Chess Data Set



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Enunciado 3**](#_2c35568x6wg8)

[**2. Código y sus apartados 3**](#_oitqhxk3ijhw)

[**2.1 Librerías 3**](#_t8cwbo6n4466)

[**2.1.2 Código 3**](#_10d5mursqpn2)

[**2.2 Import Dataset 3**](#_a8ubj7b3amwi)

[**2.2.2 Comando 3**](#_45rpww5lwche)

[**2.2.3 Ejecución 4**](#_c4ulc9b9vckn)

[**2.3 Data split train test 4**](#_e5rlsl5x28a)

[**2.3.1 Parámetros 4**](#_2lpcpqf1yucm)

[**2.3.2 Ejecución 4**](#_7ahzankw3to8)

[**2.3.3 Generación de datos de validación y entrenamiento 4**](#_qmr19m3iic28)

[**2.3.4 Ejecución 5**](#_46q4qqqyybpt)

[**2.4 Modelo 5**](#_prgxo315qo53)

[**2.4.2 Comando 5**](#_bij4lk39u8d0)

[**2.4.3 Ejecución 5**](#_qv2cfjfboroi)

[**2.5 Compilación del modelo 5**](#_xy61pt2j7ol5)

[**2.5.1 Comando 5**](#_5xl3c7ck3dov)

[**2.6 Entrenamiento 6**](#_vgkhcteptizl)

[**2.6.1 Comando 6**](#_v5h484daqob5)

[**2.6.2 Ejecución 6**](#_1np4e0gqjrsa)

[**2.7 Gráficas 6**](#_stzy375827wu)

[**2.7.2 Gráfica de pérdida 6**](#_f2f2a2ppq72g)

[**2.7.2 Gráfica de error de red 6**](#_fi3jddj6u8tx)

[**2.7.2 Gráfica de precisión 7**](#_k32tcooev32n)

[**2.7.2 Gráfica de pérdida 7**](#_hptpu8h9rc56)

[**2.8 Predicción 8**](#_pz1vgqolnpdb)

[**2.8.1 Generar predicciones 8**](#_mbk9k6wznc7u)

[**2.8.2 Generar ejemplo de predicciones 8**](#_n776b6g7d3us)

[**2.8.3 Resultado 9**](#_n3lguf7fzy5h)

[**2.9 Distintas funciones de activación y optimizadores. 9**](#_3iqsnbs5yznh)

[**2.9.1 Función de activación Relu 9**](#_joe4269v54fw)

[**2.9.1.1 Código completo 9**](#_9pfn9s20d6dc)

[**2.9.1.2 Resultado con optimizador Adam 13**](#_ugjfdwiyvgmv)

[**2.9.1.2 Resultado con optimizador SGD 13**](#_ydyeo1x2v8cx)

[**2.9.1.2 Resultado con optimizador RMSprop 13**](#_rmt43v2h0hyj)

[**2.9.2 Función de activación tanh 14**](#_6vcz5pnykbjq)

[**2.9.2.1 Código completo 14**](#_vesa0jll4zy2)

[**2.9.2.2 Resultado con optimizador Adam 18**](#_z4i6h4q9pcl4)

[**2.9.2.2 Resultado con optimizador SGD 18**](#_awlf42pvqc86)

[**2.9.2.2 Resultado con optimizador RMSprop 19**](#_c4vxbjomxv20)

[**2.9.2.3 Resultado 19**](#_xmc1h0dv1xxz)

[**2.9.1 Función de activación Sigmoid 20**](#_4deprdpajzbu)

[**2.9.1.1 Código completo 20**](#_opq38713gtve)

[**2.9.1.2 Resultado con optimizador Adam 23**](#_hphonaphfdh2)

[**2.9.1.2 Resultado con optimizador SGD 23**](#_lmujz3f4hf98)

[**2.9.1.2 Resultado con optimizador RMSprop 24**](#_856uwwuk0ss2)

[**2.9.1.3 Resultado 25**](#_u2xsz6xpoqb2)

[**2.9 Técnicas de regularización 25**](#_a8qalq7jlqfp)

[**3. Github y Colab 26**](#_lmdngf8dlin3)

## 

## 

## 

## 

## **1. Enunciado**

Descarga el Chessman image dataset de:

<https://www.kaggle.com/datasets/niteshfre/chessman-image-dataset?resource=download>

## **2. Código y sus apartados**

### **2.1 Librerías**

#### **2.1.2 Código**

import os

import shutil

import glob

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from tensorflow.keras.applications import VGG19

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.regularizers import l2

import kagglehub

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.layers import LeakyReLU

from tensorflow.keras.optimizers import SGD

from tensorflow.keras.optimizers import RMSprop

### **2.2 Import Dataset**

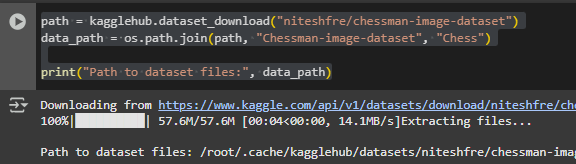
#### **2.2.2 Comando**

path = kagglehub.dataset\_download("niteshfre/chessman-image-dataset")

data\_path = os.path.join(path, "Chessman-image-dataset", "Chess")

print("Path to dataset files:", data\_path)

#### **2.2.3 Ejecución**



### **2.3 Data split train test**

Utiliza técnicas de regularización, L1, L2 y o Dropout, así como de inicialización de parámetros. Explica cómo afecta al entrenamiento.

#### **2.3.1 Parámetros**

img\_height, img\_width = 150, 150 # Dimensiones de las imágenes

batch\_size = 32

epochs = 20

learning\_rate = 0.001

#### **2.3.2 Ejecución**

#### **2.3.3 Generación de datos de validación y entrenamiento**

train\_datagen = ImageDataGenerator(

rescale=1.0/255,

validation\_split=0.2 # División entre entrenamiento y validación

)

train\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation'

)

#### **2.3.4 Ejecución**



### **2.4 Modelo**

#### **2.4.2 Comando**

base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

#for layer in base\_model.layers[:15]: # Capas de neuronas dormidas

for layer in base\_model.layers:

layer.trainable = False

model = Sequential([

base\_model,

Flatten(),

# Funciones de activación

#LeakyReLU(alpha=0.1),

#Dense(256, activation='tanh', kernel\_regularizer=l2(0.01)),

Dense(256, activation='relu', kernel\_regularizer=l2(0.01)),

Dropout(0.5),

Dense(train\_generator.num\_classes, activation='softmax')

])

#### **2.4.3 Ejecución**



### **2.5 Compilación del modelo**

#### **2.5.1 Comando**

model.compile(optimizer=Adam(learning\_rate=learning\_rate),

loss='categorical\_crossentropy',

metrics=['accuracy'])

### **2.6 Entrenamiento**

#### **2.6.1 Comando**

# Entrenamiento

history = model.fit(

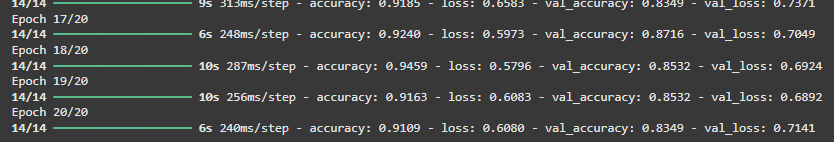
train\_generator,

validation\_data=validation\_generator,

epochs=epochs

)

#### **2.6.2 Ejecución**



### **2.7 Gráficas**

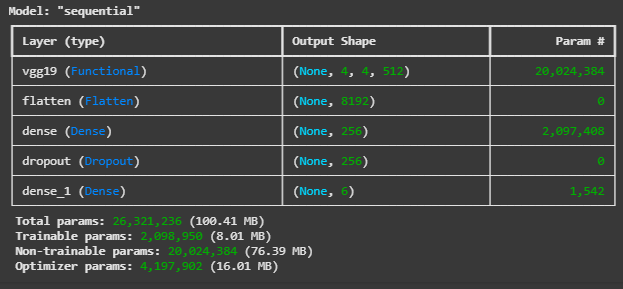
#### **2.7.2 Gráfica de pérdida**

model.summary()

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])



#### **2.7.2 Gráfica de error de red**

def plot\_acc(history, title="Model Accuracy"):

"""Imprime una gráfica mostrando la accuracy por epoch obtenida en un entrenamiento"""

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title(title)

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper left')

plt.show()

def plot\_loss(history, title="Model Loss"):

"""Imprime una gráfica mostrando la pérdida por epoch obtenida en un entrenamiento"""

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title(title)

plt.ylabel('Loss')

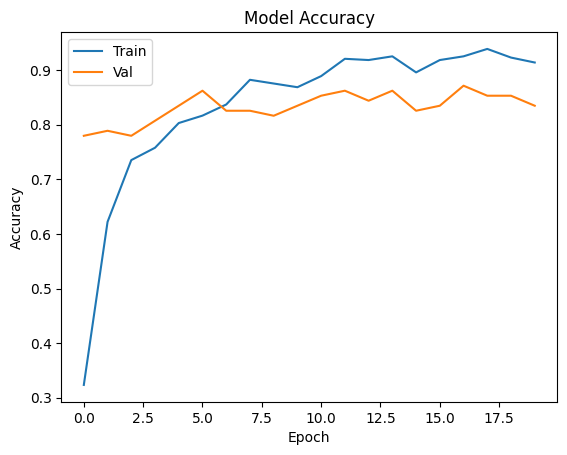
plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()

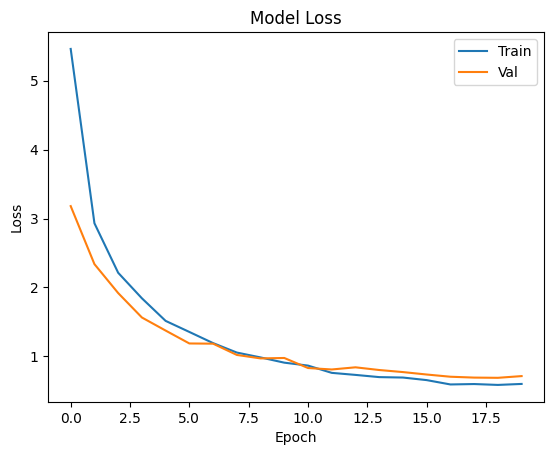
##### **2.7.2 Gráfica de precisión**

plot\_acc(history)



##### **2.7.2 Gráfica de pérdida**

plot\_loss(history)



#### 

### **2.8 Predicción**

#### **2.8.1 Generar predicciones**

# Generate predictions

validation\_generator.reset()

predictions = model.predict(validation\_generator)

y\_pred = np.argmax(predictions, axis=1)

y\_true = validation\_generator.classes



#### **2.8.2 Generar ejemplo de predicciones**

# Example prediction

sample\_image, \_ = next(validation\_generator)

predicted\_class = np.argmax(model.predict(sample\_image[0:1]))

plt.imshow(sample\_image[0])

plt.title(f"Predicted: {list(train\_generator.class\_indices.keys())[predicted\_class]}")

plt.axis('off')

plt.show()



#### **2.8.3 Resultado**



### **2.9 Distintas funciones de activación y optimizadores.**

#### **2.9.1 Función de activación Relu**

##### **2.9.1.1 Código completo**

import os

import shutil

import glob

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.applications import VGG19

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam, SGD, RMSprop

from tensorflow.keras.regularizers import l2

import kagglehub

import matplotlib.pyplot as plt

import numpy as np

# Descarga y carga del dataset

path = kagglehub.dataset\_download("niteshfre/chessman-image-dataset")

data\_path = os.path.join(path, "Chessman-image-dataset", "Chess")

print("Path to dataset files:", data\_path)

# Parámetros

img\_height, img\_width = 150, 150

batch\_size = 32

epochs = 20

# Preparación de datos

train\_datagen = ImageDataGenerator(

rescale=1.0/255,

validation\_split=0.2

)

train\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation'

)

# Función para crear un nuevo modelo

def create\_model(activation, optimizer):

base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

for layer in base\_model.layers:

layer.trainable = False

model = Sequential([

base\_model,

Flatten(),

Dense(256, activation=activation, kernel\_regularizer=l2(0.01)),

Dropout(0.5),

Dense(train\_generator.num\_classes, activation='softmax')

])

model.compile(optimizer=optimizer,

loss='categorical\_crossentropy',

metrics=['accuracy'])

return model

# Funciones de activación y optimizadores

#activation\_functions = ['relu', 'tanh', 'sigmoid']

activation\_functions = ['relu']

optimizers = {

'Adam': Adam(learning\_rate=0.001),

'SGD': SGD(learning\_rate=0.01, momentum=0.9),

'RMSprop': RMSprop(learning\_rate=0.001)

}

# Pruebas con diferentes configuraciones

for activation in activation\_functions:

for opt\_name, optimizer in optimizers.items():

print(f"Testing activation: {activation}, optimizer: {opt\_name}")

# Crear un nuevo modelo

model = create\_model(activation, optimizer)

# Entrenamiento del modelo

history = model.fit(

train\_generator,

validation\_data=validation\_generator,

epochs=epochs

)

# Evaluación del modelo

val\_loss, val\_acc = model.evaluate(validation\_generator)

print(f"Final Validation Accuracy with {activation}, {opt\_name}: {val\_acc:.4f}")

# Graficar resultados

def plot\_metrics(history, title\_suffix=""):

"""Grafica accuracy y pérdida por épocas"""

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train')

plt.plot(history.history['val\_accuracy'], label='Validation')

plt.title(f'Accuracy {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train')

plt.plot(history.history['val\_loss'], label='Validation')

plt.title(f'Loss {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

plot\_metrics(history, f"(Activation: {activation}, Optimizer: {opt\_name})")

# Evaluación por clase

predictions = model.predict(validation\_generator)

y\_pred = np.argmax(predictions, axis=1)

y\_true = validation\_generator.classes

# Matriz de confusión

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=validation\_generator.class\_indices.keys(),

yticklabels=validation\_generator.class\_indices.keys())

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# Reporte de clasificación

report = classification\_report(y\_true, y\_pred, target\_names=validation\_generator.class\_indices.keys())

print(report)

# Example prediction

sample\_image, \_ = next(validation\_generator)

predicted\_class = np.argmax(model.predict(sample\_image[0:1]))

plt.imshow(sample\_image[0])

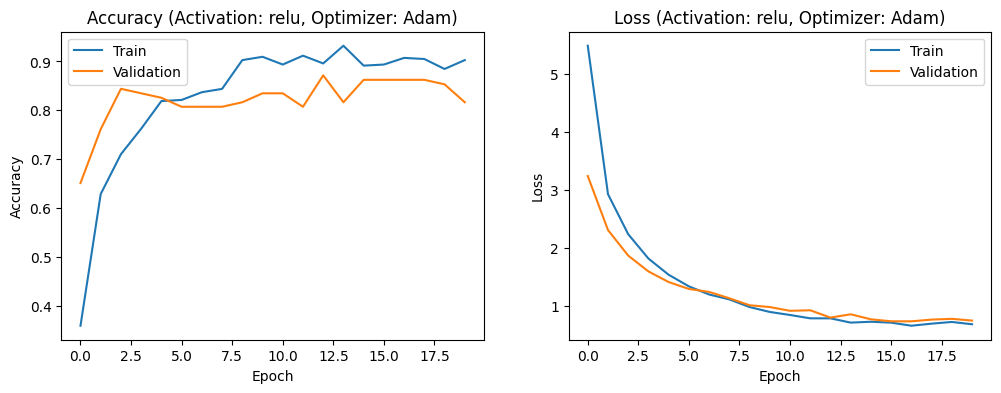
plt.title(f"Predicted: {list(train\_generator.class\_indices.keys())[predicted\_class]}")

plt.axis('off')

plt.show()

##### **2.9.1.2 Resultado con optimizador Adam**





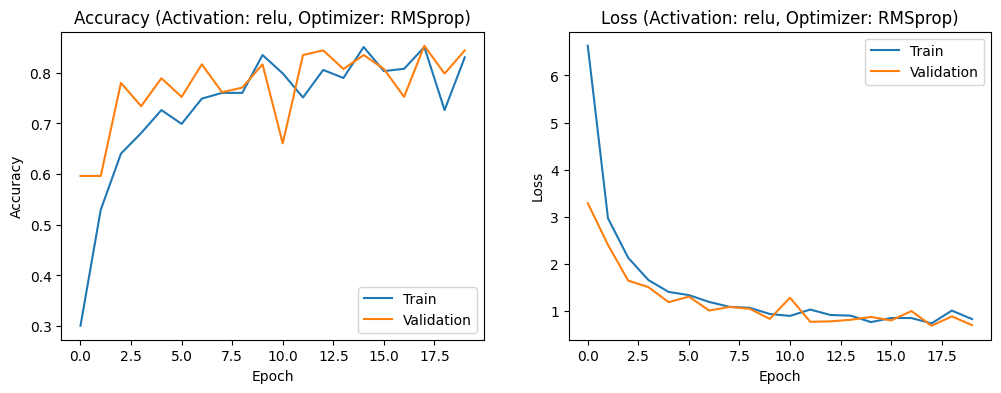
##### **2.9.1.2 Resultado con optimizador SGD**

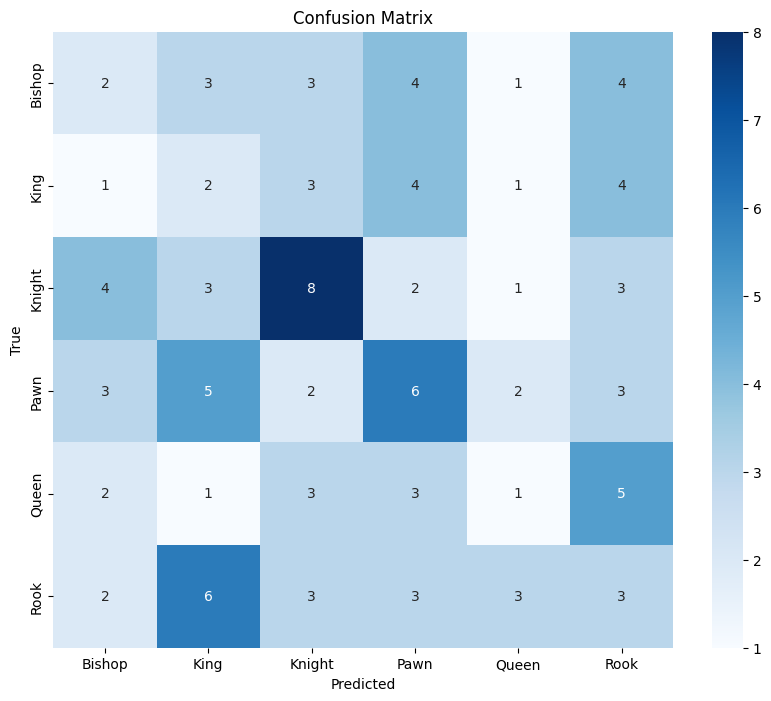




##### **2.9.1.2 Resultado con optimizador RMSprop**



**2.9.1.3 Resultado**





#### **2.9.2 Función de activación tanh**

##### **2.9.2.1 Código completo**

import os

import shutil

import glob

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.applications import VGG19

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam, SGD, RMSprop

from tensorflow.keras.regularizers import l2

import kagglehub

import matplotlib.pyplot as plt

import numpy as np

# Descarga y carga del dataset

path = kagglehub.dataset\_download("niteshfre/chessman-image-dataset")

data\_path = os.path.join(path, "Chessman-image-dataset", "Chess")

print("Path to dataset files:", data\_path)

# Parámetros

img\_height, img\_width = 150, 150

batch\_size = 32

epochs = 20

# Preparación de datos

train\_datagen = ImageDataGenerator(

rescale=1.0/255,

validation\_split=0.2

)

train\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation'

)

# Función para crear un nuevo modelo

def create\_model(activation, optimizer):

base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

for layer in base\_model.layers:

layer.trainable = False

model = Sequential([

base\_model,

Flatten(),

Dense(256, activation=activation, kernel\_regularizer=l2(0.01)),

Dropout(0.5),

Dense(train\_generator.num\_classes, activation='softmax')

])

model.compile(optimizer=optimizer,

loss='categorical\_crossentropy',

metrics=['accuracy'])

return model

# Funciones de activación y optimizadores

#activation\_functions = ['relu', 'tanh', 'sigmoid']

activation\_functions = ['tanh']

optimizers = {

'Adam': Adam(learning\_rate=0.001),

'SGD': SGD(learning\_rate=0.01, momentum=0.9),

'RMSprop': RMSprop(learning\_rate=0.001)

}

# Pruebas con diferentes configuraciones

for activation in activation\_functions:

for opt\_name, optimizer in optimizers.items():

print(f"Testing activation: {activation}, optimizer: {opt\_name}")

# Crear un nuevo modelo

model = create\_model(activation, optimizer)

# Entrenamiento del modelo

history = model.fit(

train\_generator,

validation\_data=validation\_generator,

epochs=epochs

)

# Evaluación del modelo

val\_loss, val\_acc = model.evaluate(validation\_generator)

print(f"Final Validation Accuracy with {activation}, {opt\_name}: {val\_acc:.4f}")

# Graficar resultados

def plot\_metrics(history, title\_suffix=""):

"""Grafica accuracy y pérdida por épocas"""

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train')

plt.plot(history.history['val\_accuracy'], label='Validation')

plt.title(f'Accuracy {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train')

plt.plot(history.history['val\_loss'], label='Validation')

plt.title(f'Loss {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

plot\_metrics(history, f"(Activation: {activation}, Optimizer: {opt\_name})")

# Evaluación por clase

predictions = model.predict(validation\_generator)

y\_pred = np.argmax(predictions, axis=1)

y\_true = validation\_generator.classes

# Matriz de confusión

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=validation\_generator.class\_indices.keys(),

yticklabels=validation\_generator.class\_indices.keys())

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

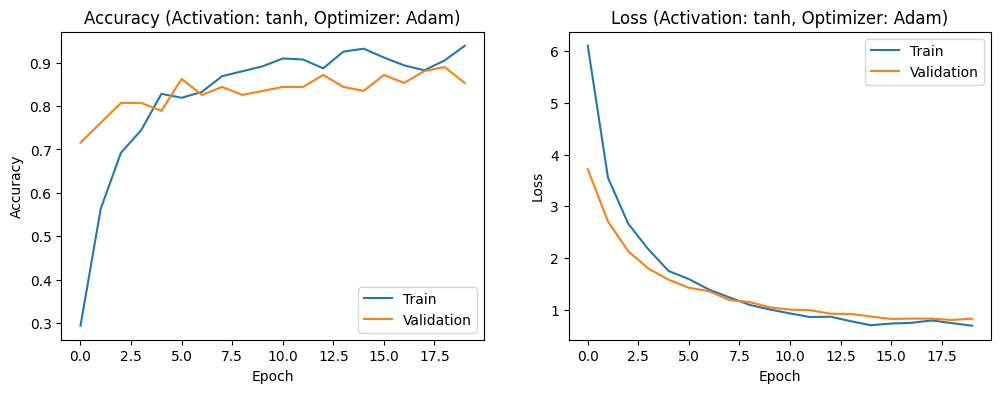
# Reporte de clasificación

report = classification\_report(y\_true, y\_pred, target\_names=validation\_generator.class\_indices.keys())

print(report)

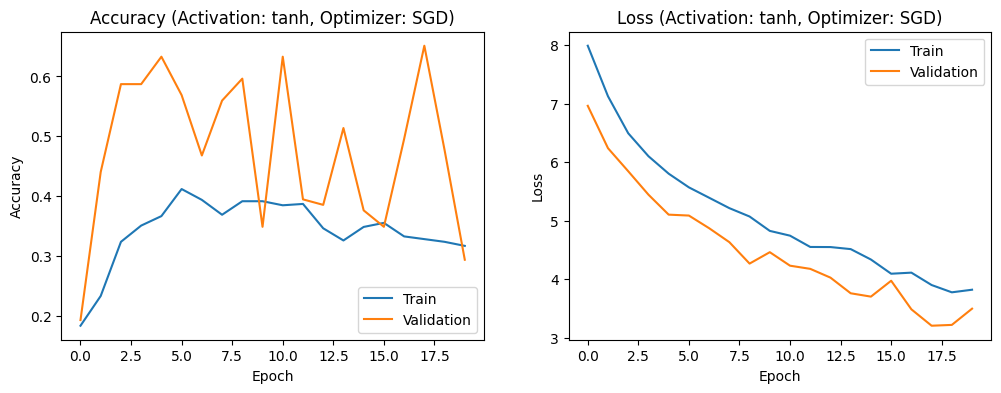
##### **2.9.2.2 Resultado con optimizador Adam**





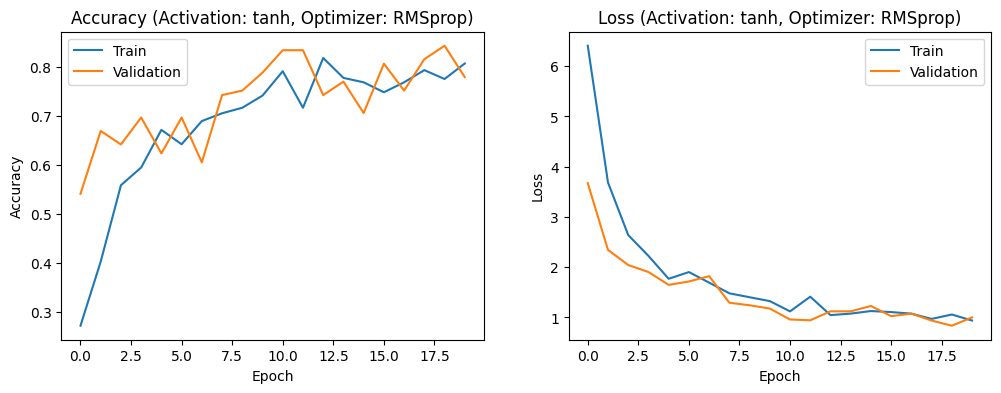
##### **2.9.2.2 Resultado con optimizador SGD**



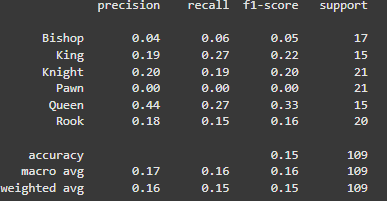
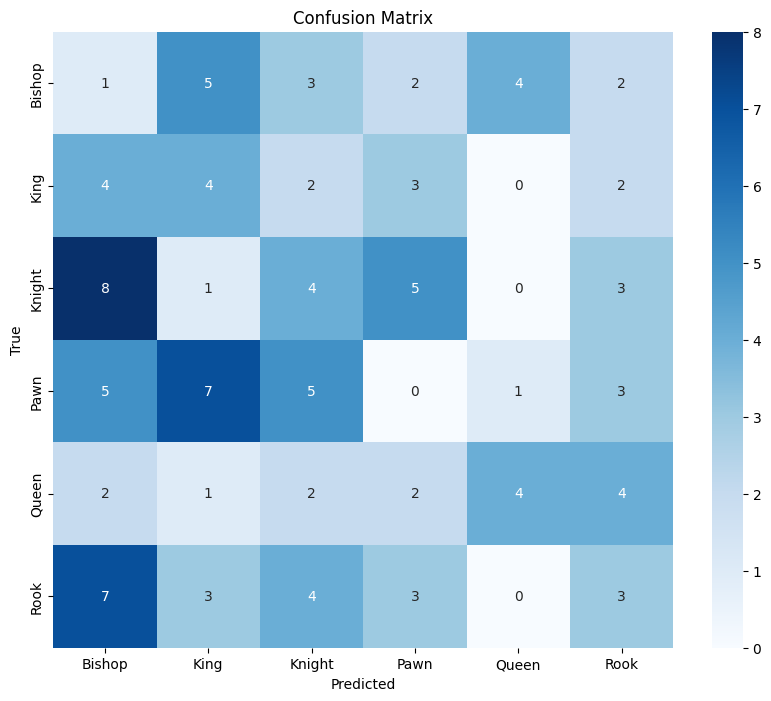


##### **2.9.2.2 Resultado con optimizador RMSprop**





##### **2.9.2.3 Resultado**



#### **2.9.1 Función de activación Sigmoid**

##### **2.9.1.1 Código completo**

import os

import shutil

import glob

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.applications import VGG19

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam, SGD, RMSprop

from tensorflow.keras.regularizers import l2

import kagglehub

import matplotlib.pyplot as plt

import numpy as np

# Descarga y carga del dataset

path = kagglehub.dataset\_download("niteshfre/chessman-image-dataset")

data\_path = os.path.join(path, "Chessman-image-dataset", "Chess")

print("Path to dataset files:", data\_path)

# Parámetros

img\_height, img\_width = 150, 150

batch\_size = 32

epochs = 20

# Preparación de datos

train\_datagen = ImageDataGenerator(

rescale=1.0/255,

validation\_split=0.2

)

train\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

data\_path,

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation'

)

# Función para crear un nuevo modelo

def create\_model(activation, optimizer):

base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

for layer in base\_model.layers:

layer.trainable = False

model = Sequential([

base\_model,

Flatten(),

Dense(256, activation=activation, kernel\_regularizer=l2(0.01)),

Dropout(0.5),

Dense(train\_generator.num\_classes, activation='softmax')

])

model.compile(optimizer=optimizer,

loss='categorical\_crossentropy',

metrics=['accuracy'])

return model

# Funciones de activación y optimizadores

#activation\_functions = ['relu', 'tanh', 'sigmoid']

activation\_functions = ['sigmoid']

optimizers = {

'Adam': Adam(learning\_rate=0.001),

'SGD': SGD(learning\_rate=0.01, momentum=0.9),

'RMSprop': RMSprop(learning\_rate=0.001)

}

# Pruebas con diferentes configuraciones

for activation in activation\_functions:

for opt\_name, optimizer in optimizers.items():

print(f"Testing activation: {activation}, optimizer: {opt\_name}")

# Crear un nuevo modelo

model = create\_model(activation, optimizer)

# Entrenamiento del modelo

history = model.fit(

train\_generator,

validation\_data=validation\_generator,

epochs=epochs

)

# Evaluación del modelo

val\_loss, val\_acc = model.evaluate(validation\_generator)

print(f"Final Validation Accuracy with {activation}, {opt\_name}: {val\_acc:.4f}")

# Graficar resultados

def plot\_metrics(history, title\_suffix=""):

"""Grafica accuracy y pérdida por épocas"""

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train')

plt.plot(history.history['val\_accuracy'], label='Validation')

plt.title(f'Accuracy {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train')

plt.plot(history.history['val\_loss'], label='Validation')

plt.title(f'Loss {title\_suffix}')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

plot\_metrics(history, f"(Activation: {activation}, Optimizer: {opt\_name})")

# Evaluación por clase

predictions = model.predict(validation\_generator)

y\_pred = np.argmax(predictions, axis=1)

y\_true = validation\_generator.classes

# Matriz de confusión

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=validation\_generator.class\_indices.keys(),

yticklabels=validation\_generator.class\_indices.keys())

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

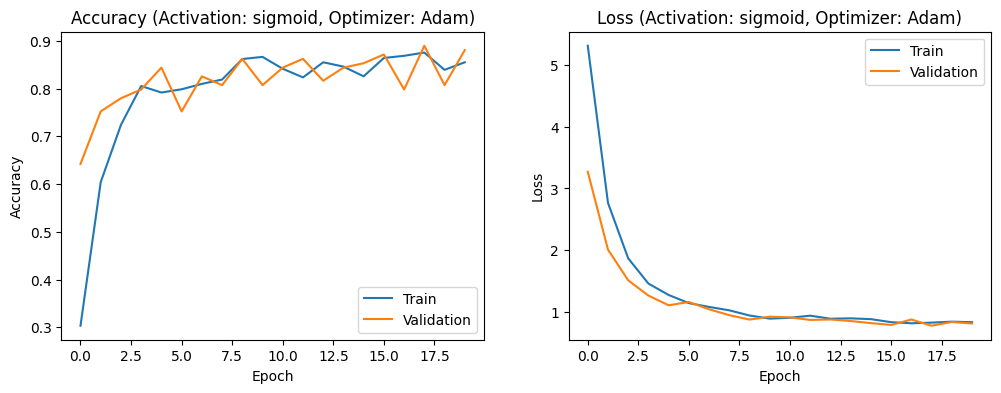
# Reporte de clasificación

report = classification\_report(y\_true, y\_pred, target\_names=validation\_generator.class\_indices.keys())

print(report)

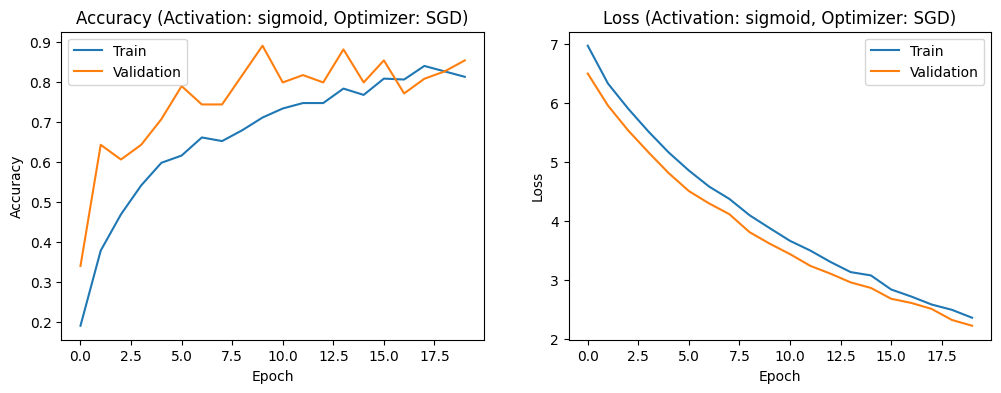
##### **2.9.1.2 Resultado con optimizador Adam**





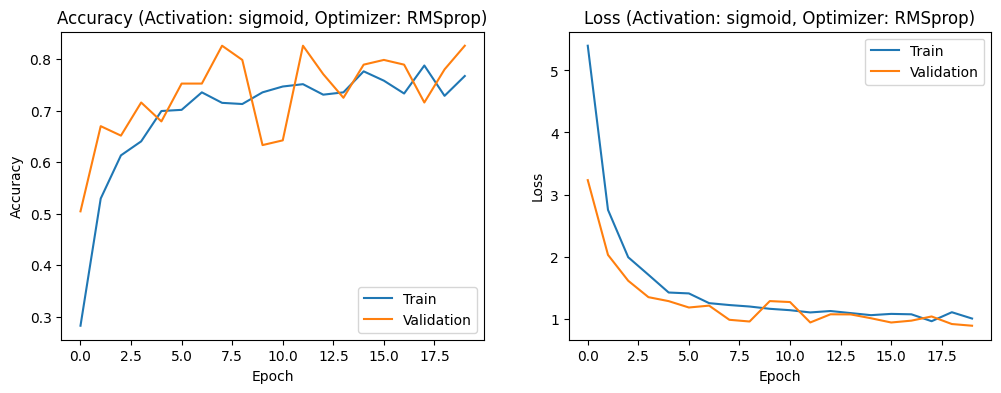
##### **2.9.1.2 Resultado con optimizador SGD**



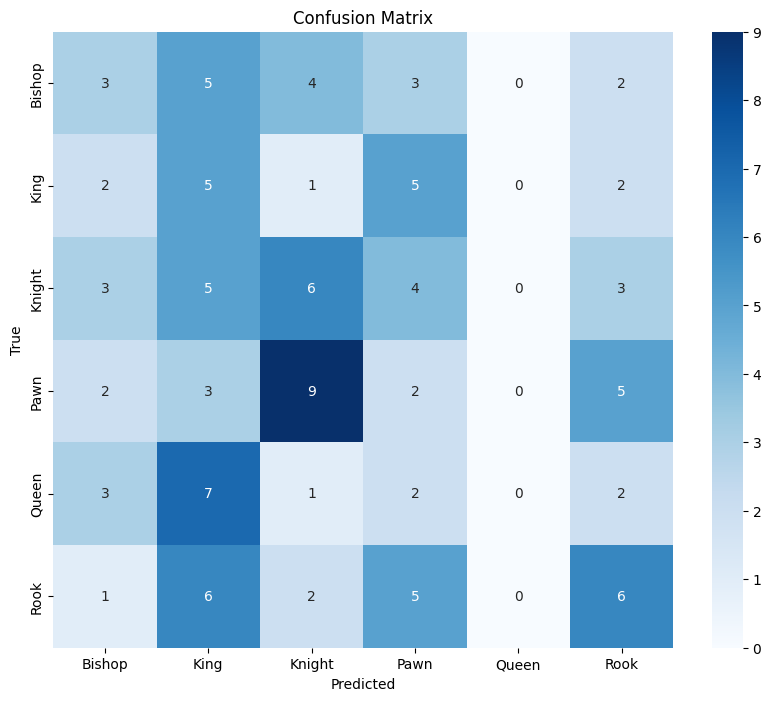


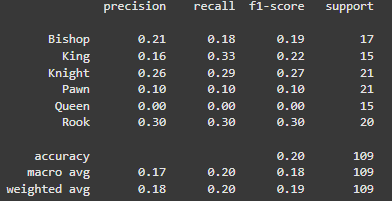
##### **2.9.1.2 Resultado con optimizador RMSprop**





##### **2.9.1.3 Resultado**





### **2.9 Técnicas de regularización**

Utiliza técnicas de regularización, L1, L2 y o Dropout, así como de inicialización de parámetros. Explica cómo afecta al entrenamiento.

**En cuanto a L2 he creado esta línea de código:**

Dense(256, activation='relu', kernel\_regularizer=l2(0.01))

Controla el sobreajuste al mantener los pesos pequeños.

Promueve un modelo más suave y menos propenso a aprender patrones ruidosos.

**En cuanto a Dropout he creado:**

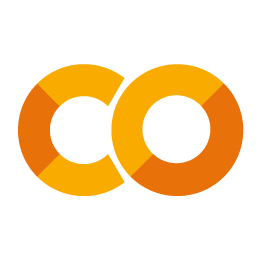
Dropout(0.5)

Durante el entrenamiento, desactiva aleatoriamente el 50% de las neuronas en esta capa.

Evita que las neuronas dependan excesivamente unas de otras (co-adaptación).

Mejora la capacidad del modelo para generalizar a nuevos datos.

## **3. Github y Colab**

[](https://colab.research.google.com/drive/1EhMai7vHlPAtpJjBvlJsuHfdq_oo5ctx?usp=sharing)