import 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)

**MODELO WITH FLATTEN, DROPOUT**

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()

=======================================================

**COMPILE**

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

             optimizer='rmsprop',

             metrics=['accuracy'])

**FIT**

history=model.fit(train\_images,

         train\_labels,

         batch\_size=64,

         epochs=10)

Epoch 1/10

938/938 [==============================] - 10s 10ms/step - loss: 0.3181 - accuracy: 0.8900

Epoch 2/10

938/938 [==============================] - 9s 10ms/step - loss: 0.3196 - accuracy: 0.8891

Epoch 3/10

938/938 [==============================] - 9s 10ms/step - loss: 0.3254 - accuracy: 0.8876

Epoch 4/10

938/938 [==============================] - 9s 10ms/step - loss: 0.3221 - accuracy: 0.8889

Epoch 5/10

938/938 [==============================] - 9s 9ms/step - loss: 0.3214 - accuracy: 0.8909

Epoch 6/10

938/938 [==============================] - 8s 9ms/step - loss: 0.3243 - accuracy: 0.8885

Epoch 7/10

938/938 [==============================] - 8s 9ms/step - loss: 0.3300 - accuracy: 0.8882

Epoch 8/10

938/938 [==============================] - 9s 9ms/step - loss: 0.3292 - accuracy: 0.8874

Epoch 9/10

938/938 [==============================] - 9s 9ms/step - loss: 0.3306 - accuracy: 0.8877

Epoch 10/10

938/938 [==============================] - 9s 9ms/step - loss: 0.3328 - accuracy: 0.8875

**SCORE**

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

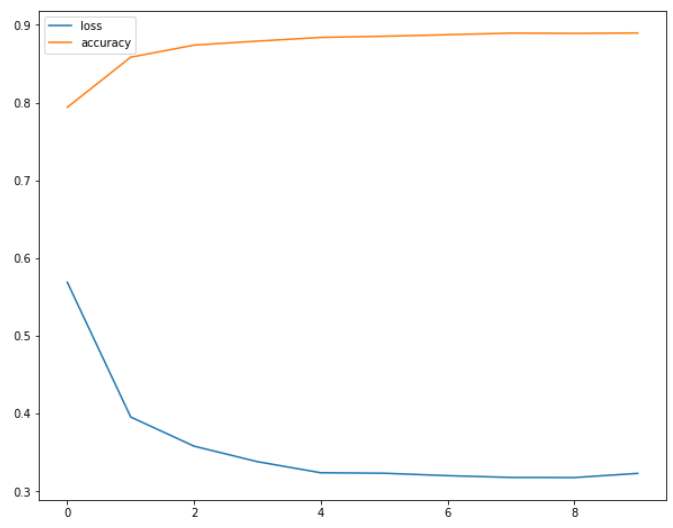
print(score)

LOSS ACCURACY

[0.2901, 0.902]

import pandas as pd

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



===================================

**MODELO WITH REGULATORS**

<https://keras.io/api/layers/regularizers/>

Regularizers allow to apply penalties on layer parameters or layer activity during optimization.

These penalties are summed into the loss function that the network optimizes.

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),

model.add(Flatten()),

      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()

**COMPILE**

**FIT**

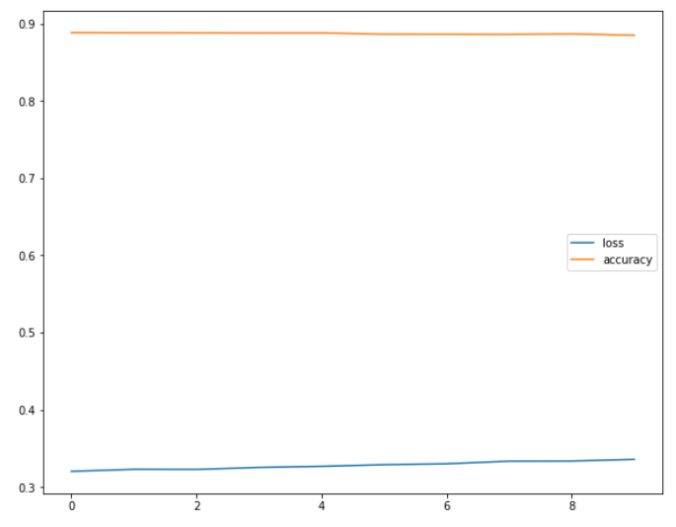
**SCORE**

**GRAPH**

**[0.27475, 0.90530002]**

Loss menor, 0.27 reduciendose

Accuracy aumento 0.905



**MODELO CON LEARNING RATE**

**COMPILE PREVIO**

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

             optimizer='rmsprop',

             metrics=['accuracy'])

**COMPILE NUEVO**

**LR** = hp.Choice('learning\_rate', values = [1e-2, 1e-3, 1e-4])

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

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

metrics=['accuracy'])

from tensorflow import keras

opt = keras.optimizers.Adam(learning\_rate=0.01)

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

             optimizer = opt,

             metrics=['accuracy'])

==================================

**CALLBACKS DESPUES DE COMPILAR**

**COMPILE**

**FIT**

early = tf.keras.callbacks.EarlyStopping(monitor='accuracy',patience=1)

model.fit(train\_images,

         train\_labels,

         batch\_size=64,

        callbacks=[early],

         epochs=10)

from tensorflow.keras.callbacks import ModelCheckpoint

model.compile(optimizer = "adam",

loss = "categorical\_crossentropy",

metrics = ["accuracy"])

# Creamos unos checkpoint de nuestros modelos con TensorFlow, verbose: mostrar mas información

checkpoint\_path = "model\_checpoints/checkpoint"

checkpoint\_weighs= ModelCheckpoint(

    filepath = checkpoint\_path,

    frecuency = "epoch",

    save\_weights\_only = True,

    verbose = 1

)

# Entrenamos el modelo, y almacenamos en cada steps los pesos del moedelo.

history\_weight = model\_weight.fit(

    train\_generator,

    epochs = 20,

    callbacks = [checkpoint\_weighs],

    validation\_data = validation\_generator

)

model\_weight.save\_weights("model\_manul/my\_model")

fit

checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath='mi\_primer\_red\_conv.hdf5',

                                verbose=1,

                                monitor='accuracy',

                                save\_best\_only=True)

**SCORE**

[0.3124, 0.903]

**TRAINING CALLBACK, SI LLEGO A 95%**

from tensorflow.keras.callbacks import Callback

class **TrainingCallback**(Callback):

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

    if logs.get("accuracy") > 0.95:

      print("Lo logramos, modelo llego a 95%, detenemos el entrenamiento")

      self.model.stop\_training = True

callback = **TrainingCallback**()

history\_callback = model\_callback.fit(

    train\_generator,

    epochs = 20,

    callbacks = [callback],

    validation\_data = validation\_generator

def get\_model():

  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(len(classes), activation = "softmax")

  ])

  return model

model\_early = get\_model()

model\_early.summary()

model\_early.compile(optimizer= "adam",

loss = "categorical\_crossentropy", metrics = ["accuracy"])

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_2 (Conv2D) (None, 26, 26, 75) 750

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 13, 13, 75) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_4 (Flatten) (None, 12675) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_12 (Dense) (None, 256) 3245056

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_6 (Dropout) (None, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_13 (Dense) (None, 128) 32896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_7 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_14 (Dense) (None, 24) 3096

=================================================================

Total params: 3,281,798

Trainable params: 3,281,798

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CodeText

COMPILE

FIT

SCORE

EARLY STOPPING

callback\_early = tf.keras.callbacks.EarlyStopping(monitor = "loss",

patience=3, mode = "auto")

history\_early = model\_early.fit(

    train\_generator,

    epochs= 20,

    callbacks = [callback\_early],

    validation\_data = validation\_generator

)