
Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Ingeniería en Ciencia de Datos y Matemáticas


# Evidencia 1: Identificación de Naipes a través de un Algoritmo de Aprendizaje Profundo

## Diseño de Redes Neuronales y Aprendizaje Profundo


**Presentado por:**

*Ferreira Gudarrama Emiliano* A01654418

*Núñez López Daniel I.* A01654137

*Ugalde Jiménez Ana Sofía* A01702639


Monterrey, Nuevo León. Fecha, 5 de diciembre de 2022

# 1. Introducción

La identificación de naipes a partir de una imagen ha sido objeto de gran cantidad de investigaciones y propuestas en el mundo de la inteligencia artificial enfocada en juegos de cartas. Recientemente se han visto grandes avances en el desarrollo de IA para juegos multijugador de Póker texas. Diferentes tipos de redes neuronales se han utilizado para diferentes propósitos en dichos juegos, Por ejemplo, en enero del año 2022 se propuso una red neural de auto-codificador (autoencoder) para predecir el comportamiento de los demás jugadores (Wang et al., 2022). Asimismo, se pueden encontrar otras propuestas de mayor antigüedad, como los agentes de póker texas hold'em sin límite desarrollados en 2009 utilizando un método híbrido conocido como redes neuronales en evolución (Nicolai y Hilderman, 2009). Sin embargo, una parte importante para que estos agentes puedan interactuar de manera adecuada con el mundo real es la identificación de las cartas de la baraja a partir de una imagen.

# 2. Objetivo

El objetivo de este proyecto es diseñar un modelo de aprendizaje profundo capaz de reconocer las cartas de una baraja a partir de una imagen con una precisión mayor al 70

# 3. Marco Teórico

## 3.1. Dataset

En este proyecto se presenta una red neuronal capaz de identificar un naipe a partir de una imagen en forma de archivo. La base de datos fue tomada de Kaggle (Gerry, s.f.), del dataset Cards Image Dataset-Classification, así mismo, el autor cuenta con código de cómo cargar la base de datos adecuadamente, por lo que se siguieron sus pasos, sin embargo, para la creación del modelo y la experimentación, no se utilizó ningún código ya existente, ya que además de buscar la obtención de

buenos resultados, se busca un mayor aprendizaje de cómo utilizar redes neuronales para resolver problemas de clasificación de imágenes.

Este dataset cuenta con imágenes de 224 x 224 x 3 en formato .jpg, igualmente, las imágenes están recortadas para que solamente se vea una carta por fotografía y que la carta ocupe más del 50 % de los pixeles de la imagen. Hay 7624 imágenes para entrenamiento, 265 para el testeo y 265 para la validación.

## 3.2. Framework

El framework a utilizar es Tensorflow junto con la librería de Keras. Tensorflow es desarrollado por Google y se utiliza para crear y entrenar modelos de Machine Learning (ML) y Deep Learning (DL). Este está construido alrededor del concepto de un grafo computacional,en donde los nodos son representados por operaciones matemáticas y las esquinas representan tensores que fluyen entre las operaciones, esto permite paralelizar computaciones en una variedad de hardware para realizar el cálculo de los gradientes de funciones matemáticas complejas.

## 3.3. Modelo de ANN: CNN

El modelo a utilizar será convolucional, esto debido a que son buenos modelos específicamente diseñados para trabajar con estrucutras de datos en cuadrículas, como lo son las imágenes en 2D. Algunas de las ventajas de utilizar estos modelos son:

- Conectividad local: Cada neurona de la capa convolucional está conectada a una región pequeña de los datos de entrada, esto se traduce en que el modelo aprenda ciertas características de dicha región (texturas, esquinas, figuras) que son relevantes para su clasificación.
- Pesos compartidos: Las neuronas son capaces de compartir sus conocimientos con otras, eso reduce el número de parámetros porque los mismos pesos se pueden utilizar en regiones distintas lo que hace al modelo más eficiente de entrenar.

- Invariantes sobre la translación: Un modelo bien entrenado es capaz de clasificar imágenes incluso si el objeto de la imagen no está centrado o está parcialmente oculto. Esto se debe a que el modelo aprende las características relevantes para la tarea de clasificación y comúnmente son invariables respecto a la translación.
- Eficiencia: Se diseñan para ser eficientes tanto en términos de uso de memoria como costo computacional. Utilizan operaciones matriciales que se pueden implementar en una variedad de plataformas.

Dentro de las posibles capas a utilizar, se considerarán las siguientes:

1. MaxPooling2D: Esta capa se encarga de realizar una operación de muestreo descendente en los datos de entrada, dividiendo los datos en regiones que no se superpongan y conservando solamente el valor máximo para cada región. Esta capa tiene el efecto de reducir la dimensionalidad de los datos de entrada, reduciendo el costo computacional del modelo y mejorando su rendimiento. El parámetro "pool_size"define el tamaño de la agrupación, y se indica el tamaño de divisiones en regiones que no se superpongan. La característica principal de Max Pooling es que es utilizado comúnmente para extraer características específicas de los datos, como bordes o esquinas.
2. AveragePooling2D: Esta es muy similar a MaxPooling, sin embargo, toma el promedio de los valores de cada región. Tiene el mismo efecto de reducción de dimensionalidad. La característica principal de Average Pooling es que es utilizado comúnmente para extraer características generales de los datos, como texturas y formas.
3. Dropout: Las capas de dropout tienen como función regularizar los datos, en donde, al azar, se selecciona una fracción de los datos de entrada y se igualan a cero. Esto causa que se abandonen ciertas unidades en la capa y que se force a la red a aprender representaciones redundantes de los datos. Esto previene sobre-ajuste de datos y mejora el rendimiento del modelo.
4. Flatten: Este tipo de capas aplanan el tensor de entrada en un tensor unidimensional, y se realiza mediante el colapso las dimensiones espaciales (por ejemplo: el ancho y largo) de los datos de entrada a una sola dimisión. Generalmente, esto se realiza

5. Dense: Esta capa cuenta con las neuronas de la red, en donde cada neurona está conectada a las neuronas de la capa anterior. Esto significa que las capas Dense son capas completamente conectadas, donde no solamente se conectan con las neuronas de la misma capa, sino que con cualquier otra neurona de las capas vecinas. Estas capas son clave para el diseño de las redes neuronales y se utilizan en conjunto con otro tipo de capas (algunas de ellas son las mencionadas anteriormente).

## 3.4. Función de Pérdida

Como función de pérdida, utilizaremos Sparse Categorical Entropy, esta es una medición de la incertidumbre o aleatoriedad de un conjunto de datos categóricos. En un dataset, cada punto pertenece a uno de cada cierto número de categorías. Por ejemplo: supongamos un dataset de 2 categorías (0 y 1), un punto con valor de 0 pertenecería a la categoría 0. Se mide considerando la proporción de datos que pertenecen a cada categoría. En un dataset con alta entropía, los puntos se distribuyen a lo largo de todas las categorías, meintras que en un dataset con baja entropía, los puntos se concentran en pocas categorías.

Su fórmula es:

$$H = -\sum p(x) * log(p(x)) \tag{1}$$

donde $p(x)$ es la probabilidad de que un punto pertenezca a la categoría $x$, y donde la suma es sobre todas las categorías del dataset. Esta ecuación calcula la entropía de los datos mediante la suma de las probabilidades de cada categoría, multiplicada por el logaritmo de la probabilidad. El logaritmo se utiliza como peso de la contribución de cada categoría a la entropía global, esto para que las categorías con probabilidades bajas tengan mayor impacto en la entropía.

### 3.5. Optimizador

El optimizador a utilizar es .ªdam", ya que es una extensión del algoritmo de descenso por gradiente, el cual está diseñado para ser eficiente en encontrar valores óptimos de los parámetros de un modelo.

A diferencia del descenso por gradiente, con este optimizador los parámetros se actualizan utilizando una combinación del gradiente y un promedio móvil ponderado exponencialmente de los gradientes. Así, se combinan tanto las ventajas del descenso por gradiente como a adaptarse a la tasa de aprendizaje de cada parámetro, lo que brinda convergencia más rápida y mejor rendimiento.

### 3.6. Métricas de Desempeño

- Categorical Accuracy: Es una de las métricas más sencillas e intuitivas y es por esto que es de las más comunes para evaluar modelos de clasificación, sin embargo, esta métrica puede ser engañosa en algunos casos; datasets no balanceados, clases con importancia o significancia distinta, etc.
- Top 3 Categorical Accuracy: Es similar a la anterior, pero en lugar de considerar solamente la primera clase predicha para un punto, considera el top 3 de clases predichas. La predicción se considera como correcta si la clase verdadera está entre el top 3 de las clases predichas. A diferencia de la anterior, esta métrica se puede utilizar cuando los datos no están balanceados, además de ser útil cuando hay clases que tengan importancia o significancia distinta.

## 4. Experimentos

Para definir el número de epochs se realizó un monitoreo del training accuracy, en donde el delta mínimo entre los epochs fue de $1x10^{-4}$, considerando a los últimos 3 epochs. En caso de cumplir el criterio, corre un último epoch y al finalizar el mismo, se detiene el entrenamiento.

| Layer Type | Output Shape | Kernel Size | Bias Used | Activation Function | Notes |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3 ,3) | NO | ReLU | Input: (256, 256, 3) |
| conv | (252, 252, 32) | (3, 3) | YES | ReLU | |
| maxPool | (126, 126, 32) | (2, 2) | - | - | |
| dropout | (126, 126, 32) | - | - | - | rate=0.2 |
| conv | (124, 124, 64) | (3, 3) | YES | ReLU | |
| maxPool | (41, 41, 64) | (3, 3) | - | - | |
| dropout | (41, 41, 64) | - | - | - | rate=0.2 |
| conv | (39, 39, 128) | (3, 3) | YES | ReLU | |
| maxPool | (9, 9, 128) | (4, 4) | - | - | |
| dropout | (9, 9, 128) | - | - | - | rate=0.2 |
| flatten | (10368) | - | - | - | |
| dense | (1024) | - | YES | ReLU | |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 1: Estructura del Modelo Base

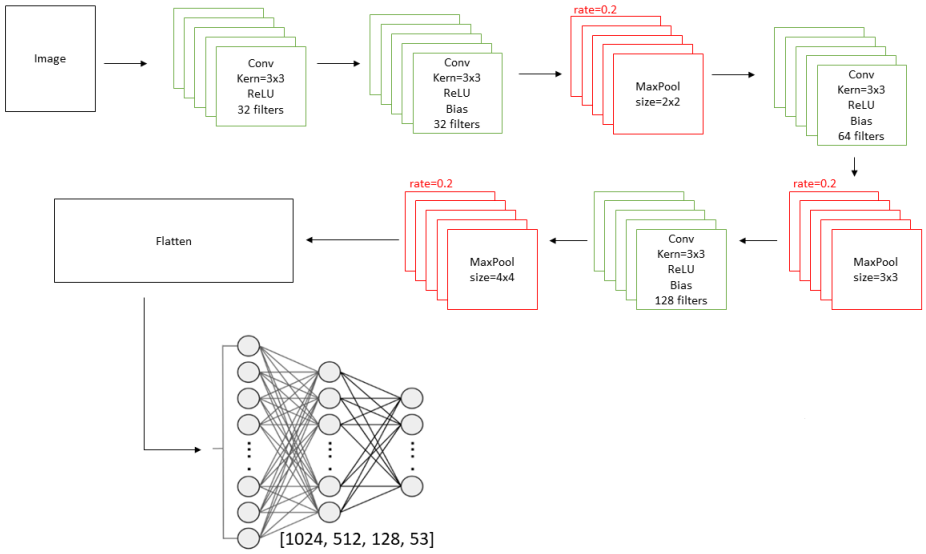


Figura 1: Estructura del Modelo Base

| Layer Type | Output Shape | Kernel Size | Bias Used | Activation Function | Notes |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3, 3) | NO | ReLU | Input(256, 256, 3) |
| maxPool | (127, 127, 32) | (2, 2) | - | - | |
| conv | (125, 125, 32) | (3, 3) | YES | ReLU | |
| maxPool | (62, 62, 32) | (2, 2) | - | - | |
| dropout | (62, 62, 32) | - | - | - | rate = 0.2 |
| conv | (60, 60, 64) | (3, 3) | YES | ReLU | |
| maxPool | (30, 30, 64) | (2, 2) | - | - | |
| dropout | (30, 30, 64) | - | - | - | rate = 0.2 |
| conv | (28, 28, 128) | (3, 3) | YES | ReLU | |
| maxPool | (14, 14, 128) | (2, 2) | - | - | |
| dropout | (14, 14, 128) | - | - | - | rate = 0.2 |
| conv | (12, 12, 512) | (3, 3) | YES | ReLU | |
| maxPool | (6, 6, 512) | (2, 2) | - | - | |
| dropout | (6, 6, 512) | - | - | - | rate = 0.2 |
| flatten | (18432) | - | - | - | |
| dense | (1024) | - | YES | ReLU | |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 2: Estructura del Modelo 1

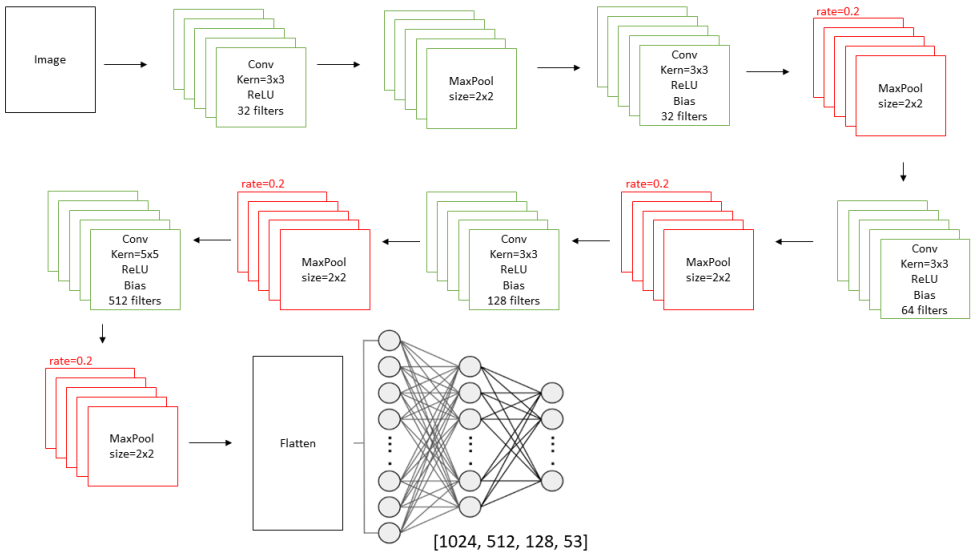


Figura 2: Estructura del Modelo 1

| Layer Type | Output Shape | Kernel Size | Bias Used | Activation Function | Notes |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3, 3) | NO | ReLU | Input(256, 256, 3) |
| maxPool | (127, 127, 32) | (2, 2) | - | - | |
| conv | (125, 125, 32) | (3, 3) | YES | ReLU | |
| maxPool | (62, 62, 32) | (2, 2) | - | - | |
| dropout | (62, 62, 32) | - | - | - | rate = 0.2 |
| conv | (60, 60, 64) | (3, 3) | YES | ReLU | |
| maxPool | (30, 30, 64) | (2, 2) | - | - | |
| dropout | (30, 30, 64) | - | - | - | rate = 0.2 |
| conv | (28, 28, 128) | (3, 3) | YES | ReLU | |
| maxPool | (14, 14, 128) | (2, 2) | - | - | |
| dropout | (14, 14, 128) | - | - | - | rate = 0.2 |
| conv | (12, 12, 512) | (3, 3) | YES | ReLU | |
| maxPool | (6, 6, 512) | (2, 2) | - | - | |
| dropout | (6, 6, 512) | - | - | - | rate = 0.2 |
| conv | (4, 4, 1024) | (3, 3) | YES | ReLU | |
| avgPool | (4, 4, 1024) | (1, 1) | - | | |
| conv | (2, 2, 512) | (3, 3) | YES | ReLU | |
| maxPool | (1, 1, 512) | (2, 2) | - | | |
| flatten | (512) | - | - | - | |
| dense | (1024) | - | YES | ReLU | |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 3: Estructura del Modelo 2

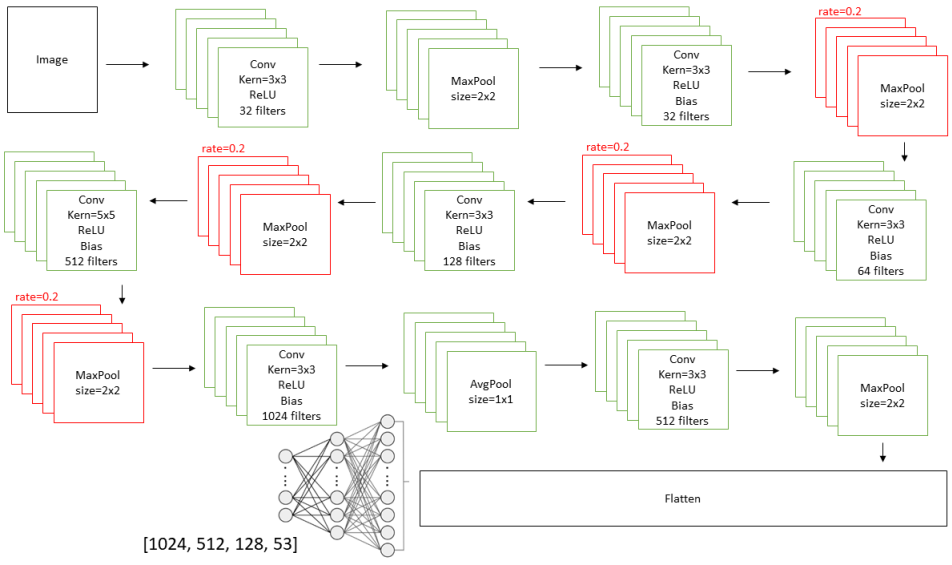


Figura 3: Estructura del Modelo 2

| Layer Type | Output Shape | Kernel Size | Bias Used | Activation Function | Notes |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3, 3) | NO | ReLU | Input(256, 256, 3) |
| maxPool | (127, 127, 32) | (2, 2) | - | - | |
| conv | (125, 125, 32) | (3, 3) | YES | ReLU | |
| maxPool | (62, 62, 32) | (2, 2) | - | - | |
| dropout | (62, 62, 32) | - | - | - | rate = 0.2 |
| flatten | (123008) | - | - | - | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 4: Estructura del Modelo 3

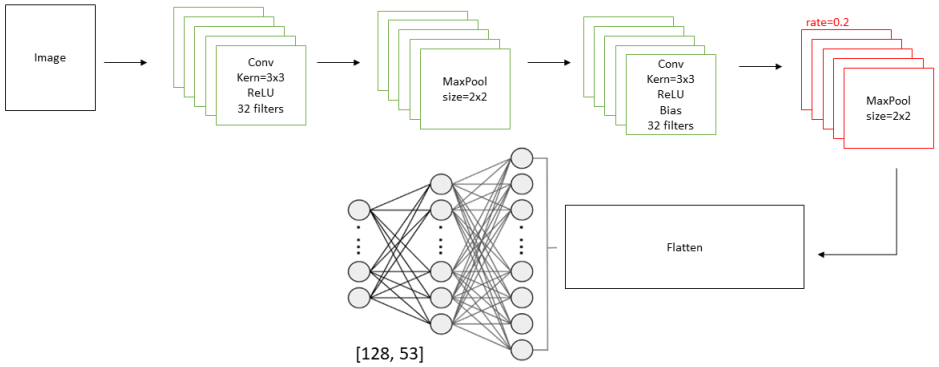


Figura 4: Estructura del Modelo 3

| Layer Type | Output Shape | Kernel Size | Bias Used | Activation Function | Notes |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3, 3) | NO | ReLU | Input(256, 256, 3) |
| conv | (252, 252, 32) | (3, 3) | YES | ReLU | |
| maxPool | (126, 126, 32) | (2, 2) | - | - | |
| conv | (124, 124, 64) | (3, 3) | YES | ReLU | |
| maxPool | (41, 41, 64) | (3, 3) | - | - | |
| conv | (39, 39, 128) | (3, 3) | YES | ReLU | |
| maxPool | (9, 9, 128) | (4, 4) | - | - | |
| flatten | (10368) | - | - | - | |
| dense | (1024) | - | YES | ReLU | |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 5: Estructura del Modelo 4

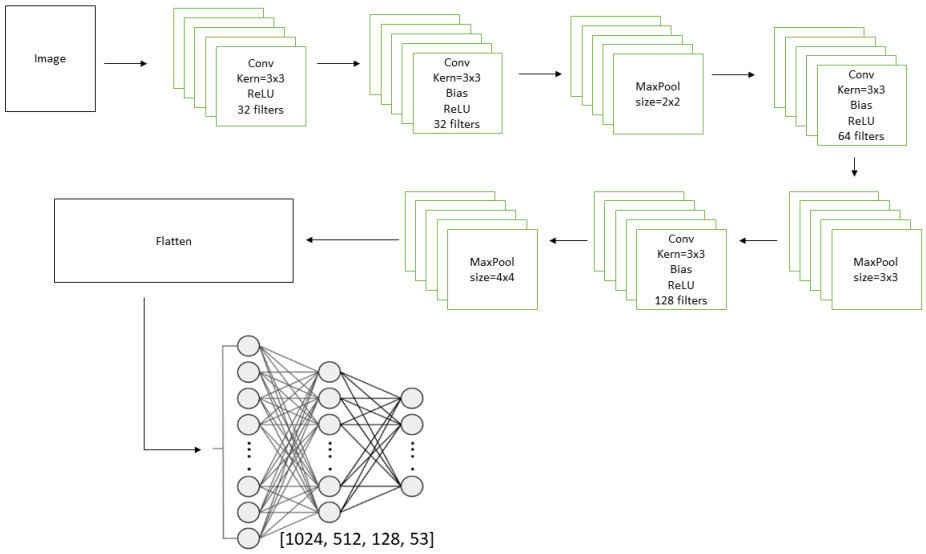


Figura 5: Estructura del Modelo 4

| **Layer Type** | **Output Shape** | **Kernel Size** | **Bias Used** | **Activation Function** | **Notes** |
|---|---|---|---|---|---|
| conv | (254, 254, 32) | (3, 3) | NO | ReLU | Input(256, 256, 3) |
| conv | (252, 252, 32) | (3, 3) | NO | ReLU | |
| maxPool | (126, 126, 32) | (2, 2) | - | - | |
| dropout | (126, 126, 32) | - | - | - | rate=0.2 |
| conv | (124, 124, 64) | (3, 3) | NO | ReLU | |
| maxPool | (41, 41, 64) | (3, 3) | - | - | |
| dropout | (41, 41, 64) | - | - | - | rate = 0.2 |
| conv | (39, 39, 128) | (3, 3) | NO | ReLU | |
| maxPool | (9, 9, 128) | (4, 4) | - | - | |
| dropout | (9, 9, 128) | - | - | - | |
| flatten | (10368) | - | - | - | rate = 0.2 |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 6: Estructura del Modelo 5

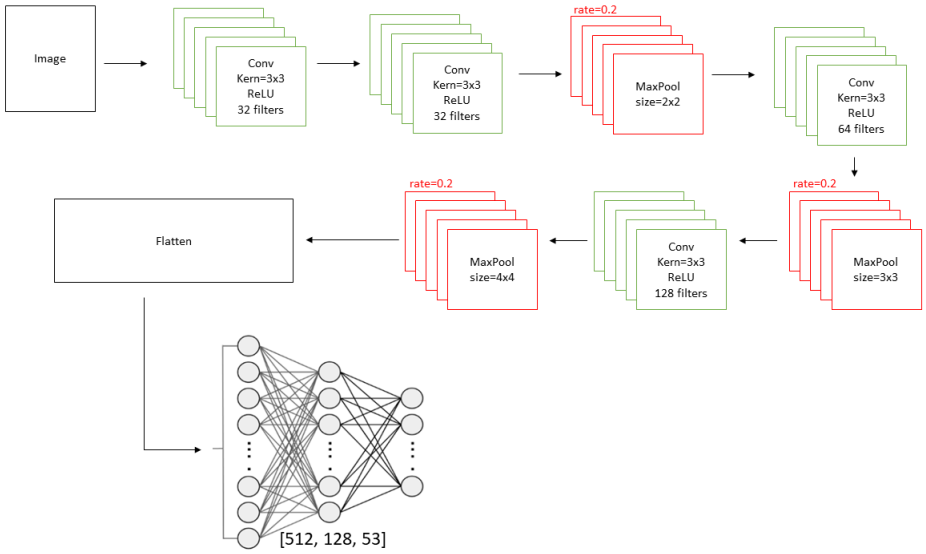


Figura 6: Estructura del Modelo 5

| **Layer Type** | **Output Shape** | **Kernel Size** | **Bias Used** | **Activation Function** | **Notes** |
|---|---|---|---|---|---|
| conv | (253, 253, 32) | (4, 4) | NO | ReLU | Input(256, 256, 3) |
| conv | (250, 250, 32) | (4, 4) | NO | ReLU | |
| maxPool | (125, 125, 32) | (2, 2) | - | - | |
| dropout | (125, 125, 32) | - | - | - | rate=0.2 |
| conv | (122, 122, 64) | (4, 4) | NO | ReLU | |
| maxPool | (40, 40, 64) | (3, 3) | - | - | |
| dropout | (40, 40, 64) | - | - | - | rate = 0.2 |
| conv | (37, 37, 128) | (4, 4) | NO | ReLU | |
| maxPool | (9, 9, 128) | (4, 4) | - | - | |
| dropout | (9, 9, 128) | - | - | - | rate = 0.2 |
| flatten | (10368) | - | - | - | |
| dense | (512) | - | YES | ReLU | |
| dense | (128) | - | YES | ReLU | |
| dense | (53) | - | YES | Softmax | |

Cuadro 7: Estructura del Modelo 6

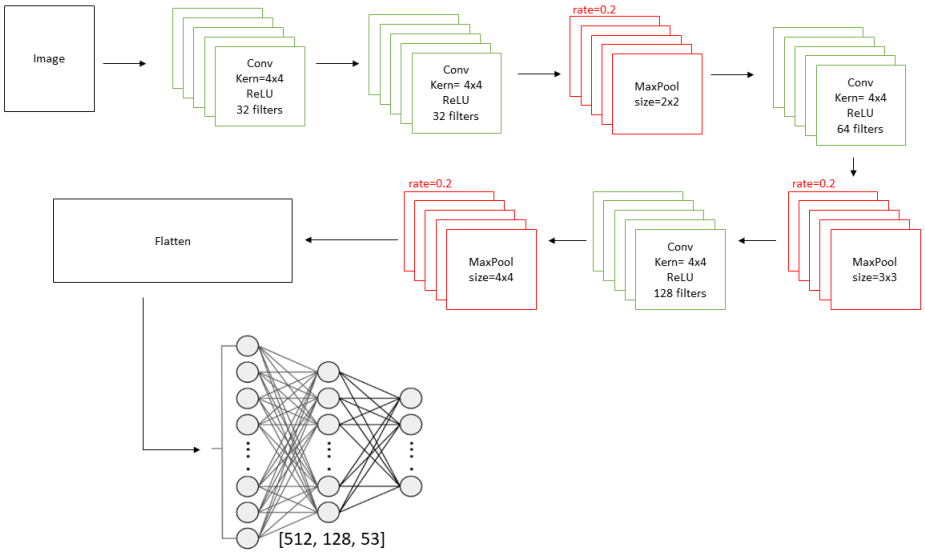


Figura 7: Estructura del Modelo 6

# 5. Resultados

| Modelo | Epochs | Training Loss | Training Accuracy | Training Top 3 Accuracy | Validation Loss | Validation Accuracy | Validation Top 3 Accuracy | Test Loss | Test Accuracy | Test Top 3 Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|
| Base | 19 | 0.1660 | 0.9545 | 0.9937 | 0.9924 | 0.7962 | 0.9019 | 1.2508 | 0.7736 | 0.8906 |
| 1 | 24 | 0.2116 | 0.9411 | 0.9879 | 1.1769 | 0.7962 | 0.8981 | 1.0390 | 0.8000 | 0.8717 |
| 2 | 43 | 0.3486 | 0.9052 | 0.9731 | 1.0386 | 0.8000 | 0.8642 | 1.2070 | 0.7849 | 0.8604 |
| 3 | 25 | 0.2434 | 0.9575 | 0.9873 | 7.7885 | 0.3811 | 0.6189 | 7.1823 | 0.4491 | 0.6151 |
| 4 | 18 | 0.1600 | 0.9643 | 0.9933 | 1.5499 | 0.7660 | 0.8679 | 1.8327 | 0.7509 | 0.8792 |
| 5 | 25 | 0.1601 | 0.9580 | 0.9946 | 1.4822 | 0.7698 | 0.8679 | 1.6315 | 0.7811 | 0.8642 |
| 6 | 22 | 0.2104 | 0.9473 | 0.9912 | 0.9212 | 0.8302 | 0.9132 | 1.2295 | 0.7962 | 0.9132 |

Cuadro 8: Resultados

# 6. Conclusiones

Con este proyecto pudimos observar y experimentar directamente con diferentes técnicas y diseños de redes neuronales. A pesar de que el enfoque fue en clasificación de imágenes fue necesario investigar sobre los diferentes tipos de modelos y aplicaciones que vimos en clase para decidir cual sería utilizado para este proyecto. A pesar de que hay muchos ejemplos de modelos similares, incluso utilizando el mismo dataset decidimos ignorar estos ejemplos e intentar experimentar a partir de

nuestras propias ideas y nociones. En algunos casos intentamos simplificar el modelo y notamos que no afectaba mucho el tiempo de entrenamiento, que era una de las principales razones por las cuales nos interesaría hacer este cambio, pero lo que si cambiaba bastante era el desempeño. El modelo 3 fue el peor según casi todas las métricas y este incluye únicamente 2 convoluciones, 2 pooling layers y 2 dense layers. La respuesta más probable es que este modelo simplemente no es suficientemente complicado para capturar todas las características que pueden distinguir a las cartas, especialmente considerando que el dataset incluye cartas que son difíciles de distinguir por un humano de un vistazo. Utilizando la métrica de validation_accuracy podemos concluir que el modelo mejor entrenado fue el modelo 6 con 83 % de precisión, además es el que muestra menor diferencia entre la precisión con los datos de entrenamiento y con los de prueba y validación. Este es de complejidad media en nuestro espectro, alternando capas de maxpooling con pools cada vez más grandes y capas de convolución con kernels de 4x4 píxeles. Si el tamaño de las imágenes fuera menor nos podría indicar que los demás modelos con kernels más pequeños tal vez estaban fallando en abstraer características más grandes de las imágenes. Además de utilizar kernels de convolución más grandes, ninguna de estas capas utilizaba bias. Finalmente esto contribuyó a un modelo con únicamente 5,563,445 de parámetros, todos entrenables. En tan sólo 22 épocas logró clasificar la carta correcta el 99.12 %, 91.32 % y 91.32 % de las ocasiones para los datos de entrenamiento, validación y prueba respectivamente.

# Referencias

Gerry. (s.f.). Cards Image Dataset-Classification. https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification

Nicolai, G., & Hilderman, R. J. (2009). No-Limit Texas Hold'em Poker agents created with evolutionary neural networks, 125-131. https://doi.org/10.1109/CIG.2009.5286485

Wang, S., Wang, H., Gao, Q., & Hao, L. (2022). Auto-encoder neural network based prediction of Texas poker opponent's behavior. *Entertainment Computing*, *40*, 100446. https://doi.org/ https://doi.org/10.1016/j.entcom.2021.100446

# A. Gráficas de Training Accuracy y Validation Accuracy durante el entrenamiento de los modelos

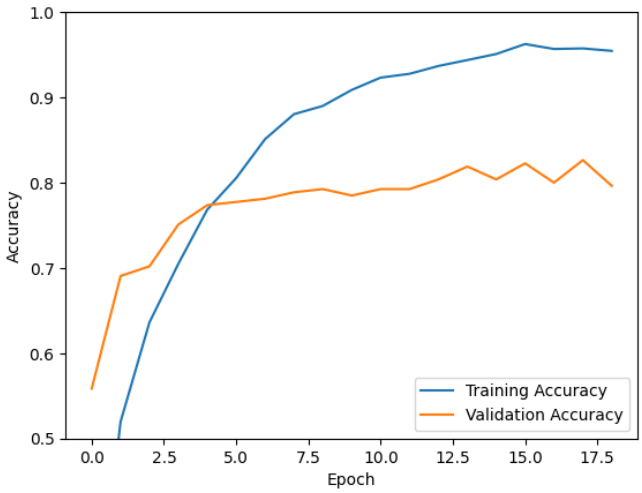


Figura 8: Modelo Base

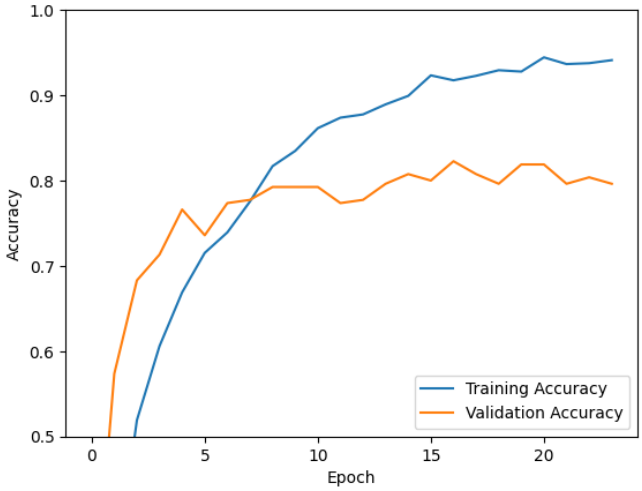


Figura 9: Modelo 1

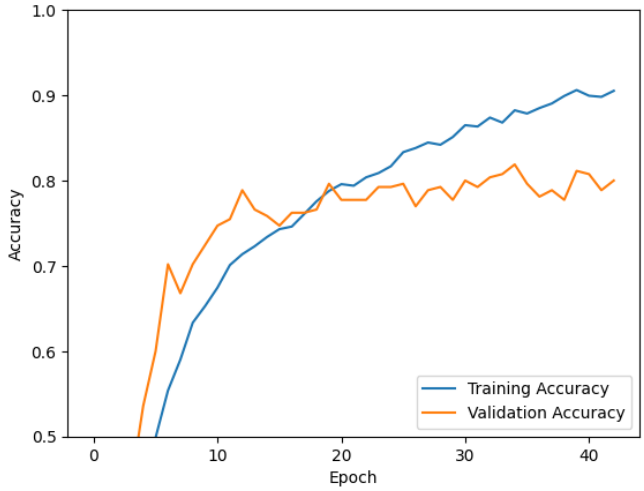


Figura 10: Modelo 2


Figura 11: Modelo 3

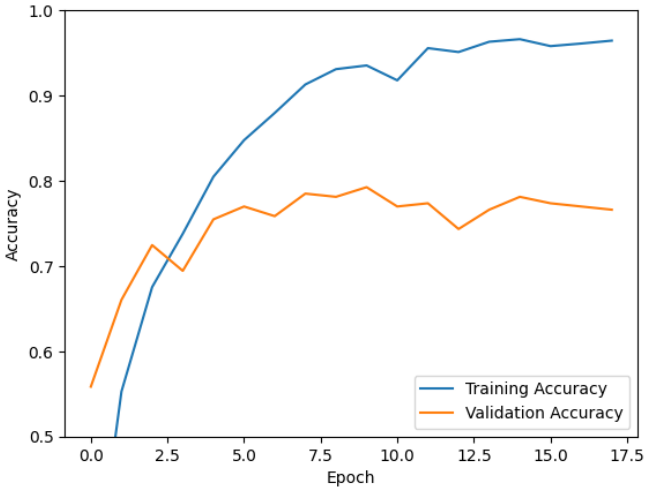


Figura 12: Modelo 4

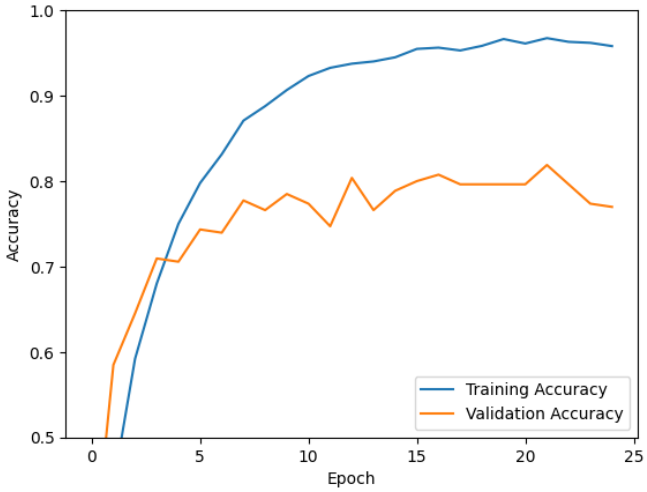


Figura 13: Modelo 5

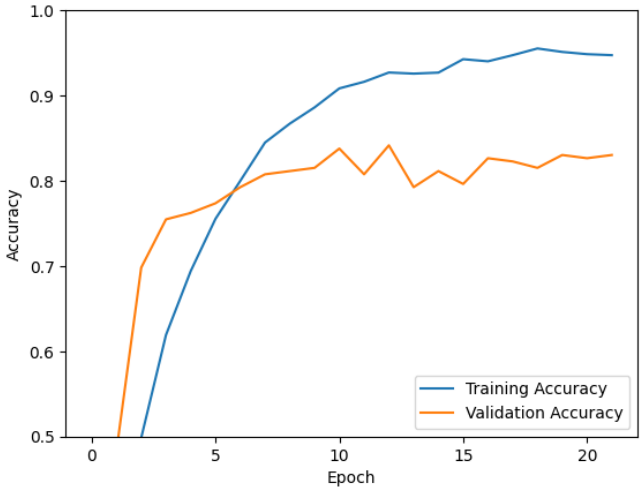


Figura 14: Modelo 6