# Descripción del problema:

En un mazo de cartas de poker hay 13 valores y 4 palos distintos. Además de los comodines. Se pretende, a partir de una imagen con una única carta, determinar a qué carta pertenece.

El dataset utilizado se encuentra en Kaggle

[Cards Image Dataset-Classification (kaggle.com)](https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification)

Salida esperada: clasificación de la carta de la imagen

# Conjunto de datos

El dataset está compuesto por imágenes RGB de 224 x 224 x 3 en formato jpg. Todas las imágenes del dataset ocupan más del 50% de los píxeles de la imagen, por lo cual esto va a ser una restricción para los inputs.

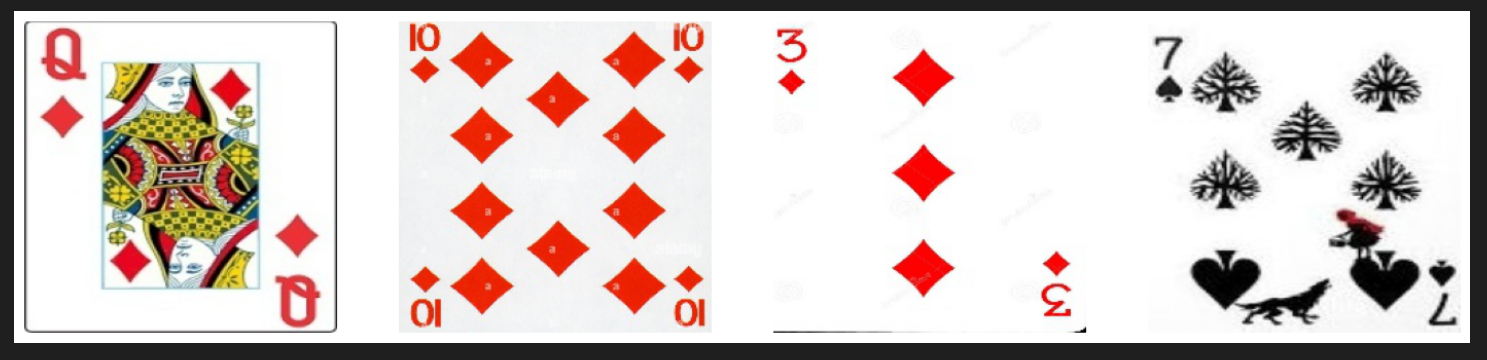
Posee 53 clases. 13 por palo más Joker

Resolución de las imágenes: 224x224

El conjunto de datos está dividido en:

train:7625 img, test: 265 img valid:265

Ejemplo de 4 imagenes



# Solución propuesta

Entrenar un clasificador de imágenes multiclase, que debiera poder generalizar para imágenes con diferentes orígenes.

El dataset está balanceado, con lo cual no hacen falta técnicas para comepensar desbalances.

La solución deberá devolver la clase de la carta de la imagen

Requerimiento: que haya una sola carta en la imagen, y la carta ocupe al menos el 50% de los píxeles.

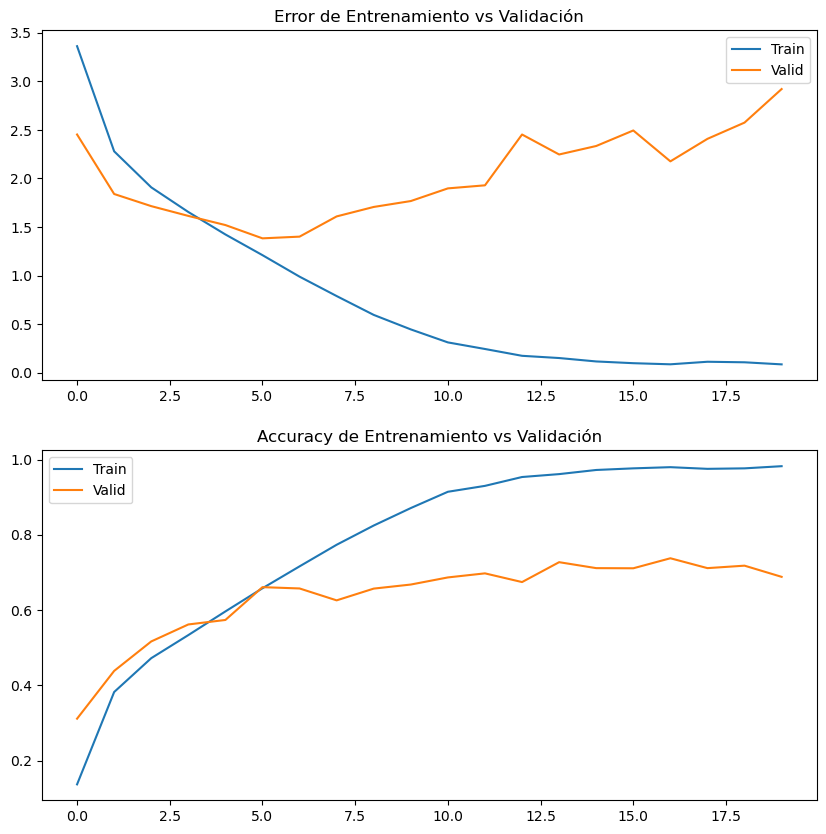
# Modelos probados

## Casero:

Inicialmente pruebo con un modelo casero muy sencillo para empezar a ver cómo responde.

Código: TPFinal\_Cards1.ipynb

Se trata de una red convolucional con cuatro capas de convolución + pooling y al final 2 capas fc. Con aproximadamente 13 M de parámetros entrenables.

Abajo observamos las métricas. 

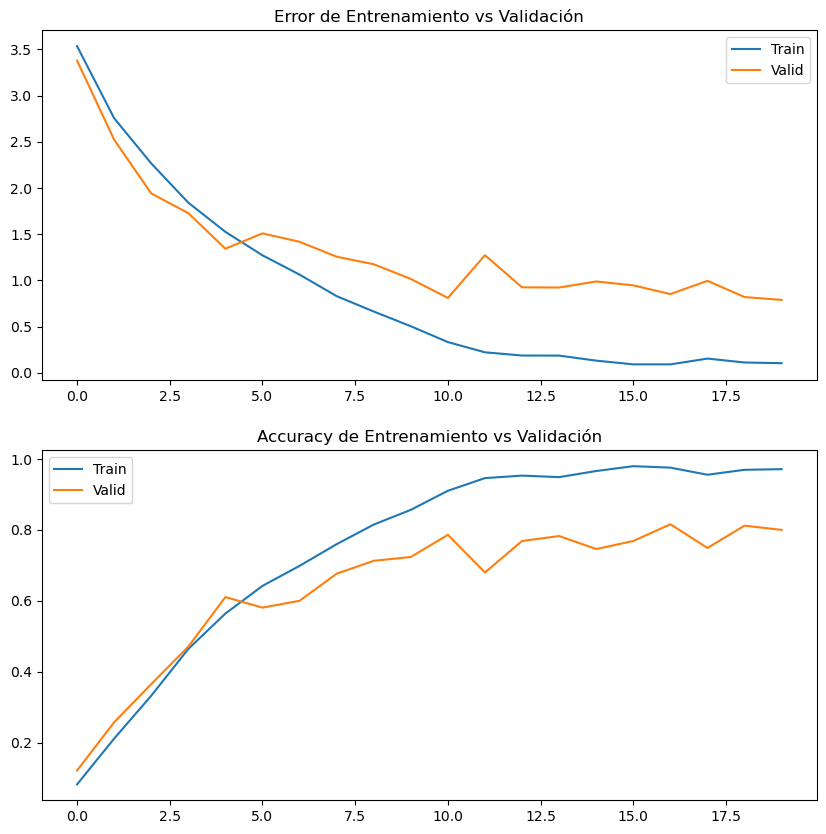
Se ve un claro sobreajuste. El modelo logra ajustar muy bien a los datos de entrenamiento, pero falla en generalizar.

## Resnet-50:

En este segundo modelo uso la arquitectura de la red Resnet-50 ya disponible en las librerías de torchvision, la cual tiene aprox 23 M de parámetros entrenables

Código: TPFinal\_Cards2.ipynb

Abajo las métricas:



Aquí se observa una leve mejora respecto del modelo casero, pero igualmente sobreajusta bastante.

## Resnet-50 con data augmentation:

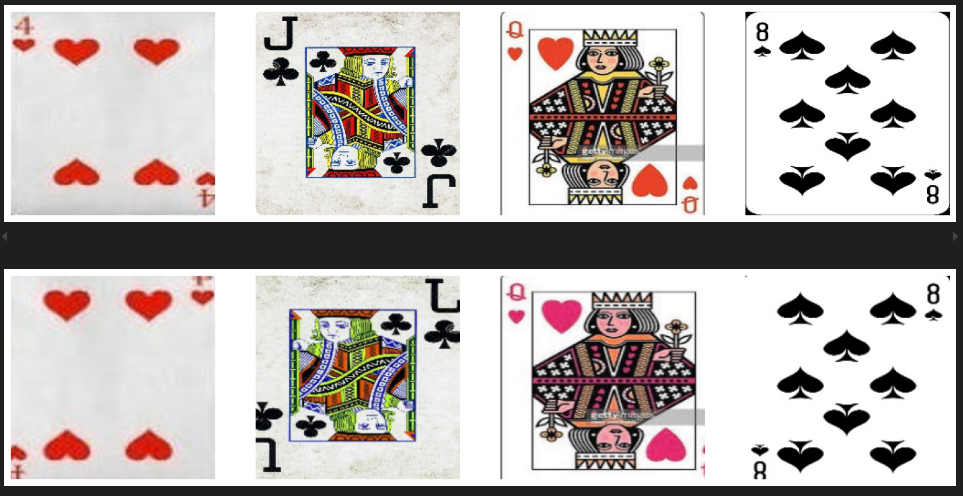
A la misma arquitectura que el modelo anterior en lugar de entrenar con los datos originales, se realizan transformaciones aleatorias a los inputs. De esta manera se pretende aimentar al modelo con datos más generalizados.

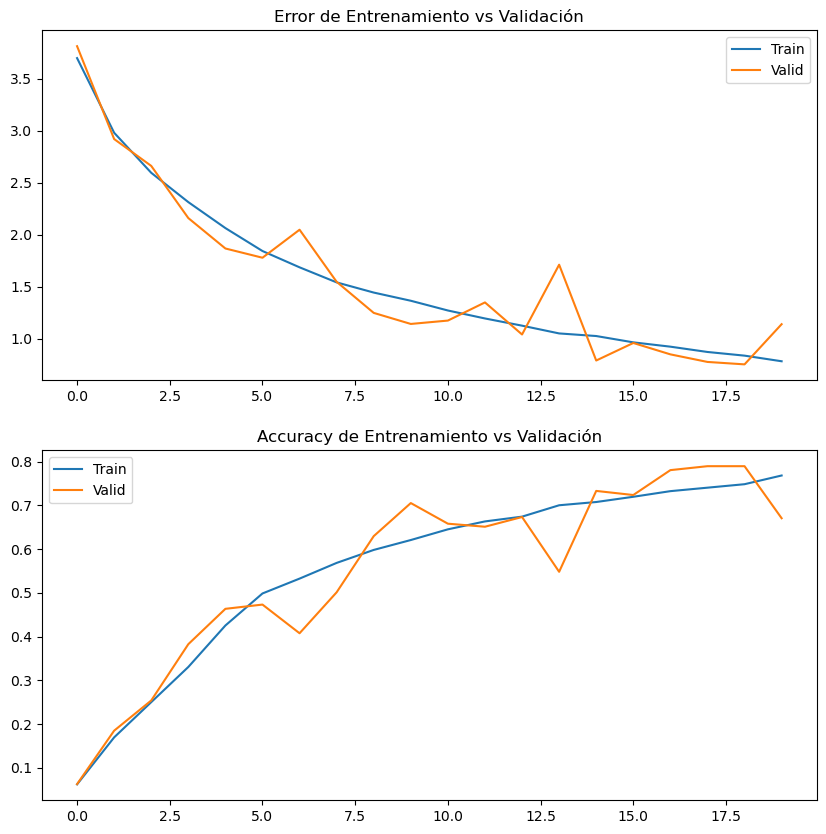
Código TPFinal\_Cards1.ipynb

Se realizan las siguientes transformaciones para aumentar:

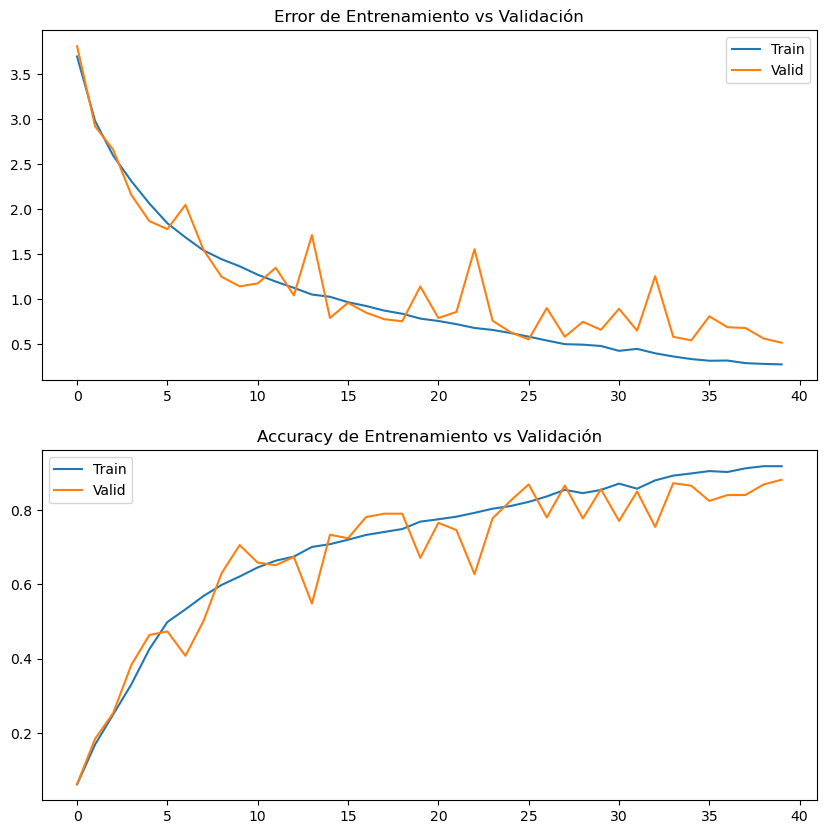
* RandomHorizontalFlip(0.5),
* RandomResizedCrop(size=(ANCHO\_IMAGENES, ALTO\_IMAGENES)
* ColorJitter(saturation=0.1, hue=0.1),
* RandomRotation(degrees=(0,180)),
* RandomPerspective(0.4),
* RandomResizedCrop(size=(ANCHO\_IMAGENES, ALTO\_IMAGENES),
* ColorJitter(saturation=0.3, hue=0.3)

Algunos ejemplos de las transformaciones:



Las métricas abajo:

Se observa que no hay sobreajuste, pero en 20 épocas el entrenamiento no ajusta tan bien como los otros modelos.

Hago una continuación con 20 épocas más y obtengo las siguientes métricas:

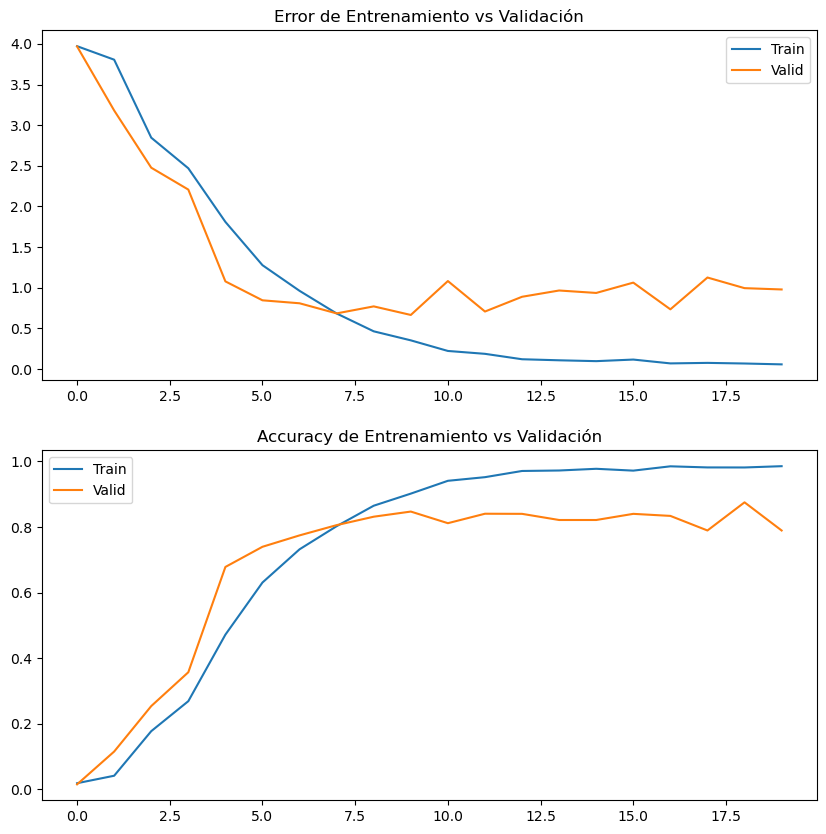
Acá se mejora mucho, llegando a casi el 90% de accuracy en validación, sin sobreajuste.

## VGG-19

En este modelo utilizo la arquitectura de VGG-19 disponible en torchvision con aproximadamente 140 M de parámetros

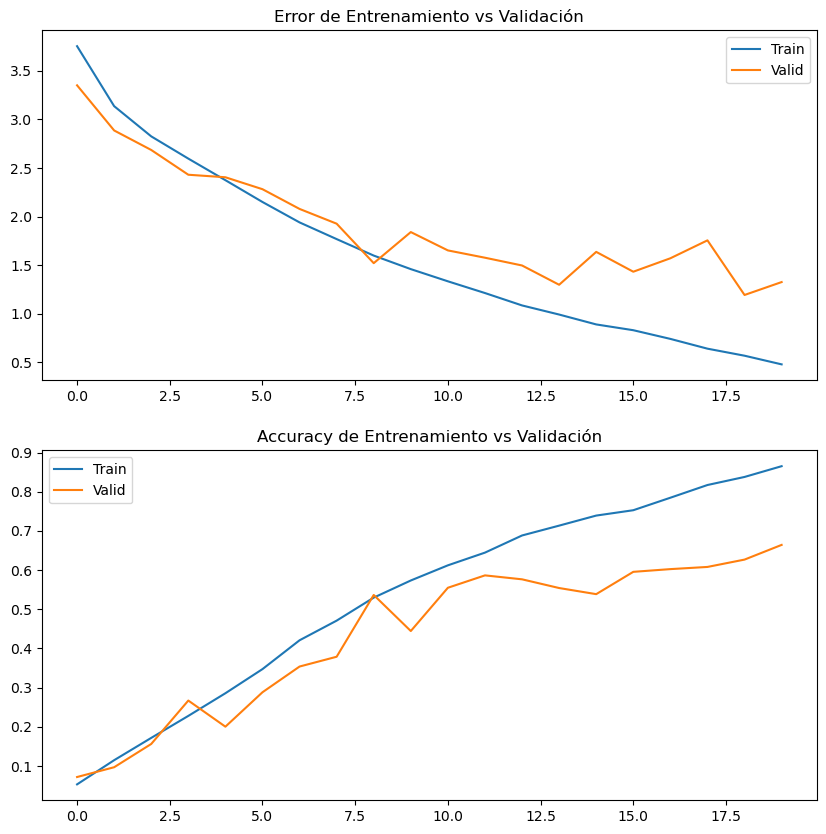
Código: TPFinal\_Cards\_3.ipynb

Métricas:



## Inception V3

En este modelo utilizo la arquitectura de Inception V3 disponible en torchvision con aproximadamente 25 M de parámetros



Nuevamente se observa Sobreajuste