ALUMNO:

Anatanael Jesús Miranda Faustino A01769232

PROFESOR
Benjamín Valdés Aguirre

Módulo 2 Análisis y Reporte

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

SEMESTRE Agosto 2023

Para poder comprender los algoritmos de aprendizaje, se realizaron dos versiones: una implementación hecha solo con Pandas y NumPy, y otra con un framework. En la implementación que carece de framework, se notó la complejidad que se presenta cuando no se cuenta con la ayuda de ninguna biblioteca adicional. Además, el modelo de predicción era muy básico, ya que solo se implementó una regresión lineal mediante el uso de gradiente descendente.

En contraste, la implementación con framework es más compleja, ya que pudimos apoyarnos en la librería TensorFlow y Keras, lo que nos permitió explorar diferentes inicializaciones, algoritmos de optimización como Adam y funciones de activación como ReLU. Este último es el modelo que elegimos para analizar.

La separación de nuestros set de datos.

```
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    path,
    validation_split=0.2,  # 20% para validación
    image_size=(224, 224),
    batch_size=32,
    subset="test",
    seed=42
)


test_data = tf.keras.preprocessing.image_dataset_from_directory(
    path,
    validation_split=0.2,  # 20% para prueba
    image_size=(224, 224),
    batch_size=32,
```

```
    subset="validation",

    seed=42

)
```

Una vez que tenemos la carga de imágenes, realizamos un split para dividir nuestro set de datos en 60% entrenamiento, 20% prueba y 20% validación. Esto lo hacemos porque el set de datos no es muy grande; para cada clase, contamos con 139 elementos, lo que no es una cantidad muy grande para la clasificación que queremos realizar. Por lo tanto, damos prioridad a la parte de entrenamiento. Además, nos apoyamos en el valor de la semilla (seed) para generar nuestros números aleatorios, que será igual a 42.

Nuestro modelo

```
mobilenet = MobileNet(include_top=True, weights='imagenet',input_tensor
= (upscale), input_shape=(224, 224, 3))



data_augmentation = tf.keras.Sequential(

  [


tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),

    tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),

    tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),

  ]

)
```

Hacemos uso del módulo Sequential, el cual se utiliza para generar variantes de las imágenes originales mediante la aplicación de transformaciones aleatorias. Establecemos algunos atributos que nos ayudarán en el aprendizaje del modelo:

- RandomFlip("horizontal"): Realiza una operación de volteo aleatorio horizontal en las imágenes con una probabilidad del 50%.
- RandomRotation(0.1): Aplica una rotación aleatoria a las imágenes con un ángulo máximo de 0.1 radianes. La magnitud de la rotación es pequeña, lo que ayuda al modelo a ser más robusto ante las variaciones en la orientación de los objetos en las imágenes.

- RandomZoom(0.1): Realiza un zoom aleatorio. Todo esto se aplica porque nuestro set de datos no es muy grande, por lo que recurrimos a estos pasos para hacer el modelo más robusto.

Capas del modelo

```
model = tf.keras.Sequential([

    Rescaling(1./255, input_shape=(224, 224, 3)),

    data_augmentation,

    mobilenet,

    Flatten(),

    Dense(128, activation='relu'),

    Dense(len(class_names), activation='softmax')])
```

Para comenzar a explicar el modelo, es importante señalar que cree una matriz multidimensional (tensor) con forma (224, 224, 3). Esto se debe a las dimensiones de los píxeles y a los tres canales que proporciona el formato RGB. En cuanto a las capas de mi modelo de redes:

Realize un proceso de Rescaling para normalizar la entrada de nuestro modelo en un rango de valores de 0 a 1, evitando así problemas de escalabilidad.

Incorpore MobileNet, una ventaja que nos brinda el framework. MobileNet es una red neuronal convolucional previamente entrenada, se utiliza como base para la transferencia de aprendizaje, permitiéndo extraer características clave de las imágenes.

A continuación, añadimos una capa Flatten. Esta capa se utiliza para convertir las características extraídas por MobileNet en un vector unidimensional.

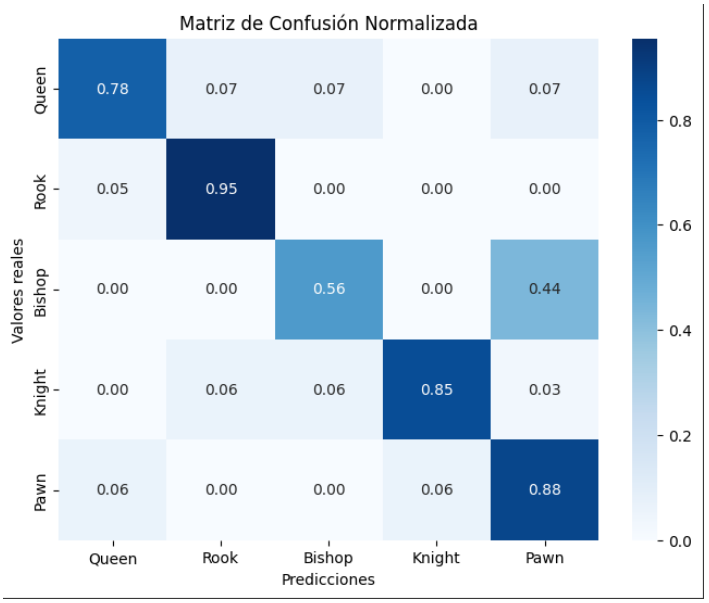
Segui con dos capas totalmente conectadas (Densas), diferenciadas únicamente por la cantidad de neuronas y las funciones de activación utilizadas:

- Dense con 128 neuronas y una función de activación 'ReLU'. Las capas densas se emplean para realizar cálculos y aprender representaciones más complejas de las características de entrada.
- Dense con 5 neuronas y función de activación softmax. Esta capa consta de 5 neuronas, ya que representa las 5 clases de salida del modelo. La activación softmax se emplea comúnmente en problemas de clasificación multiclase para calcular las probabilidades de pertenencia a cada clase.

Así configuramos el modelo, que se basa en la arquitectura de MobileNet y se adapta a las necesidades de la tarea específica.

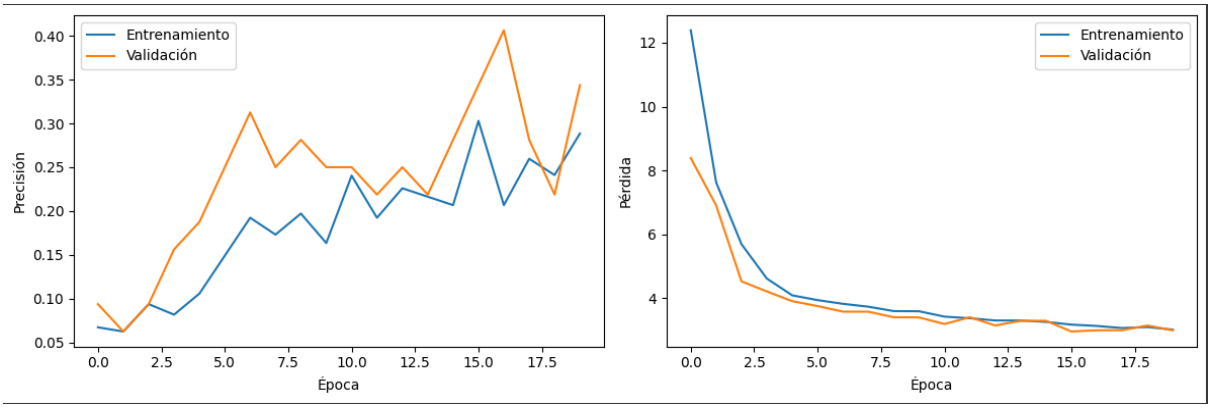
Sesgo

El sesgo del modelo de clasificación se percibe medio, ya que tiene una tendencia a confundir incorrectamente las clases "Pawn" y "Bishop". Esto nos indica que hay un rendimiento desigual en diferentes clases. Para corregir este sesgo, se necesita considerar la distribución de clases en los datos y cómo el modelo se comporta en cada una de ellas.



Aquí se puede ver que tiene tendencia a equivocar "bishop" con "pawn". Además, vemos que a pesar de que los valores entre clases son bajos, tampoco se alcanza la referencia de "bishop", por lo que decimos que tiene un sesgo **medio-bajo.**

Varianza



Se presenta una varianza **alta** ya que se tiene una gran variabilidad en la precisión donde se observa como esta disminuye y aumenta de acuerdo a la iteración de cada epoch, no alcanzado una suavidad y estabilidad.

Diagnóstico y explicación el nivel de ajuste del modelo:

El modelo de reconocimiento de piezas de ajedrez ha demostrado que aún no está completamente ajustado, es decir, está "**underfit**".

```
17/17 [==============================] - 112s 7s/step - loss: 0.5343 - accuracy: 0.8253 - val_loss: 1.0236 - val_accuracy: 0.7000
Epoch 59/60
17/17 [==============================] - 112s 7s/step - loss: 0.4806 - accuracy: 0.8522 - val_loss: 0.6009 - val_accuracy: 0.7846
Epoch 60/60
17/17 [==============================] - 112s 7s/step - loss: 0.4667 - accuracy: 0.8215 - val_loss: 0.5740 - val_accuracy: 0.7846
```

El modelo ha logrado un considerable nivel de precisión en el conjunto de datos de entrenamiento, lo que indica que ha aprendido bien las características y patrones presentes en estas imágenes de piezas de ajedrez. Sin embargo, el nivel de pérdida que muestra el conjunto de entrenamiento es considerable, lo que sugiere que el modelo no se ha ajustado de manera efectiva a los datos de entrenamiento, creando discrepancia entre las predicciones y las etiquetas reales. La gráfica de la pérdida con respecto a los epochs muestra el punto de convergencia del modelo, punto donde tenemos un error bajo.

La pérdida en el conjunto de validación es coherente y no está por encima de la pérdida en el conjunto de entrenamiento. Tenemos margen para mejoras. Se podría considerar estrategias como la recopilación de datos adicionales, el aumento de datos, la optimización de hiperparámetros o incluso el uso de arquitecturas de modelos más avanzadas para mejorar aún más el rendimiento.

Mejoras

El modelo tiene margen para mejoras. Podríamos considerar estrategias como la recopilación de datos adicionales, ya que un conjunto de datos de alta calidad es fundamental para entrenar un modelo de aprendizaje automático eficaz. El set de datos se queda algo corto y, además, no todas las fotografías son de calidad; se podría decir que varias de estas tienen ruido. Los datos son la piedra angular de cualquier modelo, ya que en base a estos, nuestro modelo aprende patrones y toma decisiones.

Un conjunto de datos limpio, diverso y representativo garantiza que el modelo generalice bien a nuevas situaciones y produzca resultados precisos y confiables en aplicaciones del mundo real. También podría mejorar el modelo probando diferentes modelos preentrenados como ImageNet y agregar capas de ajuste para evitar el underfit y el overfit o incluso modificando la cantidad de capas y neuronas por capa para así crear modelos más avanzados y mejorar aún más el rendimiento.

Después de las mejoras

Consideramos que lo mejor es aumentar nuestro set de datos; sin embargo, debido a la falta de personal para recabar y etiquetar dichos datos, optamos por hacer aún más robusto el modelo.

```
baseModel = tf.keras.applications.VGG16(

    include_top=False,

    weights="imagenet",

    input_shape=(width, height, 3))
```

cambie la arquitectura del modelo por una arquitectura VGG16, que es una red profunda con muchas capas convolucionales, además de usar pesos preentrenados de VGG 16 entrenados en el conjunto de datos ImageNet.

Nuevo modelo

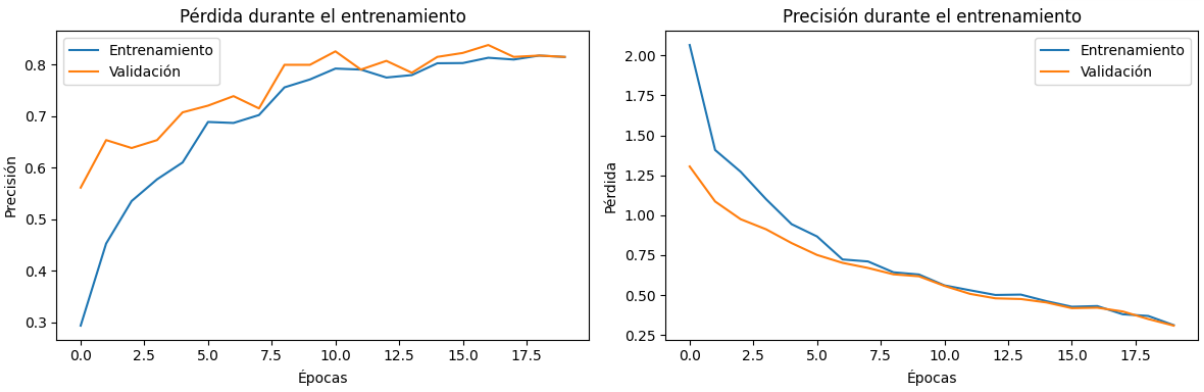
```


model = tf.keras.Sequential([

    Rescaling(1./255, input_shape=(224, 224, 3)),

    baseModel,

    BatchNormalization(),

    Flatten(),
```

```
    Dense(64, activation='relu'),

    Dropout(0.3),

    Dense(128, activation='relu'),

    Dense(len(class_names), activation='softmax')

])
```

El nuevo modelo ya no incluye la parte de keras.Sequential debido al uso de ImageNet, tratando de evitar el overfit. Agregamos una capa Dense extra con 64 neuronas y una función de activación ReLU. También agregamos la función de Dropout, que es un hiperparámetro que desactiva el 30% de las neuronas durante cada paso hacia adelante. Ajustar este valor es una parte importante del proceso de ajuste fino de una red neuronal para obtener un buen rendimiento y evitar el sobreajuste.

Después de correr el modelo, vemos que la precisión aumenta sin necesidad de tantos epochs.

```
Epoch 19/20
17/17 [==============================] - 370s 22s/step - loss: 0.3693 - accuracy: 0.8580
Epoch 20/20
17/17 [==============================] - 373s 22s/step - loss: 0.3717 - accuracy: 0.8752
```



Las gráficas de desempeño muestran que la varianza se reduce, y la pérdida converge en un valor más bajo.