INTEGRANTES DEL EQUIPO:

Anatanael Jesús Miranda Faustino A01769232

PROFESOR
Benjamín Valdés Aguirre

Módulo 2 Uso de framework

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. Para la implementación que carece de framework, se notó la complejidad que se presenta cuando no se cuenta con la ayuda de ninguna biblioteca extra. 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 incursionar en diferentes inicializaciones, algoritmos de optimización como Adam y funciones de activación como Relu. Este último es el modelo que escogemos 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 se tiene la carga de imágenes hacemos un split para dividir nuestro set de datos en 60% traint, 20% de test y 20% de validación. Esto lo hacemos ya que el set de datos no es muy grande, para cada clase se tiene 139 elementos, la cual no es una data tan grande para la clasificación que se quiere hacer, por lo que damos prioridad a la parte de entrenamiento. Además nos apoyamos del seed para generar nuestro números aleatorios con base a la semilla 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, que se utiliza para generar variantes de las imágenes originales al aplicar transformaciones aleatorias. Establecemos algunos atributos que nos ayudarán en el aprendizaje del modelo:

RandomFlip("horizontal"): Es una operación de volteo aleatorio horizontal a las imágenes con una probabilidad del 50%.

RandomRotation(0.1): Esto no sirve para aplicar 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 a las variaciones en la orientación de los objetos en las imágenes.

RandomZoom(0.1): Esto realiza un zoom aleatorio.

Todo esto se aplica ya que nuestro set de datos no es muy grande por ello 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 nuestro modelo, es importante señalar que creamos una matriz multidimensional (tensor) con forma (224, 224, 3). Esto se debe a las dimensiones de los píxeles y las tres canales que proporciona el formato RGB. En cuanto a las capas de nuestro modelo de redes:

Realizamos 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.

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

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

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

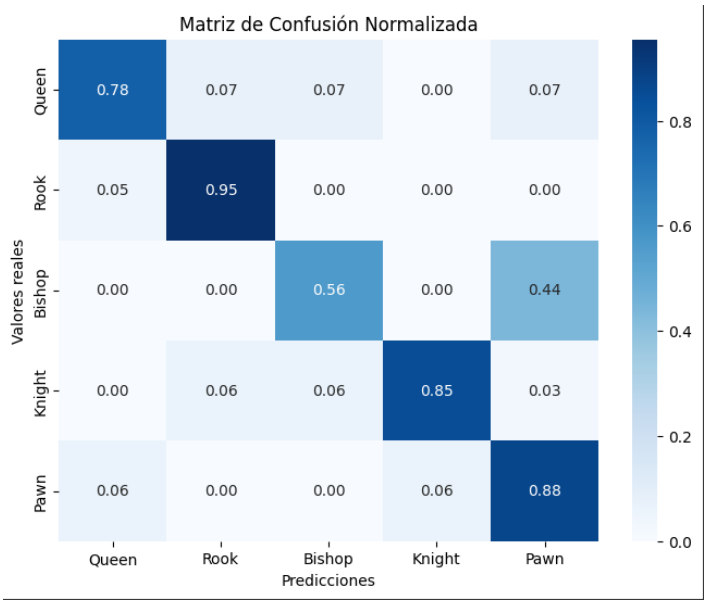
- Dense con 128 neuronas y una función de activación 'relu'. Las capas densas se emplean para realizar cálculos y aprender representaciones más complejas de las características de entrada.

- Dense con 5 neuronas y función de activación softmax. Esta capa consta de 5 neuronas, ya que representa la salida de nuestro modelo. La activación softmax se emplea comúnmente en problemas de clasificación multiclase para calcular las probabilidades de pertenencia a cada clase.

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

Sesgo

El sesgo de el modelo de clasificación se percibe medio, ya que tiene una tendencia incorrectamente la clase “Knight” esto nos dice que hay un rendimiento desigual en diferentes clases. Para corregir este sesgo, necesitamos considerar la distribución de clases en los datos y cómo el modelo se comporta en cada una de ellas.

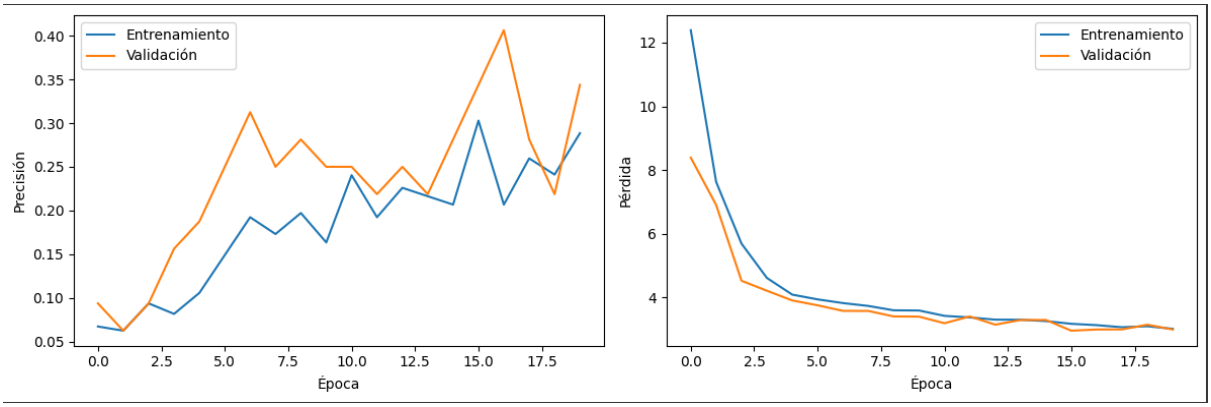


Aquí podemos ver que tiene tendencia es a equivocar pawn con bishop, también vemos que apesar de que los valores entre clases es cajo, también no se alcanza la referencia de bishop por lo que decimos que tiene un sesgo **medio bajo.**

varianza

```
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
```

Se presenta una varianza baja ya que tenemos una pérdida en el conjunto de entrenamiento disminuye de manera constante a medida que aumentan las épocas, Cuando se evalúa en el conjunto de validación (que contiene imágenes que el modelo no ha visto durante el entrenamiento), al final de todas las epochs se ve que el conjunto de entrenamiento alcanzó un accuracy es de 0.82 y el de validación es 0.78 demostrando que la brecha entre estas es baja



Diagnóstico y explicación el nivel de ajuste del modelo:

El modelo de reconocimiento de piezas de ajedrez ha demostrado que aun no esta completamente ajustado es decir esta "unferfitt" a los datos de entrenamiento. 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 perdida 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 epoch, 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. Podríamos 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

A pesar de que el modelo está considerablemente bien ajustado, todavía existe 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, nuestro set de datos se queda algo corto, además, no todas las fotografías son de calidad, se podría decir que varias de estas tienen ruido.

Los datos representan la base sobre la cual el 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 podríamos mejorar nuestro probando diferentes optimizadores de hiper parámetros o incluso modificando la cantidad de capas y neuronas por capa, para así crear modelos más avanzados para mejorar aún más el rendimiento.

Después de las mejoras

Se considera que lo mejor es aumentar nuestro set de datos sin embargo por falta de personal para recabar y etiquetar dichos datos optamos por hacer aún más robusto nuestro modelo.

```
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    path,
    validation_split=0.15,  # 15% para validación
    image_size=(224, 224),
    batch_size=32,
    subset="training",
    seed=42
)


test_data = tf.keras.preprocessing.image_dataset_from_directory(
    path,
    validation_split=0.15,  # 15% para prueba
    image_size=(224, 224),
    batch_size=32,
    subset="validation",
    seed=42
)
```

Cambiamos el tamaño para el se de pruebas para asignar más a train.

```
# Crear el modelo
model = tf.keras.Sequential([
```

```
    Rescaling(1./255, input_shape=(224, 224, 3)),
    data_augmentation,
    mobilenet,
    Flatten(),
    Dense(64, activation='relu'),
    Dense(128, activation='relu'),
    Dense(len(class_names), activation='softmax')
])
```

Agregamos una capa Dense extra con 64 neuronas y una función de activación relu. Después de correr el modelo vemos que el accuracy aumenta sin tantos epoch

```
Epoch 17/50
18/18 [==============================] - 136s 7s/step - loss: 0.7061 - accuracy: 0.7671 - val_loss: 1.0374 - val_accuracy: 0.7010
Epoch 18/50
18/18 [==============================] - 137s 8s/step - loss: 0.7040 - accuracy: 0.7726 - val_loss: 2.7350 - val_accuracy: 0.1959
Epoch 19/50
18/18 [==============================] - 126s 7s/step - loss: 0.6965 - accuracy: 0.7726 - val_loss: 1.8672 - val_accuracy: 0.2680
Epoch 20/50
18/18 [==============================] - 124s 7s/step - loss: 0.6145 - accuracy: 0.8159 - val_loss: 1.4180 - val_accuracy: 0.4330
Epoch 21/50
18/18 [==============================] - 123s 7s/step - loss: 0.7427 - accuracy: 0.7491 - val_loss: 1.4611 - val_accuracy: 0.5361
```