mport tensorflow as tf

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.layers import Conv2D, Dropout, MaxPooling2D, Flatten, Dense

from tensorflow.keras.utils import to\_categorical

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

train\_images = train\_images.astype('float32') / 255

test\_images = test\_images.astype('float32') / 255

train\_images = train\_images.reshape(train\_images.shape[0], 28, 28, 1)

test\_images = test\_images.reshape(test\_images.shape[0], 28, 28, 1)

train\_labels = tf.keras.utils.to\_categorical(train\_labels, 10)

test\_labels = tf.keras.utils.to\_categorical(test\_labels, 10)

**PRIMER MODELO CON DROPOUT, FLATTEN**

model = tf.keras.Sequential()

# Must define the input shape in the first layer of the neural network

model.add(Conv2D(filters=64, kernel\_size=2, padding='same',

          activation='relu', input\_shape=(28,28,1)))

model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.3))

model.add(Conv2D(filters=32, kernel\_size=2, padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

model.summary()

**COMPILAR MODELO**

model.compile(loss='categorical\_crossentropy',

             optimizer='rmsprop',

             metrics=['accuracy'])

**ENTRENAR MODELO**

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

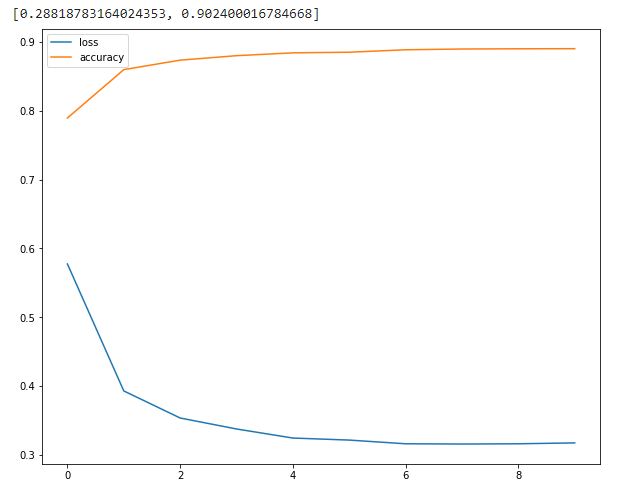
**RESULTADOS**

score = model.evaluate(test\_images, test\_labels, verbose=0)

print(score)

import pandas as pd

pd.DataFrame(history.history).plot(figsize=(10,8));



NUEVO MODELO, CON REGULADORES, ELIMINAMOS FLATTEN

from tensorflow.keras import regularizers

model\_optimizer = tf.keras.models.Sequential([

   tf.keras.layers.Flatten(input\_shape= (28, 28, 1)),

   tf.keras.layers.Dense(256,

kernel\_regularizer = regularizers.l2(1e-5), activation = "relu"),

    tf.keras.layers.Dropout(0.2),

     tf.keras.layers.Dense(128,

kernel\_regularizer= regularizers.l2(1e-5),activation = "relu"),

     tf.keras.layers.Dropout(0.2),

     tf.keras.layers.Dense(10, activation = "softmax")

])

model\_optimizer.summary()

**COMPILA**

model.compile(loss='categorical\_crossentropy',

             optimizer='rmsprop',

             metrics=['accuracy'])

**ENTRENA**

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

**RESULTADOS**

score = model.evaluate(test\_images, test\_labels, verbose=0)

print(score)

import pandas as pd

pd.DataFrame(history.history).plot(figsize=(10,8));

**SE VA EMPEORANDO, CAMBIAMOS OPTIMIZADOR AL COMPILAR**

from tensorflow import keras

model.compile(loss='categorical\_crossentropy',

             optimizer = keras.optimizers.Adam(learning\_rate=0.001),

             metrics=['accuracy'])

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

score = model.evaluate(test\_images, test\_labels, verbose=0)

print(score)

import pandas as pd

pd.DataFrame(history.history).plot(figsize=(10,8));

**MEJORO!!!! DE 89 A 90% CON L.R. DE 0.01 NOS DA MUY MAL, CON 0.001 AL 90%**

from tensorflow.keras import regularizers

model = tf.keras.models.Sequential([

   tf.keras.layers.Conv2D(75, (3,3), activation = "relu", input\_shape= (28, 28, 1)),

   tf.keras.layers.MaxPool2D((2,2)),

   tf.keras.layers.Flatten(),

   tf.keras.layers.Dense(256, kernel\_regularizer = regularizers.l2(1e-5), activation = "relu"),

   tf.keras.layers.Dropout(0.2),

   tf.keras.layers.Dense(128, kernel\_regularizer= regularizers.l2(1e-5),activation = "relu"),

   tf.keras.layers.Dropout(0.2),

   tf.keras.layers.Dense(10, activation = "softmax")

])

model.summary()

model.compile(loss='categorical\_crossentropy',

             optimizer = "adam",

             metrics=['accuracy'])

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

score = model.evaluate(test\_images, test\_labels, verbose=0)

print(score)

import pandas as pd

pd.DataFrame(history.history).plot(figsize=(10,8));

**MEJORO 91%**

AGREGAMOS OPTIMIZADOR CON LEARNING RATE EN LA COMPILACION

from tensorflow import keras

model.compile(loss='categorical\_crossentropy',

             optimizer = keras.optimizers.Adam(learning\_rate=0.001),

             metrics=['accuracy'])

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

MEJORO 92%

NUEVO MODELO

from tensorflow.keras import regularizers

model = tf.keras.models.Sequential([

   tf.keras.layers.Conv2D(75, (3,3), activation = "relu", input\_shape= (28, 28, 1)),

   tf.keras.layers.MaxPool2D((2,2)),

   tf.keras.layers.Flatten(),

   tf.keras.layers.Dense(256, kernel\_regularizer = regularizers.l2(1e-5), activation = "relu"),

   tf.keras.layers.Dropout(0.2),

   tf.keras.layers.Dense(128, kernel\_regularizer= regularizers.l2(1e-5),activation = "relu"),

   tf.keras.layers.Dropout(0.2),

   tf.keras.layers.Dense(10, activation = "softmax")

])

from tensorflow import keras

model.compile(loss='categorical\_crossentropy',

             optimizer = keras.optimizers.Adam(learning\_rate=0.001),

             metrics=['accuracy'])

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=20)

score = model.evaluate(test\_images, test\_labels, verbose=0)

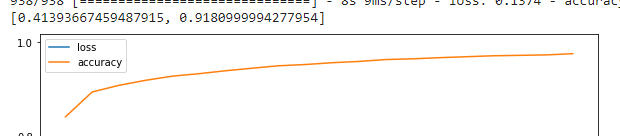
print(score)

import pandas as pd

pd.DataFrame(history.history).plot(figsize=(10,8));

Se cambio a epochs=20 porque grafico mostraba que mejoraba. Llego casi a 92% de exactitud.

Pero ya no mejora tanto al aumentar los epochs, aunque con menos se reduce exactitud.



MODIFICA EL COMPILADO

from tensorflow import keras

model.compile(loss='categorical\_crossentropy',

             optimizer = keras.optimizers.Adam(learning\_rate=0.001),

             metrics=['accuracy'])

from tensorflow.keras.callbacks import Callback

from tensorflow.keras.callbacks import Callback

class TrainingCallback(Callback):

  def on\_epoch\_end(self, epoch, logs = {}):

    if logs.get("accuracy") > 0.90:

      print("Lo logramos, modelo llego a 95%, detenemos entrenamiento "+str(epoch))

      self.model.stop\_training = True

callback = TrainingCallback()

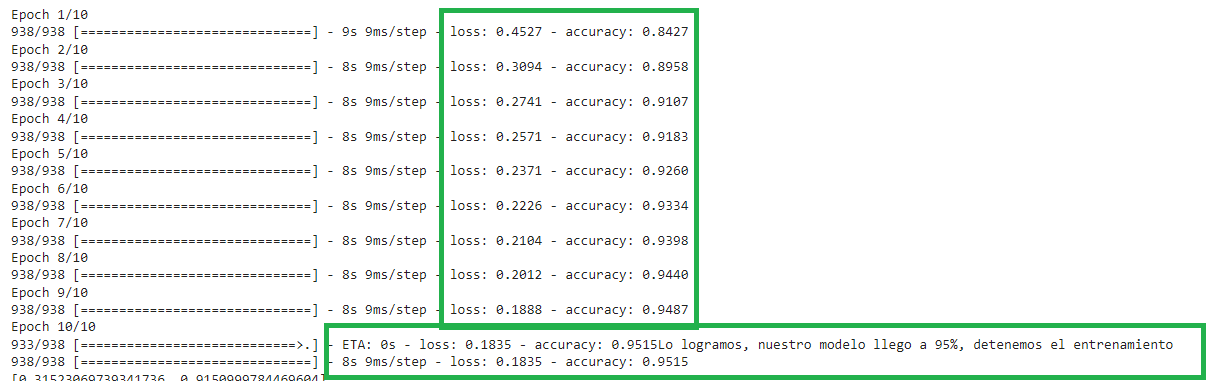
history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

          callbacks = [callback],

         epochs=40)



Logro el 95% con epochs =