ms/step - accuracy: 0.2738 - loss: 2.2173 - val_accuracy: 0.6331 - val_loss: 1.3777
Epoch 2/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 339ms/step - accuracy: 0.7213 - loss: 0.9493 - val_accuracy: 0.7455 - val_loss: 1.1190
Epoch 3/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 48s 342ms/step - accuracy: 0.8779 - loss: 0.4570 - val_accuracy: 0.7707 - val_loss: 1.0305
Epoch 4/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 334ms/step - accuracy: 0.9198 - loss: 0.2874 - val_accuracy: 0.7716 - val_loss: 1.3261
Epoch 5/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 337ms/step - accuracy: 0.9703 - loss: 0.1345 - val_accuracy: 0.7851 - val_loss: 1.4148
Epoch 6/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 49s 353ms/step - accuracy: 0.9745 - loss: 0.0954 - val_accuracy: 0.7644 - val_loss: 1.6090
Epoch 7/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 338ms/step - accuracy: 0.9868 - loss: 0.0623 - val_accuracy: 0.7806 - val_loss: 1.7345
Epoch 8/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 49s 347ms/step - accuracy: 0.9918 - loss: 0.0414 - val_accuracy: 0.7887 - val_loss: 1.7975
Epoch 9/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 337ms/step - accuracy: 0.9966 - loss: 0.0204 - val_accuracy: 0.7833 - val_loss: 2.0663
Epoch 10/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 336ms/step - accuracy: 0.9990 - loss: 0.0131 - val_accuracy: 0.7707 - val_loss: 1.8656
Epoch 11/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 49s 352ms/step - accuracy: 0.9980 - loss: 0.0131 - val_accuracy: 0.7788 - val_loss: 2.2167
Epoch 12/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 45s 323ms/step - accuracy: 0.9979 - loss: 0.0181 - val_accuracy: 0.7743 - val_loss: 2.5815
Epoch 13/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 331ms/step - accuracy: 0.9994 - loss: 0.0065 - val_accuracy: 0.7779 - val_loss: 2.5526
```

MEJOR EPOCA

La mejor época fue la numero 8.

VAL ACCURACY

Presento un val accuracy de 0.7887

VAL LOSS

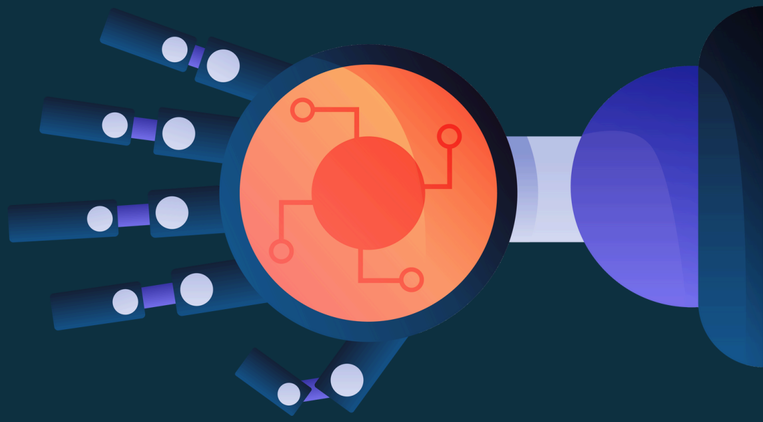
El val loss tuvo un valor de 1.7975.

MODELO 5

```
from tensorflow.keras import layers, models

model = models.Sequential()


model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(120, 120, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))



model.compile(optimizer='adam',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
model.summary()
early_stopping          =          EarlyStopping(monitor='val_accuracy',          patience=5,
restore_best_weights=True)

history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=
[early_stopping])
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 118, 118, 32)	320
max_pooling2d_6 (MaxPooling2D)	(None, 59, 59, 32)	0
flatten_3 (Flatten)	(None, 111392)	0
dense_6 (Dense)	(None, 64)	7,129,152
dense_7 (Dense)	(None, 10)	650

Total params: 7,130,122 (27.20 MB)
Trainable params: 7,130,122 (27.20 MB)
Non-trainable params: 0 (0.00 B)



Para este quinto modelo, utilice las imagenes en escala de grises y en tamaño 120x120. Elimine una etapa de convolucion y una de maxpooling.

MODELO 5:

RESULTADOS

(MODELO CON PEORES RESULTADOS)

```
history = model.fit(train_generator, epochs=20,validation_data= val_generator ,callbacks=[early_stopping])
Epoch 1/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 50s 340ms/step - accuracy: 0.1395 - loss: 3.7974 - val_accuracy: 0.2968 - val_loss: 2.0594
Epoch 2/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 48s 346ms/step - accuracy: 0.3726 - loss: 1.8829 - val_accuracy: 0.4038 - val_loss: 1.7835
Epoch 3/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 325ms/step - accuracy: 0.5118 - loss: 1.5066 - val_accuracy: 0.4604 - val_loss: 1.6605
Epoch 4/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 327ms/step - accuracy: 0.6065 - loss: 1.2449 - val_accuracy: 0.5126 - val_loss: 1.5752
Epoch 5/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 334ms/step - accuracy: 0.6963 - loss: 0.9630 - val_accuracy: 0.5576 - val_loss: 1.5341
Epoch 6/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 327ms/step - accuracy: 0.7764 - loss: 0.7444 - val_accuracy: 0.5908 - val_loss: 1.5211
Epoch 7/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 330ms/step - accuracy: 0.8290 - loss: 0.5940 - val_accuracy: 0.6052 - val_loss: 1.5615
Epoch 8/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 329ms/step - accuracy: 0.8533 - loss: 0.4992 - val_accuracy: 0.6259 - val_loss: 1.5722
Epoch 9/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 332ms/step - accuracy: 0.8974 - loss: 0.3933 - val_accuracy: 0.6295 - val_loss: 1.6326
Epoch 10/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 46s 327ms/step - accuracy: 0.9242 - loss: 0.3060 - val_accuracy: 0.6511 - val_loss: 1.7323
Epoch 11/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 338ms/step - accuracy: 0.9404 - loss: 0.2631 - val_accuracy: 0.6313 - val_loss: 1.8565
Epoch 12/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 45s 324ms/step - accuracy: 0.9616 - loss: 0.1907 - val_accuracy: 0.6556 - val_loss: 1.9460
Epoch 13/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 332ms/step - accuracy: 0.9659 - loss: 0.1716 - val_accuracy: 0.6358 - val_loss: 1.9216
Epoch 14/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 45s 320ms/step - accuracy: 0.9797 - loss: 0.1268 - val_accuracy: 0.6484 - val_loss: 2.0670
Epoch 15/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 47s 333ms/step - accuracy: 0.9843 - loss: 0.1005 - val_accuracy: 0.6520 - val_loss: 2.1035
Epoch 16/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 49s 347ms/step - accuracy: 0.9919 - loss: 0.0700 - val_accuracy: 0.6475 - val_loss: 2.2099
Epoch 17/20 140/140 ━━━━━━━━━━━━━━━━━━━━ 81s 339ms/step - accuracy: 0.9916 - loss: 0.0642 - val_accuracy: 0.6538 - val_loss: 2.3197
```

MEJOR EPOCA

La mejor época fue la numero 12.

VAL ACCURACY

Presento un val accuracy de 0.6556

VAL LOSS

El val loss tuvo un valor de 1.9460

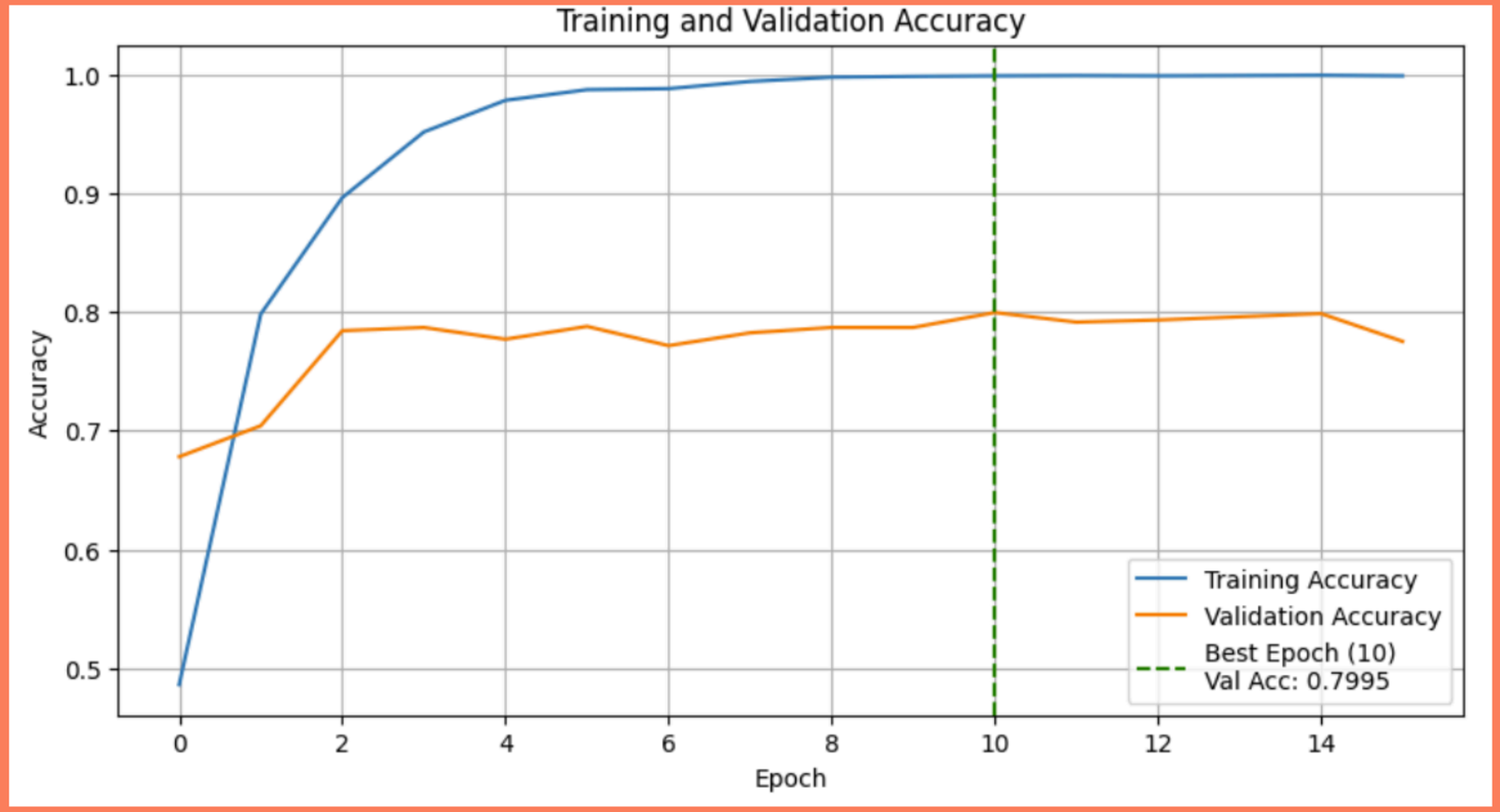
MEJOR MODELO:

MODELO 4

GRAFICAS

Por fallas tecnicas tuve que volver a correr el modelo, los resultados dieron distintos pero el modelo fue entrenado a partir de los resultados establecidos en la slide pasada.

El val loss es el espacio que existe entre la linea azul y naranja. Despues de la epoca diez el validation accuracy comenzo a bajar y despues paro gracias al early stopping. El training accuracy llego muy cerca a 1 queriendo decir que es muy probable que exista un overfitting.

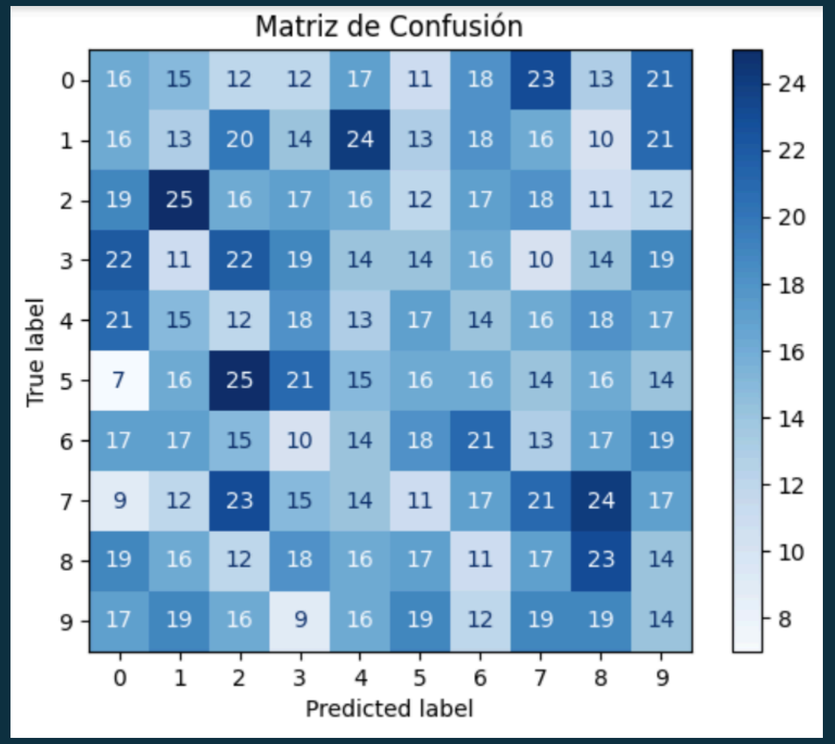


ENTRENAMIENTO DEL MODELO 4

CON TODOS LOS DATOS DE TRAIN Y VALIDANDO EN TEST

METRICAS

	precision	recall	f1-score	support
0	0.10	0.10	0.10	158
1	0.08	0.08	0.08	165
2	0.09	0.10	0.10	163
3	0.12	0.12	0.12	161
4	0.08	0.08	0.08	161
5	0.11	0.10	0.10	160
6	0.13	0.13	0.13	161
7	0.13	0.13	0.13	163
8	0.14	0.14	0.14	163
9	0.08	0.09	0.09	160
accuracy			0.11	1615
macro avg	0.11	0.11	0.11	1615
weighted avg	0.11	0.11	0.11	1615



Se puede observar que me dieron resultados muy malos en las pruebas con las imagenes de prueba. Creo que se tuvo un sobre ajuste a los datos de entrenamiento. En la matriz de confusion se puede observar que hubo muchas predicciones con errores. A continuación se verificara como funciona en un modelo a tiempo real.

CODIGO TIEMPO REAL

En las pruebas a tiempo real, todos los números fueron correctamente predichos. Lo cual me sorprendió bastante debido a los resultados tan malos que dio en las métricas.

```
import cv2
import numpy as np
import tensorflow as tf

# Cargar el modelo entrenado
model = tf.keras.models.load_model("model_CNN2.h5")

# Inicializar la cámara
cap = cv2.VideoCapture(0)

# Clases (ajusta según tu modelo si no son dígitos)
class_names = [str(i) for i in range(10)]

while True:
  ret, frame = cap.read()
  if not ret:
    break

  # Imagen original para visualización
  display_frame = frame.copy()

  # Preprocesamiento
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  resized = cv2.resize(gray, (120, 120)) # según cómo entrenaste el modelo
  input_img = resized / 255.0 # Normalizar como en entrenamiento
  input_img = input_img.reshape(1, 120, 120, 1) # añadir batch y canal

  # Clasificación
  predictions = model.predict(input_img, verbose=0)
  predicted_class = np.argmax(predictions)
  probabilities = predictions[0]

  # Mostrar la imagen que entra al modelo (en ventana aparte)
  cv2.imshow("Input to CNN (120x120)", resized)

  # Escribir probabilidades sobre la imagen original
  y0 = 30
  dy = 25
  for i, prob in enumerate(probabilities):
    color = (0, 255, 0) if i == predicted_class else (0, 0, 255) # verde si es la clase predicha
    text = f"{class_names[i]}: {prob:.2f}"
    cv2.putText(display_frame, text, (10, y0 + i*dy), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

  # Mostrar la imagen original con texto
  cv2.imshow("Webcam Classification", display_frame)

  # Salir con 'q'
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Liberar recursos
cap.release()
cv2.destroyAllWindows()
```