PlaneswalkerAI

Clasificador por color para Magic: The Gathering

Grant Nathaniel Keegan | A01700753

ABSTRACT

En este proyecto, se utilizarán algoritmos de redes neuronales convolucionales, para entrenar, y posteriormente clasificar por color, todas las cartas de *Magic: The Gathering* en formato standard. El proceso incluye varias etapas. De descarga y preprocesamiento de datos en imagen y metadatos, descargados de *Scryfall*, la base de datos más completa para cartas de MTG. De construcción y ejecución de dos algoritmos. El primero implementando una red neuronal convolucional base. La segunda, adaptando esta red neuronal a una ResNet50, para obtener un nivel de exactitud más alto. Y mejorar la ejecución de las gráficas de entrenamiento y validación en exactitud y pérdida. Al final, se exporta el modelo como un archivo .keras, para ser utilizado en un código que clasifica las cartas en diferentes carpetas. De esta forma, se utiliza un algoritmo de aprendizaje profundo para resolver un problema real, el cual es la organización de un alto número de cartas de Magic: The Gathering.

ÍNDICE

I. INTRODUCCIÓN

Magic: The Gathering es un juego de colección e intercambio de cartas (TCG) que ha incrementado de popularidad con el paso de los años. El juego ha creado una comunidad grande y competitiva, donde jugadores de Magic pueden coleccionar, intercambiar y vender cartas. El juego consiste de reglas y atributos que influencian la dirección de los juegos. Fue creado en 1994 por Richard Garfield, con 150 cartas únicas en el primer lanzamiento del juego. Hoy en día, existen más de 20,000 cartas diferentes en el juego.

El formato más competitivo de Magic: The Gathering es conocido como *Standard*, aquí, existe una serie de reglas más estrictas que aseguran que los jugadores tengan un campo de batalla equilibrado.[1] Y los torneos más reconocidos se juegan principalmente en el formato standard. Este formato tiene un sistema de rotación, donde cada tiempo definido, las cartas lanzadas en un set de hace 3 años ya no son legales para usar en torneo, para introducir nuevos sets y asegurarse de que el juego continué evolucionando.

En *Magic: The Gathering* existen 5 colores diferentes para las cartas, que determinan el estilo de juego para cada jugador. En un mazo competitivo, se sugiere elegir solamente de 1 a 3 colores para una estrategia efectiva. Un problema que tienen muchos coleccionistas es ordenar sus cartas por color, ya que es algo que puede tomar tiempo.

Para este proyecto, el objetivo es crear un modelo de aprendizaje profundo que pueda detectar el color de una carta con alta precisión, Esto permitirá dividir las cartas en una carpeta solamente por los atributos visuales de la imagen. Esto resuelve la problemática de división automática. Con solo la imagen de portada de la carta, se pueden extraer los *features* necesarios para ser detectados por un algoritmo de aprendizaje profundo.

El algoritmo de aprendizaje profundo puede detectar con alta confianza el color de las cartas en formato .jpg. En teoría, también se puede utilizar por un robot que detecte el color de las cartas físicas y las clasifique automáticamente observando la portada. Después tomando acciones para

acomodarlas de acuerdo a esta clasificación. El objetivo es que este proyecto establezca una base para aplicaciones de aprendizaje profundo para el mundo de Magic: The Gathering, y otros juegos de cartas, tales como *Pokémon* o *Yu-Gi-Oh*.

II. ANÁLISIS Y PROCESAMIENTO DE DATOS

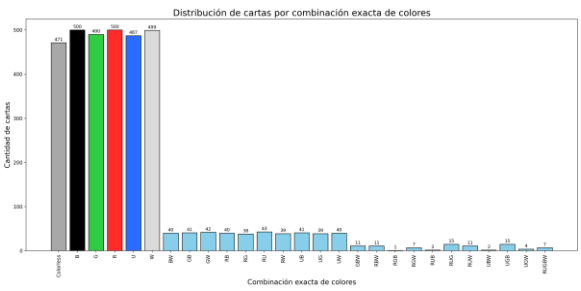
Para la base de datos utilizada en el entrenamiento de los modelos de este proyecto, se utilizará ***Scryfall***[2]. Es una base de datos abierta que contiene todas las cartas publicadas por Wizards of the Coast, la compañía detrás de Magic: The Gathering.

Existen un total de 5 diferentes colores de carta en Magic. En la base de datos están representadas con una letra y un símbolo visual en las cartas.

Color	Símbolo	Visualización en la Carta
Colorless	Colorless	 No contiene símbolo de las siguientes 5 categorías.
Red	R	
Blue	U	
Green	G	
White	W	
Black	B	

Los símbolos en las cartas son el rasgo principal del cual la CNN aprenderá a categorizar las cartas, ya que otros rasgos de información pueden variar dependiendo de la carta. Esto incluye el color de los bordes, o rasgos en el arte que se apegan al color de la carta. Por ejemplo, personajes y ubicaciones con temas similares.

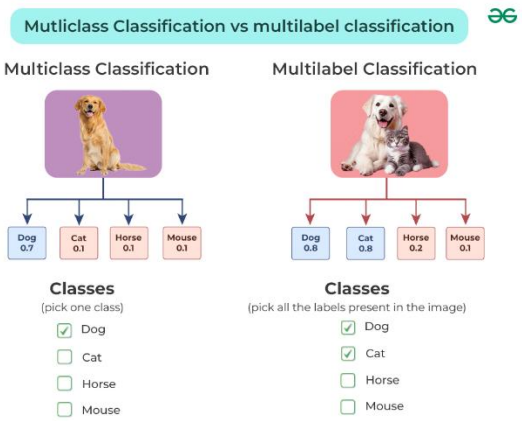
Se incorporo un análisis de la cantidad de cartas en cada categoría, contando las combinaciones de 2, 3, 4 y 5 colores. En la siguiente grafica se puede observar la distribución de estas cartas.



Como se puede observar, las cartas de un solo color son las mas numerosas, seguidas por las de 2 colores. Sorprendentemente, no hubo cartas de 4 colores, pero hubo 7 de 5, reservadas para los mazos más especiales. A continuación, voy a describir la naturaleza del problema, y el preprocesamiento de datos.

Un problema multi-etiqueta

Para poder resolver el problema de este proyecto, se necesita analizar exactamente qué tipo de datos se están tratando cuando se viene a las cartas de Magic: The Gathering. En un problema de aprendizaje de profundo con CNN, se pueden resolver problemas **multi-clase**, donde solamente se elige una predicción de clase en base a los datos de una imagen. O **multi-etiqueta**, donde se predice más de una clase valida a partir de la misma imagen. Como apoyo para resolver este problema con una CNN en keras, se consultó el video el usuario DigitalSreeni para clasificación multietiqueta.[3]



El problema con las cartas de Magic: The Gathering, es que, aunque la mayoría solamente tiene un color (o es incolora), hay una cantidad significativa que tienen 2, 3, 4 o hasta 5 colores válidos. Por ejemplo, la carta de abajo, la cual contiene los 5 colores. Por esto, si el algoritmo detecta más de un color en una carta, debe de clasificarla como una nueva clase que se considere como una mezcla de los 5 colores.

Para poder clasificar las cartas correctamente, se debe realizar un paso llamado **one-hot encoding**, el cual clasifica de forma binaria si la carta pertenece a cada clase en sus respectivas columnas. Para esto, se generan 6 diferentes columnas a parte del nombre. De aquí, se muestra la clasificación correcta de la carta, tanto para el modelo, como la clasificación final en los resultados.

Pipeline de preprocesamiento de datos

Descarga de csv | | descargar_datos_csv.py

El inicio del pipeline comienza descargando los datos crudos de las cartas en formato standard de Magic: The Gathering. Esto se hace a partir del primer código llamado *descargar_datos_csv.py*. Este código utiliza la API abierta de Scryfall para detectar cuales cartas son legales utilizando el query:

```
query = "game:paper legal:standard -is:promo -is:digital"
```

Después, se agrega cada entrada a la lista. Un paso importante es definir cuales columnas se van a guardar en el csv creado por el código, ya que hay demasiadas columnas que son irrelevantes. Se escogen 10-20, pero es importante incluir las columnas **"name"** que incluye el nombre de la carta, **"colors"** que incluye los colores de la carta, nuestro dato principal, e **"image_uri"** que incluye un link a la imagen en formato .jpg de la carta. Sera necesaria para descargar todas las cartas en .jpg y poder entrenar el modelo.

El archivo generado es *mtg_standard_cards.csv*

Este archivo será útil para dos razones: para descargar las imágenes, y crear el archivo de metadatos reales que utilizara el entrenamiento.

Descarga de imágenes | img_download.py

Es importante tener nuestra base de datos de las imágenes en formato .jpg para nuestro entrenamiento, y para probar el funcionamiento del modelo. Para esto, se crea el archivo *img_download.py*. Este archivo toma la columna del csv "image_uri", que contiene links a las imágenes en formato de imagen. Lee el texto de cada carta, elimina caracteres conflictivos, y las descarga en la carpeta de elección del usuario. El resultado son las 3,436 cartas en formato .jpg. que utilizaremos en futuros pasos.

Crear csv de metadatos | metadata_generator.py

Finalmente, se necesita una versión simplificada de nuestro archivo csv para poder etiquetar correctamente cada carta con sus valores reales. Se crea un nuevo archivo en vez de utilizar *mtg_standard_cards.csv* por dos importantes razones:

1. **Los datos de la columna "name" (los nombres de la carta) necesitan coincidir exactamente con el nombre que se muestra en los archivos de imagen.** Eliminando caracteres especiales como guiones y comas, que no se pueden guardar en las imágenes. Para esto se recorren todas las filas, y se eliminan los caracteres, creando el nuevo nombre que coincide con las imágenes .jpg.
2. Se necesita crear la columna de datos con el one-hot encoding para crear las etiquetas necesarias para el entrenamiento. El archivo toma los valores de la columna "colors" y los adapta para generar una lista con los valores reales de la carta.

El csv final de *card_metadata* se ve de la siguiente forma:

	A	B	C	D	E	F	G	H
1	name	colors	Colo	R	U	G	B	W
2	Aatchik Emerald Radian	[B, G]	0	0	0	1	1	0
3	Abandoned Campground	['Colorless']	1	0	0	0	0	0
4	Abhorrent Oculus	['U']	0	0	1	0	0	0
5	Abrade	['R']	0	1	0	0	0	0
6	Abraded Bluffs	['Colorless']	1	0	0	0	0	0
7	Absolute Virtue	[U, W]	0	0	1	0	0	1
8	Absolving Lammasu	['W']	0	0	0	0	0	1
9	Abuelo Ancestral Echo	[U, W]	0	0	1	0	0	1
10	Abyssal Gorestalker	['B']	0	0	0	0	1	0
11	Abyssal Harvester	['B']	0	0	0	0	1	0
12	Abzan Devotee	['B']	0	0	0	0	1	0
13	Abzan Monument	['Colorless']	1	0	0	0	0	0

Para este punto, se tienen ya los datos listos para el entrenamiento. Para recapitular, tenemos:

* La carpeta con las 3,436 imágenes de las cartas en .jpg

* El csv de metadatos. *card_metadata.csv*

Ahora, estos archivos se suben a una carpeta dedicada en Google Drive. Esto es para entrenarlos con Google Colab, y aprovechar los recursos de GPU de la plataforma para entrenar el modelo.

III. DESARROLLO DEL MODELO BASE CON UNA RED NEURONAL CONVOLUCIONAL

Para este proyecto, se utilizó una red neuronal convolucional (CNN) para la detección de los patrones de las imágenes. El objetivo fue construir una red neuronal convolucional capaz de extraer patrones visuales relevantes de las cartas de Magic: The Gathering sin depender de arquitecturas preentrenadas. Esto implicó diseñar una arquitectura equilibrada entre capacidad de aprendizaje, costo computacional, y riesgos de sobreajuste. Para esta primera red, se apoyó del tutorial de DigitalSreeni para clasificación multi-etiqueta con CNN[1], el tutorial de CNN de tensorflow[4], y las notas de clase sobre implementación de redes neuronales convolucionales[5].

Procesamiento de las imágenes para el modelo

Hay un aspecto muy importante que se debe de tener en cuenta para la ejecución del modelo. El preprocesamiento de las imágenes, principalmente definiendo su tamaño con x_size y y_size. Aquí es importante definir las imágenes como 224x224, ya que se utiliza como un estándar para el

entrenamiento de redes con arquitecturas de CNN modernas. También se convierten los pixeles de las imágenes a un arreglo de NumPy y se normalizan al dividirlas entre 255. Lo cual escala los valores a un rango de 0 a 1. Así se asegura que los datos de entrada estén normalizados e idénticos.

Para la división de entrenamiento, validación y pruebas, se hizo de (70, 15, 15)% respectivamente. Ya que es una división que ha generado los mejores resultados basado en experiencias previas en proyectos de aprendizaje de máquina.

Diseño del modelo base

Para este modelo, se aplicaron cuatro capas de convolución que incrementan en tamaño para mejorar el *feature extraction*, con kernels de 5x5. Y la función de activación ReLU para las 4, debido a su estabilidad y eficiencia para entrenar redes profundas.

Después de cada convolución, se agrega ***BatchNormalization*** para estabilizar las distribuciones internas de las activaciones y mejorar la generalización del modelo. Cada convolución también incluye un bloque de ***Maxpooling2D***, el cual conserva solamente las características más relevantes. Se agrega Dropout para evitar sobreajuste en el entrenamiento.

Después, se aplica la operación *flatten* para transformar los mapas de características de 2 dimensiones en un vector de 1 dimensión.

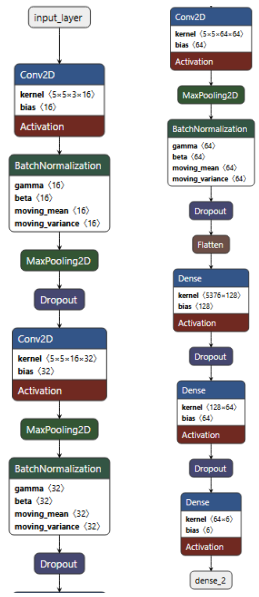
Al final, tres capas densas para clasificación. Combinando las características aprendidas y producir la predicción final. De estas capas densas, la final tiene un tamaño de 6 neuronas para nuestras 6 clases, y una ***activación sigmoide.*** Es importante recalcar que se utiliza una función sigmoide en vez de una *softmax*, que es más apta para un problema multi-clase, ya que nuestras cartas pueden pertenecer a más de una clase. También por esto se utilizó ***binary_crossentropy***, que evalúa de manera independiente la probabilidad de cada clase de las cartas.

El optimizador Adam se encarga de que la red aprenda. Es el algoritmo que ajusta los pesos de la red y permite que el modelo aprenda. Al final, también se agrega la métrica “accuracy” para evaluar el desempeño del modelo.

La estructura detrás del primer modelo es el siguiente:

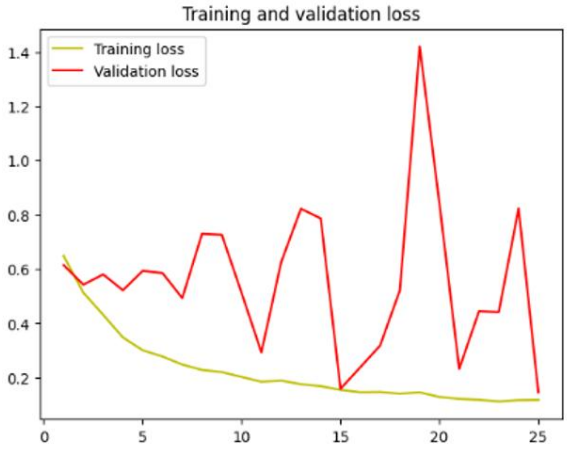
```
model = Sequential()

# Primera convolución 16x16.
model.add(Conv2D(filters=16, kernel_size=(5, 5), activation="relu", input_shape=(x_size, y_size,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Segunda convolución 32x32.
model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Tercera convolución 64x64.
model.add(Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Cuarta convolución 64x64.
model.add(Conv2D(filters=64, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Flatten.
model.add(Flatten())

# Capa densa final para clasificación.
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

# Capa de salida para las 6 etiquetas (Colorless, R, U, G, B, W)
model.add(Dense(6, activation='sigmoid'))

# Compilamos con optimizador Adam.
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

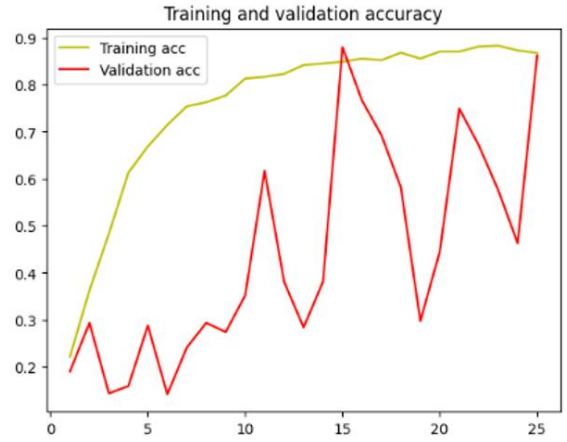
A continuación, se muestra una descripción gráfica del primer modelo con la app de Netron para visualizar modelos exportados:



IV. RESULTADOS DEL PRIMER MODELO

La ejecución del primer modelo mostro que la red convolucional funciona de una forma efectiva con el set de entrenamiento. Sin embargo, mostró inestabilidad en la validación del entrenamiento. Este fue el problema principal al que me enfrente durante el desarrollo del modelo de CNN. A continuación, se muestran las gráficas de pérdida y exactitud para los 25 epochs del entrenamiento:





Como se puede observar, la gráficas de el set de entrenamiento se estabilizan de una forma satisfactoria. Pero cuando se llega al set de validación, esta brinca demasiado, probando que no hay una estabilidad. Esto se debe a varias razones. Principalmente que la arquitectura es demasiado simple para el problema complejo de la clasificación de las cartas. También con poca profundidad, hay poca capacidad de generalizació, lo que sugiere utilizar un modelo con más capas. Como ResNet50 o EfficientNet, algo que nos ayudará en la siguiente fase del proyecto.

```
Epoch 21/25
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 88ms/step - accuracy: 0.8630 - loss: 0.1292 - val_accuracy: 0.7490 - val_loss: 0.2319
Epoch 22/25
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 88ms/step - accuracy: 0.8814 - loss: 0.1314 - val_accuracy: 0.6712 - val_loss: 0.4444
Epoch 23/25
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.8957 - loss: 0.1089 - val_accuracy: 0.5759 - val_loss: 0.4412
Epoch 24/25
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.8714 - loss: 0.1136 - val_accuracy: 0.4630 - val_loss: 0.8229
Epoch 25/25
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.8658 - loss: 0.1174 - val_accuracy: 0.8619 - val_loss: 0.1453
```

En los epochs finales del modelo, se observa como la exactitud cambia demasiado de 40% a 86%, dandonos un desempeño muy inestable, lo cual sugiere que otra arquitectura deberá ser necesaria para resolver este problema.

Los resultados de este entrenamiento, nos dan las siguiente métricas cuando se prueban en los sets de entrenamiento, validación y prueba:

```
Train:
R2: 0.8482921719551086
MSE: 0.023362284526228905
Bias: 0.006310885564250547
Varianza: 0.02332246692901153

Validation:
R2: 0.7687687277793884
MSE: 0.035654760897159576
Bias: 0.007037037237076242
Varianza: 0.035605251114556216

Test:
R2: 0.8443064093589783
MSE: 0.023497842252254486
Bias: 0.0068862254695935985
Varianza: 0.023450425515292538
```

Set de entrenamiento: Estos valores demuestran que el modelo aprende razonablemente bien sobre el set de entrenamiento, ya que un R^2 cercano a 0.85 indica que el modelo captura una gran parte de la variabilidad entre los datos. La MSE, bias y varianza bajas indican que el modelo es estable.

Set de validación: Aquí se puede ver que la R^2 es más baja que en el set de entrenamiento, lo cual confirma que el modelo no generaliza tan bien con este set de datos. La MSE, bias y varianza son más altos, lo que indica que el modelo está sobreajustando.

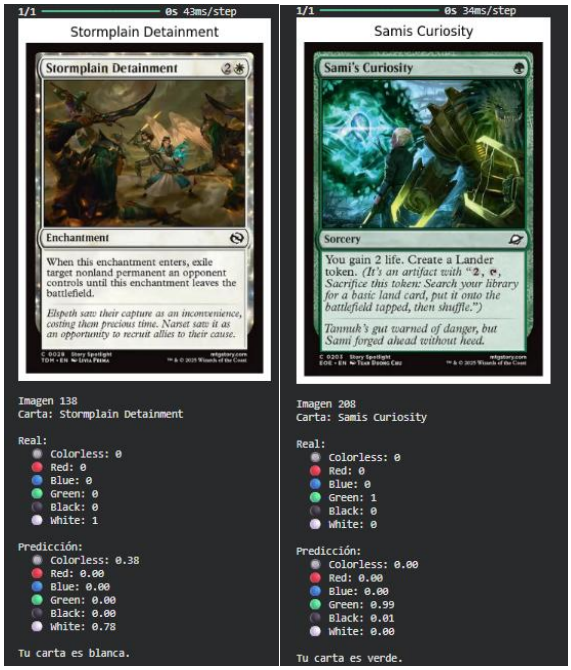
Set de prueba: Aquí los valores son similares a los del set de entrenamiento, lo que indica que, a pesar del desempeño más pobre con la validación, a la hora de entrenar los datos con el set de prueba, se obtienen resultados similares a los del set de entrenamiento.

Prueba con set de entrenamiento de cartas

Para probar el funcionamiento de el modelo dentro del archivo .ipynb, se implementó un código que genera 5 cartas aleatorias del dataset de prueba, y las despliega junto con la predicción del color de la carta y su valor real. Para que el modelo pueda hacer una predicción, **se estableció un grado de confianza de 60%** para que la detección de la carta establezca un resultado de cada color.

Esto permite resolver el problema multi-etiqueta de los resultados de las cartas, ya que si 2 o más colores se muestran con un grado de confianza mayor a 0.6, el algoritmo establece que tiene más de un color.

El resultado se puede demostrar de la siguiente forma:



En este resultado, se pueden observar dos cartas aleatorias del set de entrenamiento, la primera muestra un grado de confianza de 0.78 para el color blanco, y 0.38 para colorless. Por lo tanto, la establece como "blanca". La segunda muestra un grado de confianza de 0.99 para verde, su etiqueta real. Por lo tanto, la clasifica como verde.

Esta clasificación nos permite resolver una variedad de problemas, el principal es diseñando un algoritmo que clasifique las imágenes por color solamente tomando en cuenta sus rasgos visuales, algo que se describe en la parte final de este proyecto.

V. DESARROLLO DEL SEGUNDO MODELO (RESNET50)

Para la segunda fase del proyecto, se implementó una arquitectura más avanzada que el modelo base: **ResNet50**, un modelo residual ampliamente utilizado en visión computacional con 50 capas.[6,7] El objetivo fue aprovechar la capacidad de *transfer learning* para utilizar patrones previamente aprendidos por la red entrenada en la base de datos ImageNet, y adaptarlo para nuestro dataset con las cartas de Magic: The Gathering, con la intensión de mejorar el rendimiento.

Preprocesamiento de datos

Igual que en el modelo base, se utilizó un tamaño de 224x224 para el tamaño de las imágenes, se convierten de la misma manera en arreglos con NumPy y se normalizan, escalando los valores de 0 a 1. Así los datos de entrada son idénticos para cada carta de entrada.

Para la división de datos, de la misma forma se establece en un esquema de 70-15-15% para entrenamiento, prueba y validación respectivamente. Ya que establece mejores resultados y el tamaño de datos es suficiente para que aprenda de la fase de validación.

Construcción del modelo

Para la construcción del modelo, se utiliza el modelo base de ResNet50[6,7], utilizando los pesos de "imagenet", pero con la diferencia de que las capas densas originales son congeladas, ya que no tienen uso al ser entrenadas sobre un set diferentes de datos.

```python
# Tamaño de las imágenes
x_size = 224
y_size = 224
input_shape = (x_size, y_size, 3)

# Regularización L2
weight_decay = 1e-4

# ResNet50 como base.
inputs = Input(shape=input_shape)

base_model = ResNet50(
    include_top=False,
    weights='imagenet',
    input_tensor=inputs
)

# Congelamos la base para la primera fase de entrenamiento
base_model.trainable = False

# Capa de clasificación colores MTG.
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Capa densa final con L2
x = Dense(
    128,
    activation='relu',
    kernel_regularizer=l2(weight_decay)
)(x)

x = BatchNormalization()(x)
x = Dropout(0.40)(x)

# Capa de salida para 6 colores (multietiqueta)
outputs = Dense(
    6,
    activation='sigmoid',
    kernel_regularizer=l2(weight_decay)
)(x)

# Creamos el modelo final
model = Model(inputs=inputs, outputs=outputs)

# Compilación
model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Resumen del modelo
model.summary()
```

En un problema donde se tiene un dataset diferente, se recomienda congelar las capas cercanas al inicio de la arquitectura[8] utilizando la línea **include_top = False**, Al eliminar esta parte de ResNet50, conservamos únicamente su bloque convolucional profundo, especializado en extraer patrones visuales.

El modelo base se establece como no entrenable con **base_modelo.trainable = False**, lo que permite utilizar los patrones aprendidos sin modificarlos. Congelar las capas iniciales es recomendado, ya que representan características generales como bordes y colores que son reutilizables para la mayoría de entrenamientos visuales.

Después, se establece la capa de clasificación específica para las cartas, con *GlobalAveragePooling* para reducir la dimensionalidad del mapa de características y conservar la información más relevante de los filtros de ResNet50.

Se crea en la cabeza una capa densa de clasificación personalizada para nuestro entrenamiento, para resolver el problema específico de el modelo clasificando las cartas según sus 6 clases. Similar al primer modelo, se establece la función de activación *sigmoide* para el problema multi-etiqueta a diferencia de *softmax* para multi-clase.

En esta fase del modelo, se utiliza también una regularización L2, estableciendo el weight_decay a 1e-4 para mejorar la generalización del modelo. Al final se compila el modelo utilizando el optimizador Adam con un learning_rate ajustado a 0.005.

Justificación de otros cambios en la segunda red

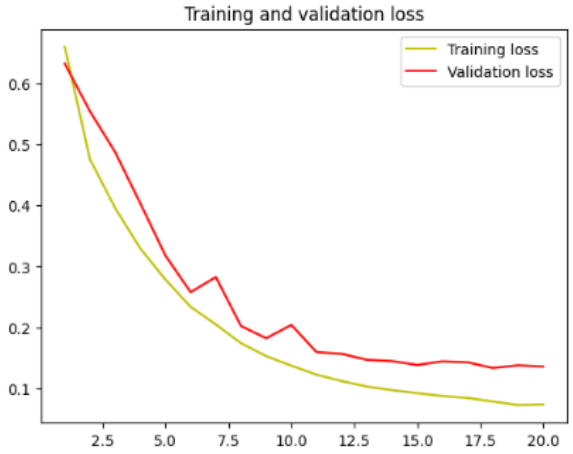
Se agregó **una técnica de regularización L2** para la capa densa final de el entrenamiento. La razón por la que se implementó es para penalizar pesos demasiado grandes y mejorar el desempeño del modelo. Ya que, sin regularización, la capa densa podría sobreajustarse demasiado. Esta es la razón también por la que al correr el modelo, se establecieron menos epochs, reduciendo el tamaño de 25 a 20.

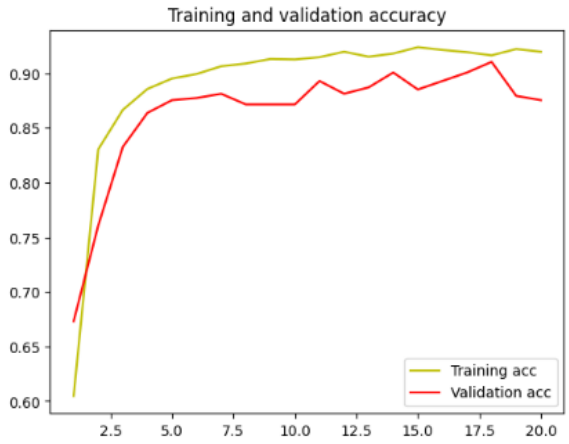
También se ajustó manualmente el learning rate del optimizador Adam en la compilación a (0.005), esta decisión se tomó por que ResNet50 ya contiene pesos preentrenados, y con un learning rate más bajo, estos pesos pueden tardar más tiempo en entrenarse.

VI. RESULTADOS DEL SEGUNDO MODELO (RESNET50)

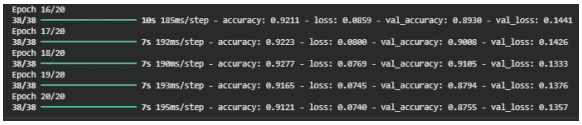
El entrenamiento del segundo modelo se establece con epochs más bajos que el modelo base. Esto se debe a la complejidad del modelo obteniendo mejores resultados a epochs más bajos. Así que se establece una cantidad menor para evitar sobreajuste. Se baja de 25 a 20.

A continuación, se evaluarán los resultados del segundo modelo de CNN que utiliza ResNet50:





Aquí se puede apreciar cómo las curvas de pérdida y entrenamiento son mucho más estables que en el modelo base, mostrando picos e irregularidades mínimos. Esto se debe a que ResNet50 funciona como un extractor de características mucho más robusto que una CNN construida desde cero. El uso de GlobalAveragePooling reduce los parámetros al final. Y la regularización L2 estabiliza aún más los pesos.



En los epochs finales del entrenamiento, se obtiene una exactitud de arriba de 87% constantemente, con una estabilidad mucho mayor que en el modelo base. Esto nos lleva a concluir que el modelo utilizando la versión modificada de ResNet50 es una mejor opción para el problema que se intenta resolver.

Para probar el funcionamiento de el nuevo modelo, se generaron de nuevo las cartas aleatorias con el set de prueba para observar que tan bien predice las clases de las cartas:

Se puede observar cómo en la carta izquierda, detecta exitosamente que es una carta de más de un color, clasificándola como roja y verde. La segunda, la clasifica exitosamente con una confianza de 0.96 como carta incolora. Esto sugiere que el modelo aprendió que las cartas que no contienen ninguno de los íconos de color, son clasificados de esa forma.

VII. APLICACIÓN DEL MODELO Y RESOLUCION DEL PROBLEMA

Código: clasificadorResNet50.py

Finalmente, una vez que esta el modelo entrenado, se exporta como .keras, está listo para poder ser utilizado en la resolución del problema: Poder clasificar las cartas, en base a sus rasgos visuales para diferentes carpetas correspondiente a su color o colores.

Para esta parte del proyecto, se implementó el código *clasificadorResNet50.py* para clasificar todas las cartas de la carpeta original *mtg_standard_cards*. Una vez aplicado: el código carga el modelo como .keras. Después establece las 6 clases para clasificarlas.

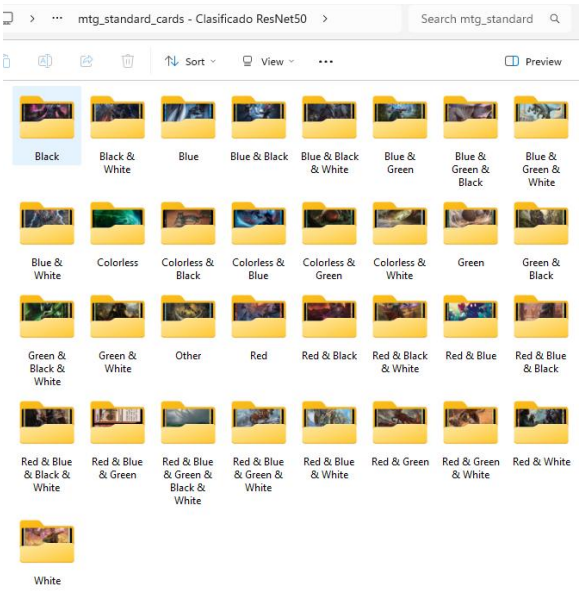
Un aspecto crítico para el funcionamiento del clasificador, es que debe tener exactamente el mismo preprocesamiento que se utilizó en el entrenamiento del modelo (imágenes redimensionadas a 224x224, conversión de colores, etc.)

Después clasifica las cartas con la línea:

```
pred = model.predict(img.reshape(1, x_size, y_size, 3))[0]
```

El cual realiza la predicción y las manda a sus carpetas correspondientes dentro del mismo directorio.

El modelo *MTGResnet50* fue el elegido para realizar la prueba final con el código, aplicándose para clasificar cada una de las cartas en su respectiva carpeta basada en su color, el resultado se muestra a continuación, una vez que se ejecuta iclasificadorResNet50.py y se escoge la carpeta de destino:



Al analizar las carpetas, se puede observar que el resultado no es 100% perfecto, ya que algunas fueron mal clasificadas, en especial cuando se trata de tarjetas que tienen más de un color, pero el resultado es lo suficientemente satisfactorio para concluir que el modelo de ResNet50 es mucho más efectivo y acertado que el modelo base.

VIII. CONCLUSIONES

Este proyecto fue una excelente forma de aplicar técnicas de aprendizaje profundo, para resolver una problemática de un hobby que mucha gente disfruta alrededor del mundo. El uso de una red neuronal convolucional es exitoso para la mayoría de los problemas que involucren imágenes. En este caso, construir una CNN desde cero ayuda a comprender cómo cada bloque utiliza sus funciones para aprender de un set de datos basado en imágenes.

El uso de una red establecida más avanzada como ResNet50, fue fundamental para ver cómo se puede mejorar un entrenamiento utilizando transfer learning. Ya que pudo extraer de una forma mejor los aspectos visuales de las cartas. La integración de los modelos generados para resolver un problema real ayuda a establecer una base para comprender cómo el aprendizaje profundo puede utilizarse de formas efectivas.

NOTAS Y REFERENCIAS

[1] Wizards of the Coast. (n.d.). Standard format. Magic: The Gathering. https://magic.wizards.com/en/formats/standard

[2] Scryfall. (n.d.). Scryfall: Magic card search. https://scryfall.com/

[3] DigitalSreeni. (2020, July 16). 142 – Multilabel classification using Keras [Video]. YouTube. https://www.youtube.com/watch?v=hraKTseOuJA

[4] TensorFlow. (s. f.). Convolutional neural network (CNN). TensorFlow. Recuperado el 28 de octubre de 2025, de https://www.tensorflow.org/tutorials/images/cnn

[5] Notas de clase. *Contenido ML Español.* Documentos y Códigos .ipynb. Benjamín Valdés Aguirre.

[6] GeeksforGeeks. (n.d.). Image classification using ResNet. https://www.geeksforgeeks.org/computer-vision/image-classification-using-resnet

[7] GeeksforGeeks. (n.d.). Residual networks (ResNet) in deep learning. https://www.geeksforgeeks.org/deep-learning/residual-networks-resnet-deep-learning

[8] GeeksforGeeks. (n.d.). Introduction to transfer learning. https://www.geeksforgeeks.org/machine-learning/ml-introduction-to-transfer-learning

REFERENCIAS ADICIONALES

[1] Wizards of the Coast. (n.d.). Introducción a Magic: The Gathering. Magic: The Gathering. https://magic.wizards.com/es/intro