Chess Data Set

Adrián Yared Armas de la Nuez

Contenido

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

```
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)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/niteshfre/ch
100%|██████████| 57.6M/57.6M [00:04<00:00, 14.1MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/niteshfre/chessman-ima
```

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

```
Found 442 images belonging to 6 classes.
Found 109 images belonging to 6 classes.
```

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

```
Downloading data from https://storage.googleapis.com/tenso
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

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

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 9s 313ms/step - accuracy: 0.9185 - loss: 0.6583 - val_accuracy: 0.8349 - val_loss: 0.7371
Epoch 17/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 6s 248ms/step - accuracy: 0.9240 - loss: 0.5973 - val_accuracy: 0.8716 - val_loss: 0.7049
Epoch 18/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 10s 287ms/step - accuracy: 0.9459 - loss: 0.5796 - val_accuracy: 0.8532 - val_loss: 0.6924
Epoch 19/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 10s 256ms/step - accuracy: 0.9163 - loss: 0.6083 - val_accuracy: 0.8532 - val_loss: 0.6892
Epoch 20/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 6s 240ms/step - accuracy: 0.9109 - loss: 0.6080 - val_accuracy: 0.8349 - val_loss: 0.7141
```

2.7 Gráficas

2.7.2 Gráfica de pérdida

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

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 4, 4, 512)	20,024,384
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2,097,408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 6)	1,542

Total params: 26,321,236 (100.41 MB)
Trainable params: 2,098,950 (8.01 MB)
Non-trainable params: 20,024,384 (76.39 MB)
Optimizer params: 4,197,902 (16.01 MB)

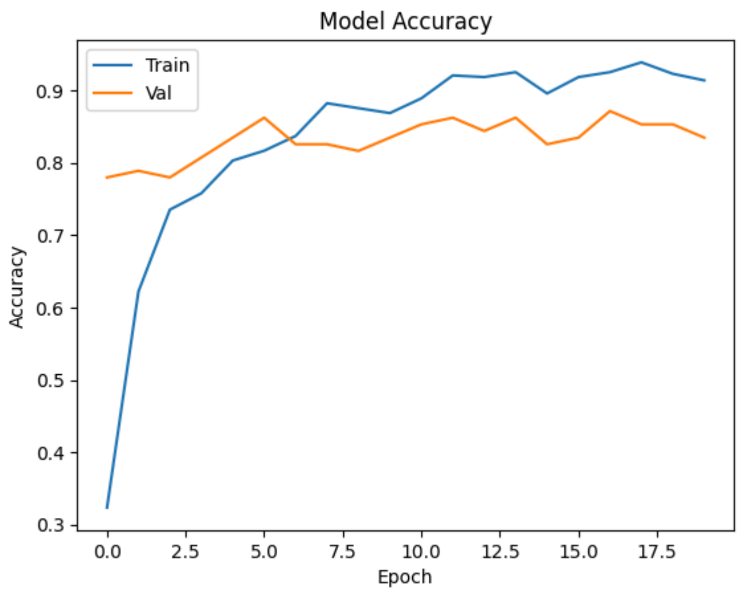
2.7.2 Gráfica de error de red

```
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()
```

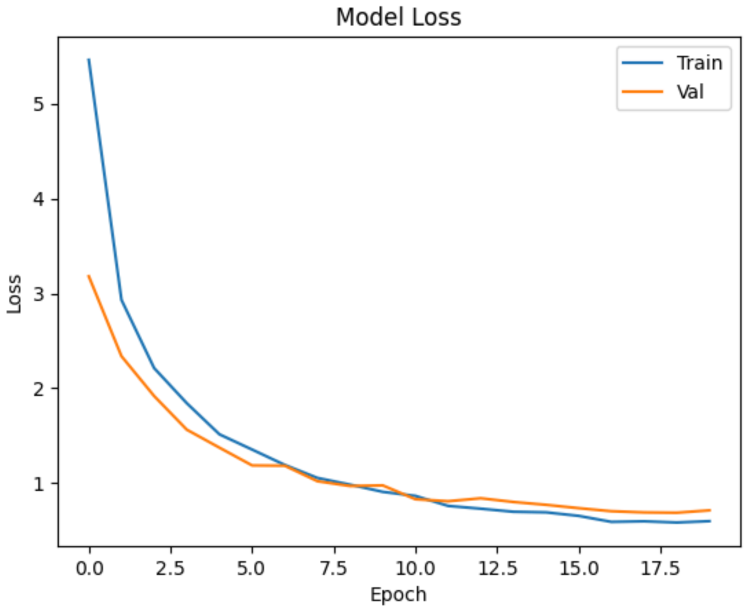
2.7.2 Gráfica de precisión

```
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
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 482ms/step
```

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()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

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,
```

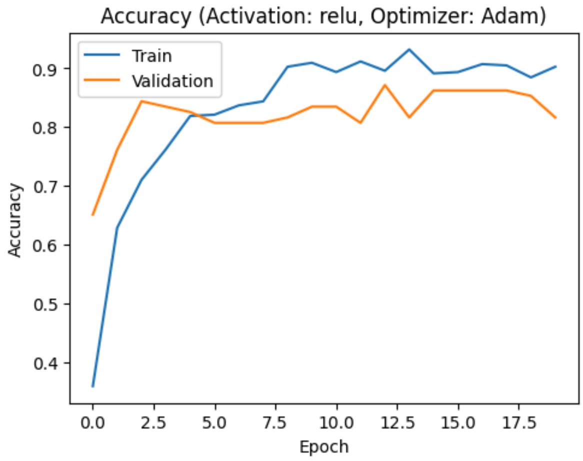
```
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    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')
```

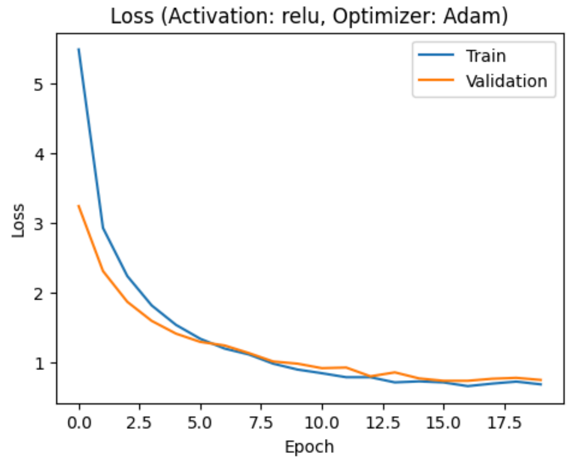
```
            plt.ylabel('Accuracy')
            plt.legend()

            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

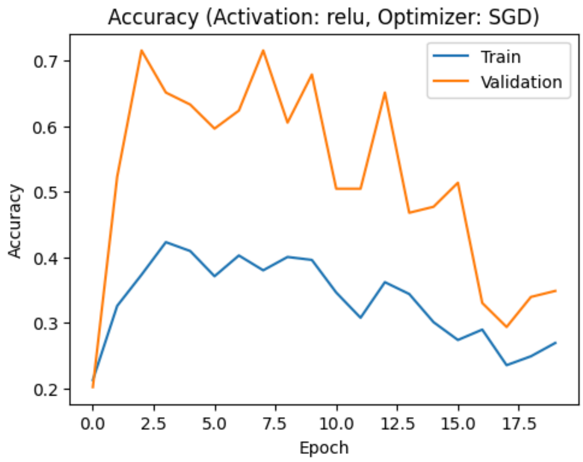
```
Final Validation Accuracy with relu, Adam: 0.8165
```

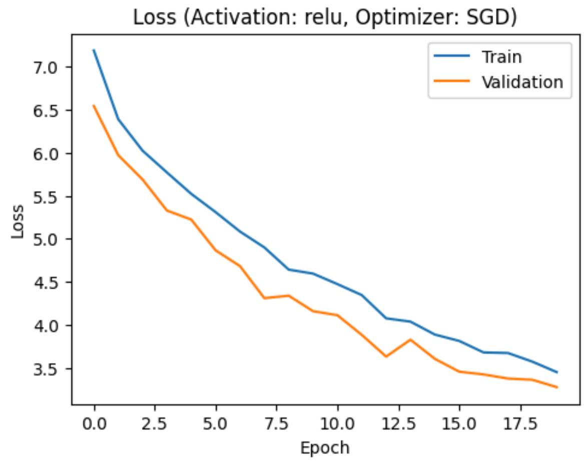




2.9.1.2 Resultado con optimizador SGD

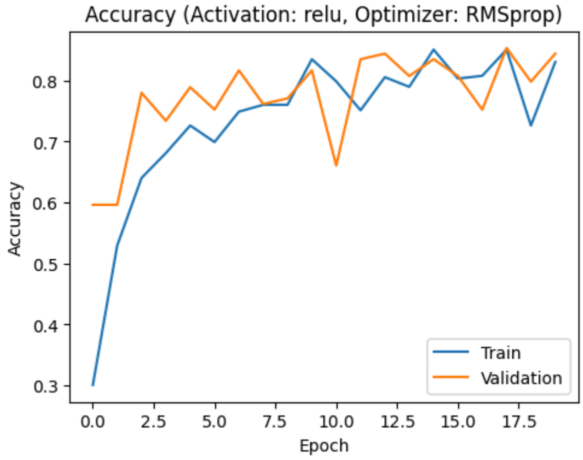
```
Final Validation Accuracy with relu, SGD: 0.3486
```

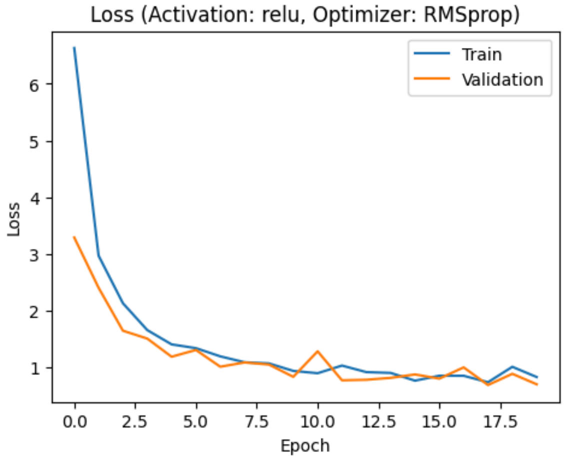




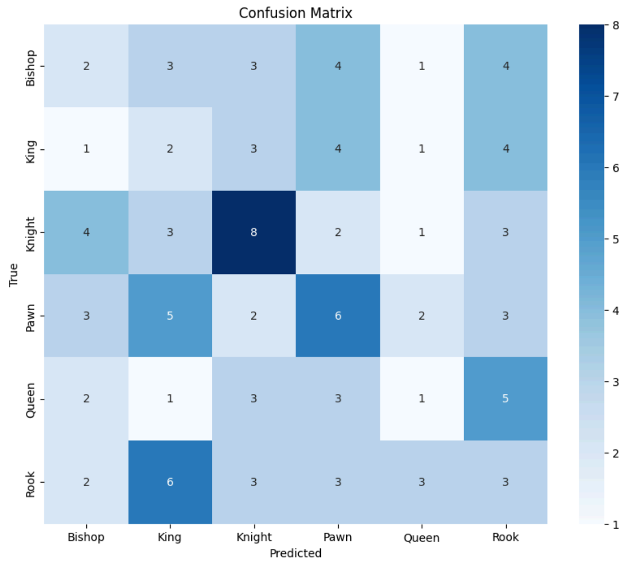
2.9.1.2 Resultado con optimizador RMSprop

```
Final Validation Accuracy with relu, RMSprop: 0.8440
```





2.9.1.3 Resultado




```
              precision    recall  f1-score   support

      Bishop       0.14      0.12      0.13        17
        King       0.10      0.13      0.11        15
      Knight       0.36      0.38      0.37        21
        Pawn       0.27      0.29      0.28        21
       Queen       0.11      0.07      0.08        15
        Rook       0.14      0.15      0.14        20

    accuracy                           0.20       109
   macro avg       0.19      0.19      0.19       109
weighted avg       0.20      0.20      0.20       109

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

2.9.2 Función de activación tanh

2.9.2.1 Código completo

```
import os
import shutil
import glob
```

```python
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'
```

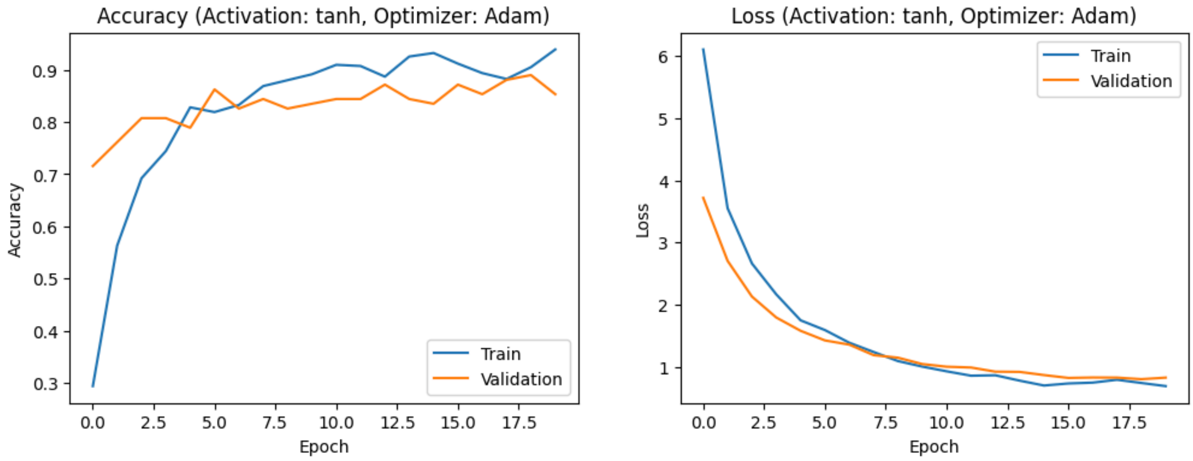
```
)

# 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)
```

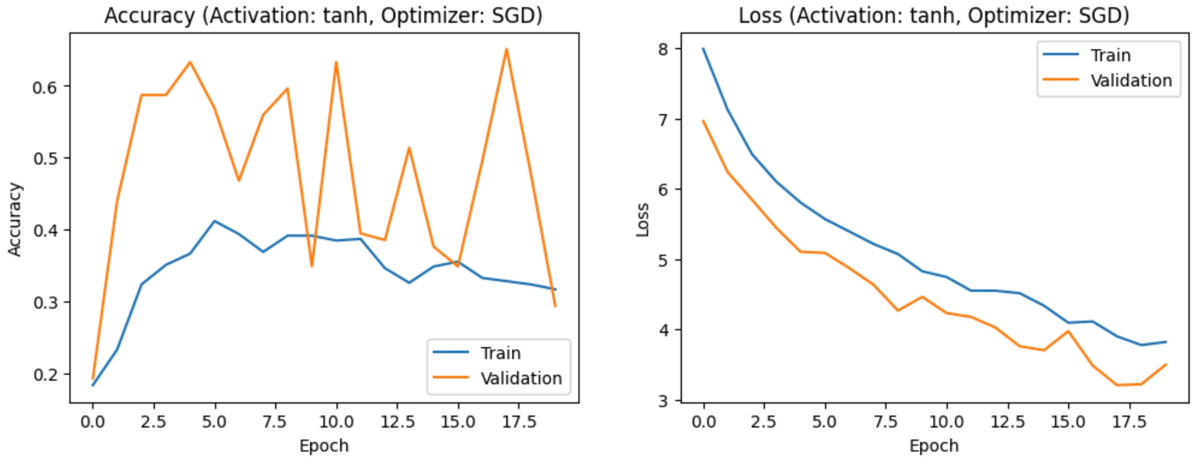
2.9.2.2 Resultado con optimizador Adam

```
Final Validation Accuracy with tanh, Adam: 0.8532
```



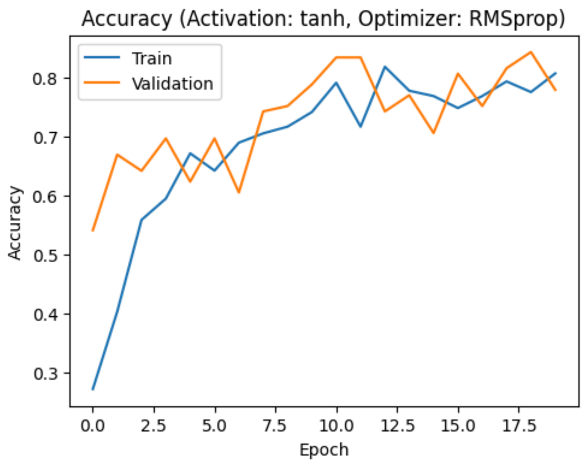
2.9.2.2 Resultado con optimizador SGD

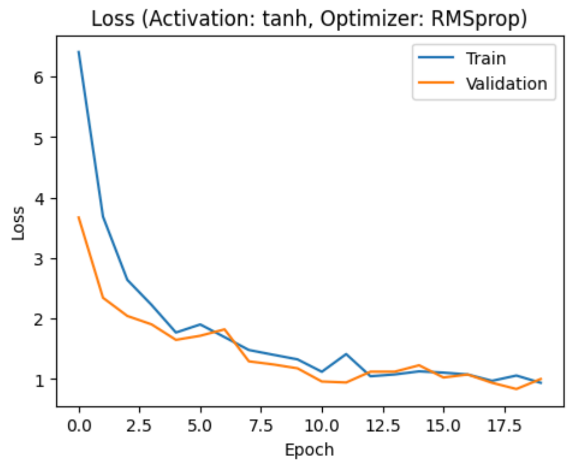
```
Final Validation Accuracy with tanh, SGD: 0.2936
```



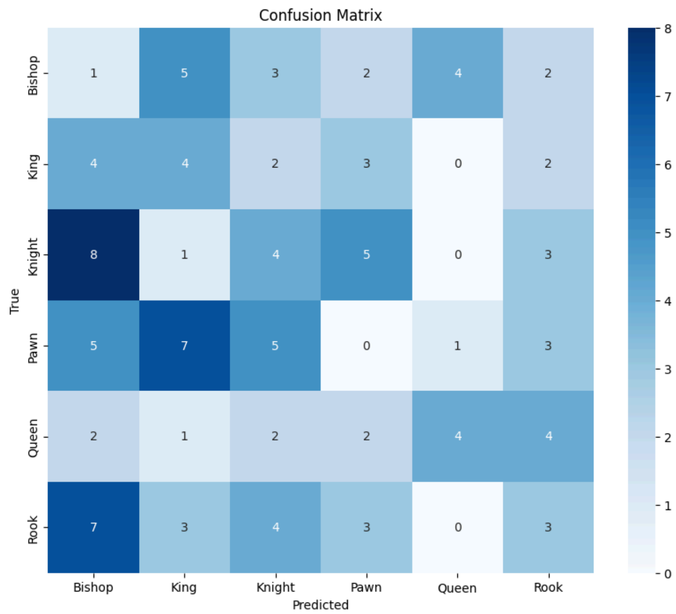
2.9.2.2 Resultado con optimizador RMSprop

```
Final Validation Accuracy with tanh, RMSprop: 0.7798
```





2.9.2.3 Resultado



Predicted: Knight

	precision	recall	f1-score	support
Bishop	0.04	0.06	0.05	17
King	0.19	0.27	0.22	15
Knight	0.20	0.19	0.20	21
Pawn	0.00	0.00	0.00	21
Queen	0.44	0.27	0.33	15
Rook	0.18	0.15	0.16	20
accuracy			0.15	109
macro avg	0.17	0.16	0.16	109
weighted avg	0.16	0.15	0.15	109

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'
```

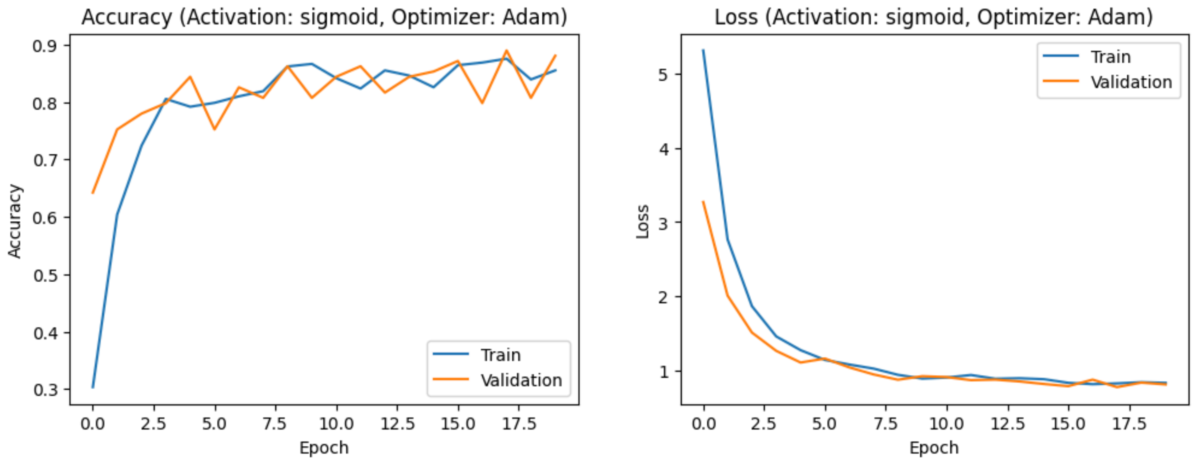
```
)

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)
```

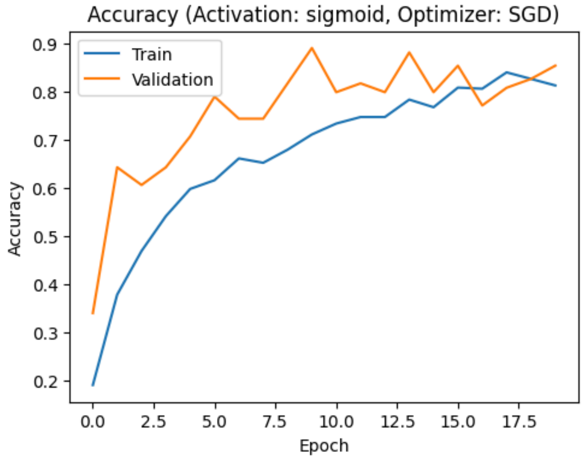
2.9.1.2 Resultado con optimizador Adam

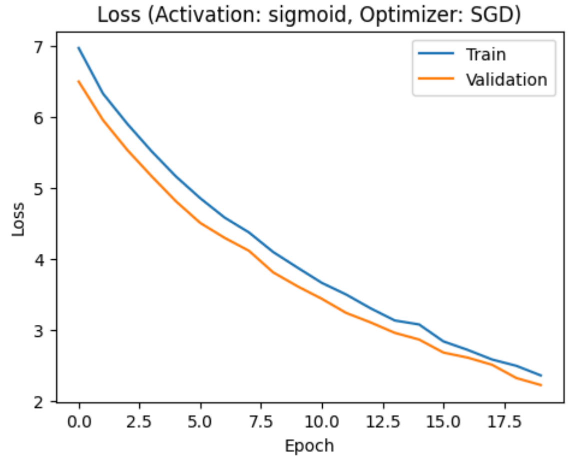
```
Final Validation Accuracy with sigmoid, Adam: 0.8807
```



2.9.1.2 Resultado con optimizador SGD

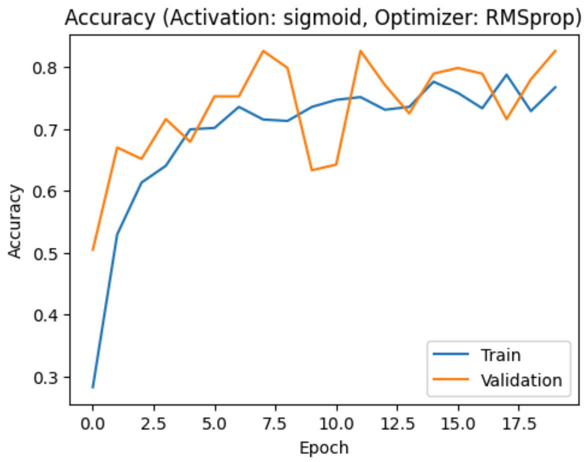
```
Final Validation Accuracy with sigmoid, SGD: 0.8532
```

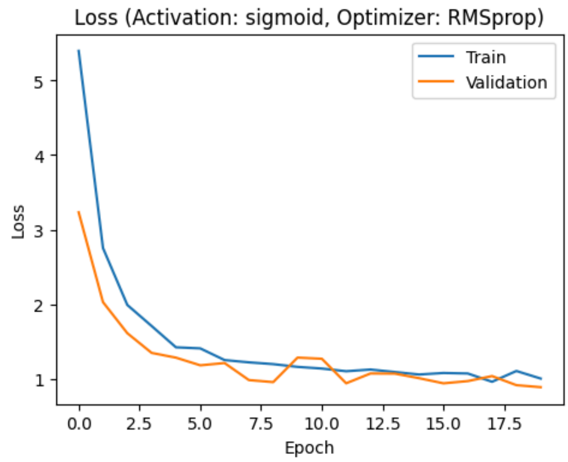




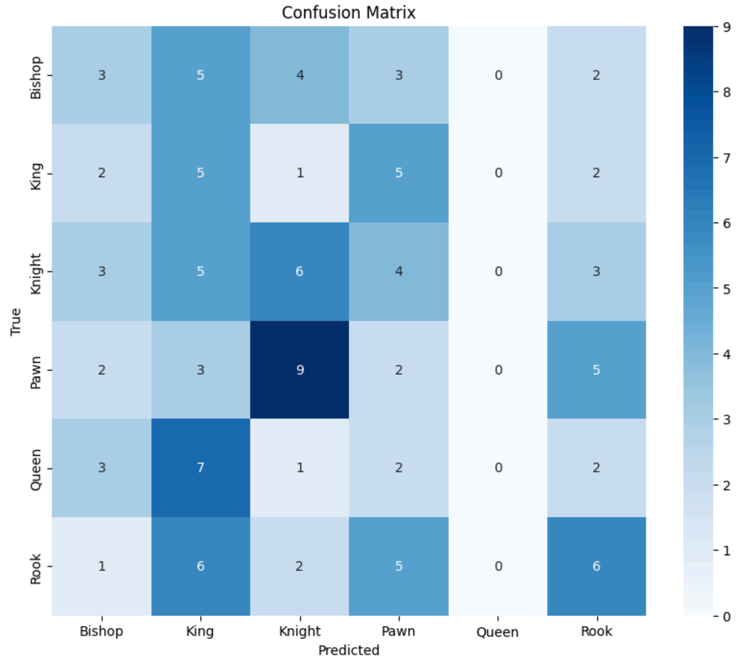
2.9.1.2 Resultado con optimizador RMSprop

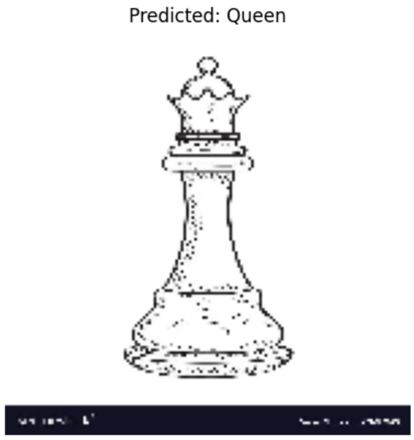
```
Final Validation Accuracy with sigmoid, RMSprop: 0.8257
```





2.9.1.3 Resultado





	precision	recall	f1-score	support
Bishop	0.21	0.18	0.19	17
King	0.16	0.33	0.22	15
Knight	0.26	0.29	0.27	21
Pawn	0.10	0.10	0.10	21
Queen	0.00	0.00	0.00	15
Rook	0.30	0.30	0.30	20
accuracy			0.20	109
macro avg	0.17	0.20	0.18	109
weighted avg	0.18	0.20	0.19	109

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