Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Inteligencia Artificial Avanzada para la Ciencia de Datos

Deep Learning Portafolio

Alejandro Castro Reus A01731065

25 de noviembre del 2022

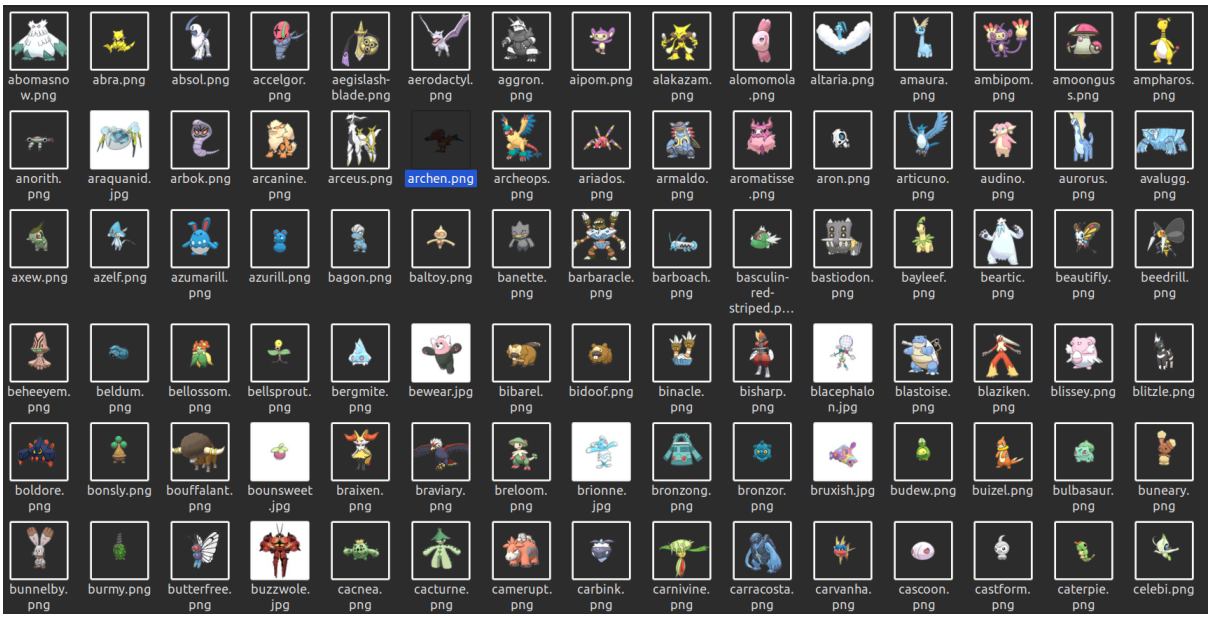
Problema a resolver

El problema que se quiere resolver con este modelo es la clasificación de imágenes de pokemones por su tipo 1. De esta forma, el modelo al recibir una imagen de un pokémon como entrada puede predecir a qué tipo pertenece.

Dataset

Se utilizó el Pokemon Image Dataset que se encuentra disponible en Kaggle (https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types) . El dataset cuenta con 809 imágenes de Pokemones y un csv con la información de los tipos que tiene cada una.

Las imágenes no están divididas por carpetas por lo que se tienen que separar con ayuda del CSV. Imagen de la carpeta de imágenes:



Tratamiento de datos y separación en train, validation y test

Lo primero que se realizó fue obtener los tipos que tenían más de 70 apariciones, esto ya que se consideró que un número menor no iba a tener los datos suficientes para entrenar apropiadamente el modelo.

Posteriormente se separaron los datos en 3 grupos: train, validation y test; teniendo 60%, 20% y 20% de los datos respectivamente. Esto da 221, 74 y 74 imágenes. Se

crearon 3 carpetas para los conjuntos de datos y cada una de ellas con 4 carpetas que representan los tipos de pokemones.

Primer modelo

Como primer acercamiento se definió una red compuesta por dos capas convolutivas con una capa de pooling entre las dos. Posteriormente una capa de flatten y por ultima dos capas densas. Se decidió esta configuración ya que como se está trabajando con imágenes lo más apropiada es usar capas convolutivas.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 238, 238, 16)      448

 max_pooling2d_9 (MaxPooling  (None, 79, 79, 16)       0
 2D)

 conv2d_18 (Conv2D)          (None, 77, 77, 32)        4640

 flatten_10 (Flatten)        (None, 189728)            0

 dense_21 (Dense)            (None, 8)                 1517832

 dense_22 (Dense)            (None, 4)                 36

=================================================================
Total params: 1,522,956
Trainable params: 1,522,956
Non-trainable params: 0
_________________________________________________________________
```

Se definieron las siguientes configuraciones en la compilación y en el fit generator:

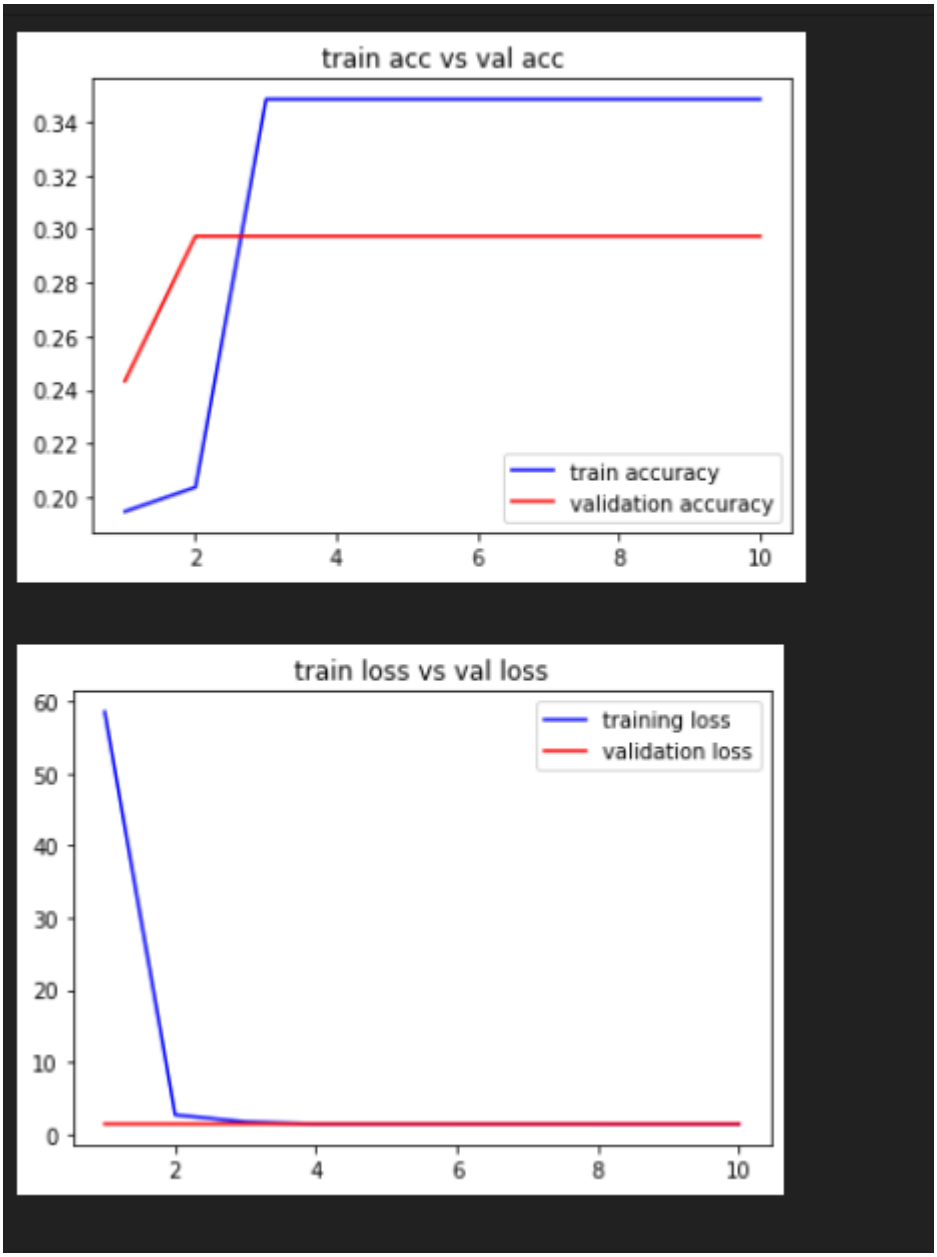
```
model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])


history1 = model1.fit_generator(train_generator, epochs=10, verbose=2, validation_data=validation_generator)
```

Y se entrenó por 10 épocas:

```
Epoch 1/10
7/7 - 5s - loss: 58.5331 - acc: 0.1946 - val_loss: 1.3861 - val_acc: 0.2432 - 5s/epoch - 718ms/step
Epoch 2/10
7/7 - 5s - loss: 2.6390 - acc: 0.2036 - val_loss: 1.3856 - val_acc: 0.2973 - 5s/epoch - 688ms/step
Epoch 3/10
7/7 - 5s - loss: 1.6611 - acc: 0.3484 - val_loss: 1.3852 - val_acc: 0.2973 - 5s/epoch - 662ms/step
Epoch 4/10
7/7 - 5s - loss: 1.3847 - acc: 0.3484 - val_loss: 1.3847 - val_acc: 0.2973 - 5s/epoch - 696ms/step
Epoch 5/10
7/7 - 5s - loss: 1.3838 - acc: 0.3484 - val_loss: 1.3841 - val_acc: 0.2973 - 5s/epoch - 732ms/step
Epoch 6/10
7/7 - 5s - loss: 1.3827 - acc: 0.3484 - val_loss: 1.3836 - val_acc: 0.2973 - 5s/epoch - 666ms/step
Epoch 7/10
7/7 - 5s - loss: 1.3819 - acc: 0.3484 - val_loss: 1.3832 - val_acc: 0.2973 - 5s/epoch - 659ms/step
Epoch 8/10
7/7 - 5s - loss: 1.3810 - acc: 0.3484 - val_loss: 1.3826 - val_acc: 0.2973 - 5s/epoch - 705ms/step
Epoch 9/10
7/7 - 5s - loss: 1.3801 - acc: 0.3484 - val_loss: 1.3820 - val_acc: 0.2973 - 5s/epoch - 649ms/step
Epoch 10/10
7/7 - 6s - loss: 1.3790 - acc: 0.3484 - val_loss: 1.3815 - val_acc: 0.2973 - 6s/epoch - 893ms/step
```

Se puede ver cómo fue evolucionando el modelo durante las distintas épocas:



Y al final con el dataset de test da una precisión del 20%

```
test_loss1, test_acc1 = model1.evaluate_generator(test_generator)
test_acc1
```

```
0.20270270109176636
```

Segundo modelo

Como segundo acercamiento se decidió deshacerse de la capa convolutiva y apostar por una realizada con transferencia de aprendizaje. Esta técnica consiste en utilizar partes de modelos ya existentes y cuya eficiencia está comprobada y adaptarlos a otros tipos de problemas. En este caso se usó la vgg16 que es un modelo compuesto por capas convolutivas que se usa para la clasificación de imágenes. La idea es que los componentes básicos con los que fue entrenado la vgg16 también son aplicables a los pokemones.

En este caso se usó la capa de vgg16, una de flatten y dos densas.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_4 (Flatten)         (None, 25088)             0

 dense_8 (Dense)             (None, 16)                401424

 dense_9 (Dense)             (None, 4)                 68

=================================================================
Total params: 15,116,180
Trainable params: 401,492
Non-trainable params: 14,714,688
_________________________________________________________________
```

Se compiló con estas hiperparámetros:

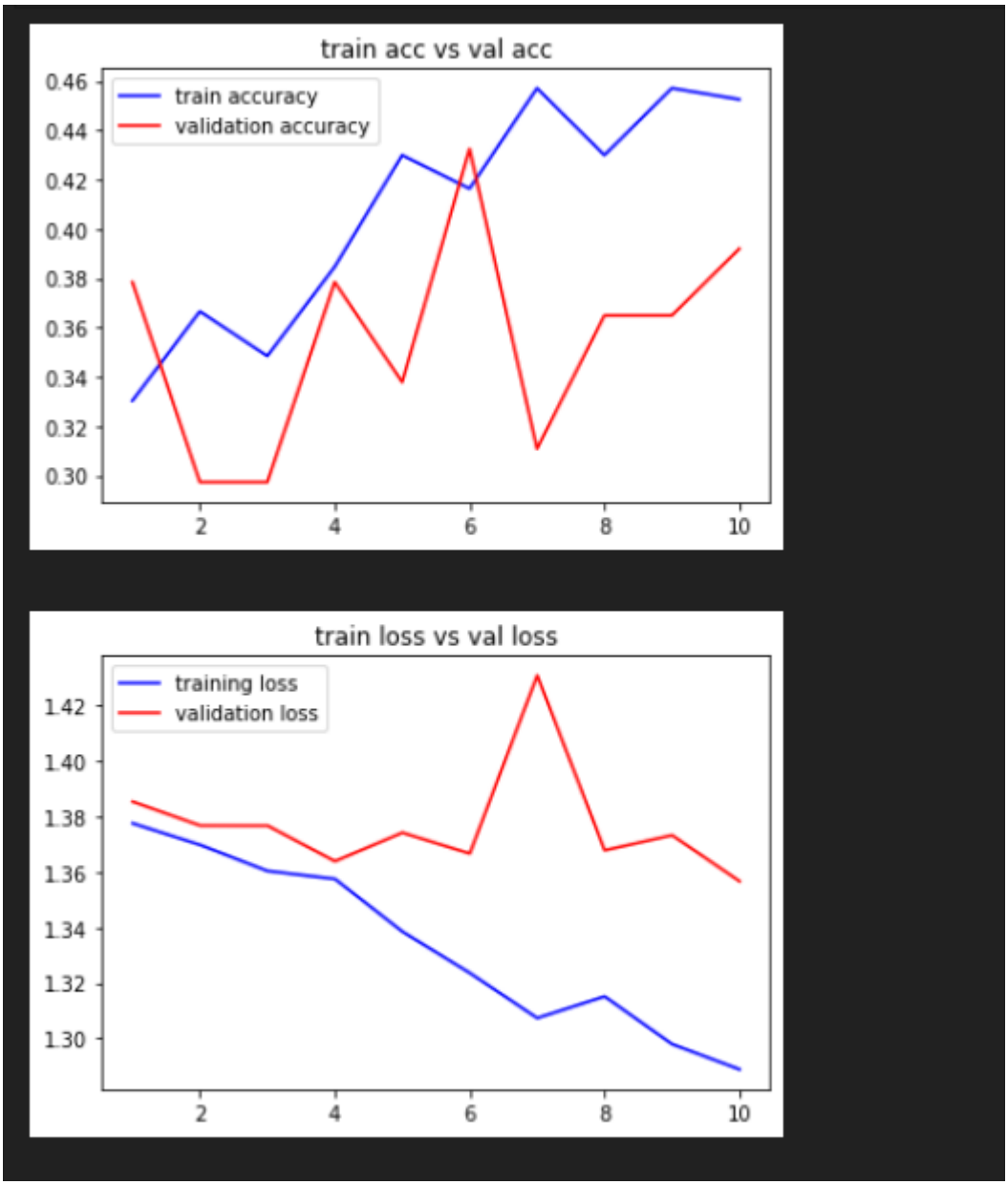
```
model2.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])


history2 = model2.fit_generator(train_generator, epochs=10, verbose=2, validation_data=validation_generator)
```

Y se corrió por 10 épocas:

```
Epoch 1/10
7/7 - 91s - loss: 1.3775 - acc: 0.3303 - val_loss: 1.3854 - val_acc: 0.3784 - 91s/epoch - 13s/step
Epoch 2/10
7/7 - 101s - loss: 1.3698 - acc: 0.3665 - val_loss: 1.3767 - val_acc: 0.2973 - 101s/epoch - 14s/step
Epoch 3/10
7/7 - 96s - loss: 1.3603 - acc: 0.3484 - val_loss: 1.3767 - val_acc: 0.2973 - 96s/epoch - 14s/step
Epoch 4/10
7/7 - 89s - loss: 1.3575 - acc: 0.3846 - val_loss: 1.3639 - val_acc: 0.3784 - 89s/epoch - 13s/step
Epoch 5/10
7/7 - 92s - loss: 1.3385 - acc: 0.4299 - val_loss: 1.3742 - val_acc: 0.3378 - 92s/epoch - 13s/step
Epoch 6/10
7/7 - 89s - loss: 1.3236 - acc: 0.4163 - val_loss: 1.3666 - val_acc: 0.4324 - 89s/epoch - 13s/step
Epoch 7/10
7/7 - 90s - loss: 1.3073 - acc: 0.4570 - val_loss: 1.4309 - val_acc: 0.3108 - 90s/epoch - 13s/step
Epoch 8/10
7/7 - 96s - loss: 1.3151 - acc: 0.4299 - val_loss: 1.3678 - val_acc: 0.3649 - 96s/epoch - 14s/step
Epoch 9/10
7/7 - 90s - loss: 1.2979 - acc: 0.4570 - val_loss: 1.3732 - val_acc: 0.3649 - 90s/epoch - 13s/step
Epoch 10/10
7/7 - 92s - loss: 1.2888 - acc: 0.4525 - val_loss: 1.3567 - val_acc: 0.3919 - 92s/epoch - 13s/step
```

Se puede ver cómo fue evolucionando el modelo durante las distintas épocas:



Y se consiguió una precisión en el conjunto de pruebas de 25.6%.

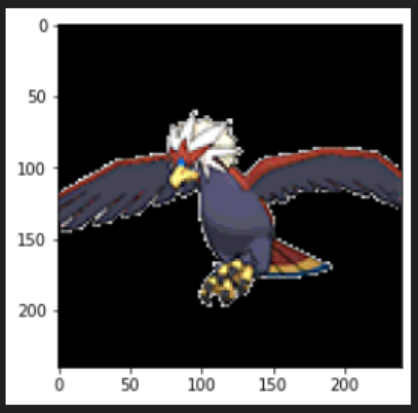
```
test_loss, test_acc = model2.evaluate_generator(test_generator)
test_acc
```

```
0.2567567527294159
```

Predicciones

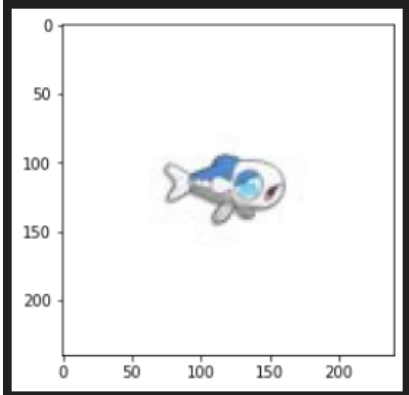
Pokémon normal lo predice correctamente

```
1/1 [==============================] - 0s 170ms/step
Normal
```



Pokémon de agua lo predice incorrectamente

```
1/1 [==============================] - 0s 191ms/step
Normal
```



Conclusiones:

El dataset contaba con los problemas de que tenía muy pocos datos para entrenar y los tipos son una característica que a ratos parece arbitraria por lo que es difícil llegar un modelo con una precisión alta. Aun así se pudo notar que usar transferencia de aprendizaje permitió una mejora en el modelo de clasificación.