En este entregable se realizó un modelo de Deep Learning con ayuda del *Pokemon Image Dataset* (https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types/) que contiene imágenes de distintos pokemones con su respectivo tipo. La intención fue realizar un modelo de clasificación de los pokemones por tipo 1.

Lo primero que se hizo fue unir el csv que se nos entrega con el nombre de los archivos:

```python
#load csv and add filenames
df = pd.read_csv("pokemon.csv")
original_dataset = "images/images/"
filenames = os.listdir(original_dataset)
filenames_dict = {}
for filename in filenames:
    filenames_dict[filename.split(".")[0]] = filename
df = df.assign(Filename=df.Name.map(filenames_dict))
df
```

[9] ✓ 0.9s

	Name	Type1	Type2	Filename
0	bulbasaur	Grass	Poison	bulbasaur.png
1	ivysaur	Grass	Poison	ivysaur.png
2	venusaur	Grass	Poison	venusaur.png
3	charmander	Fire	NaN	charmander.png
4	charmeleon	Fire	NaN	charmeleon.png
...	...	...	...	...
804	stakataka	Rock	Steel	stakataka.jpg
805	blacephalon	Fire	Ghost	blacephalon.jpg
806	zeraora	Electric	NaN	zeraora.jpg
807	meltan	Steel	NaN	meltan.jpg
808	melmetal	Steel	NaN	melmetal.jpg

809 rows × 4 columns

Algo que se pudo observar es que hay tipos con pocas ocurrencias en el dataset:

```python
#Get valid types (any that has more than 50 occurrences)
df.groupby("Type1")["Type1"].count().sort_values(ascending=False)
# valid_types = list(df.groupby("Type1")["Type1"].count().loc[lambda x: x > 70].index)
# print(f"Valid types: {valid_types}")
```

[5] ✓ 0.8s

```
Type1
Water       114
Normal      105
Grass        78
Bug          72
Fire         53
Psychic      53
Rock         46
Electric     40
Poison       34
Ground       32
Fighting     29
Dark         29
Ghost        27
Dragon       27
Steel        26
Ice          23
Fairy        18
Flying        3
Name: Type1, dtype: int64
```

Por esa misma razón se decidió usar solo los que tenían más de 70 ocurrencias:

```python
#Get valid types (any that has more than 50 occurrences)
# df.groupby("Type1")["Type1"].count().sort_values(ascending=False)
valid_types = list(df.groupby("Type1")["Type1"].count().loc[lambda x: x > 70].index)
print(f"Valid types: {valid_types}")
```

```
Valid types: ['Bug', 'Grass', 'Normal', 'Water']
```

```python
#Fitler pokemon by valid types
valid_pokemon = df[df["Type1"].isin(valid_types)];
valid_pokemon = valid_pokemon[["Name", "Type1", "Filename"]]
valid_pokemon.reset_index(inplace=True)
valid_pokemon
```

	index	Name	Type1	Filename
0	0	bulbasaur	Grass	bulbasaur.png
1	1	ivysaur	Grass	ivysaur.png
2	2	venusaur	Grass	venusaur.png
3	6	squirtle	Water	squirtle.png
4	7	wartortle	Water	wartortle.png
...	...	...	...	...
364	786	tapu-bulu	Grass	tapu-bulu.jpg
365	787	tapu-fini	Water	tapu-fini.jpg
366	793	buzzwole	Bug	buzzwole.jpg
367	794	pheromosa	Bug	pheromosa.jpg
368	797	kartana	Grass	kartana.jpg

369 rows × 4 columns

Se realizó el split de los datos en train, validation y test y se crearon 3 carpetas con cada parte del split y en cada una de ellas se creó una para los tipos. Posteriormente se exportaron las imagenes:

```python
base_dir = "pokemon/"
os.makedirs(base_dir, exist_ok=True)

train_dir = os.path.join(base_dir, "train")
os.makedirs(train_dir, exist_ok=True)
validation_dir = os.path.join(base_dir, "validation")
os.makedirs(validation_dir, exist_ok=True)
test_dir = os.path.join(base_dir, "test")
os.makedirs(test_dir, exist_ok=True)

for type in valid_types:
    new_dir = os.path.join(train_dir, type)
    os.makedirs(new_dir, exist_ok=True)

for type in valid_types:
    new_dir = os.path.join(validation_dir, type)
    os.makedirs(new_dir, exist_ok=True)

for type in valid_types:
    new_dir = os.path.join(test_dir, type)
    os.makedirs(new_dir, exist_ok=True)

for index, row in train.iterrows():
    src = os.path.join(original_dataset, row["Filename"])
    folder = os.path.join(train_dir, row["Type1"])
    dst = os.path.join(folder, row["Filename"])
    shutil.copyfile(src, dst)
```

```python
for index, row in validation.iterrows():
    src = os.path.join(original_dataset, row["Filename"])
    folder = os.path.join(validation_dir, row["Type1"])
    dst = os.path.join(folder, row["Filename"])
    shutil.copyfile(src, dst)

for index, row in test.iterrows():
    src = os.path.join(original_dataset, row["Filename"])
    folder = os.path.join(test_dir, row["Type1"])
    dst = os.path.join(folder, row["Filename"])
    shutil.copyfile(src, dst)
```

Se crearon un ImageDataGenerator por cada uno de los folders:

```python
datagen = ImageDataGenerator()
train_generator = datagen.flow_from_directory(train_dir, color_mode='rgb', class_mode="categorical", target_size=(240, 240))
validation_generator = datagen.flow_from_directory(validation_dir, color_mode='rgb', class_mode="categorical", target_size=(240, 240))
test_generator = datagen.flow_from_directory(test_dir, class_mode="categorical", color_mode='rgb', target_size=(240, 240))
```

```
Found 221 images belonging to 4 classes.
Found 74 images belonging to 4 classes.
Found 74 images belonging to 4 classes.
```

Posteriormente se procedió a hacer el primer modelo que está compuesto por dos capas convolutivas y dos densas:

```python
model1 = models.Sequential()
model1.add(layers.Conv2D(16, (3,3), activation="relu", input_shape=(240, 240, 3)))
model1.add(layers.MaxPooling2D(3,3))
model1.add(layers.Conv2D(32, (3,3), activation="relu"))
model1.add(layers.Flatten())
model1.add(layers.Dense(8, activation="relu"))
model1.add(layers.Dense(4, activation="softmax"))

model1.summary()
```

```
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
_________________________________________________________________
```

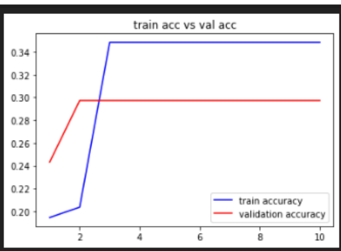
```python
model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])
```

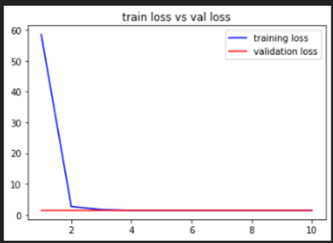
Se entrenó el modelo por 10 épocas:

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

```
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
```

Los resultados fueron los siguientes:






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

```
0.20270270109176636
```

Para el siguiente modelo se decidió hacer transfer learning con ayuda del modelo VGG16:

```
conv_base = VGG16(weights=None, include_top=False, input_shape=(240, 240, 3))

model2 = models.Sequential()
model2.add(conv_base)
model2.add(layers.Flatten())
model2.add(layers.Dense(16, activation="relu"))
model2.add(layers.Dense(4, activation="softmax"))

conv_base.trainable = False

model2.summary()
```

```
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
_________________________________________________________________
```

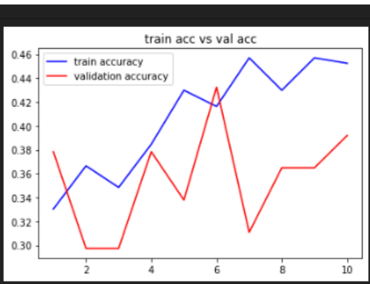
```
model2.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])
```

Igualmente se entrenó por 10 épocas:

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

```
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
```

Y los resultados mejoraron:




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

```
0.2567567527294159
```

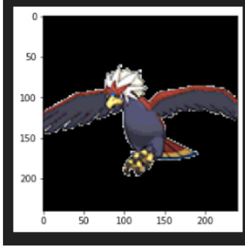
Se realizaron algunas predicciones con el dataset de test:

```python
img = load_img("./pokemon/test/Normal/braviary.png", target_size=(240, 240))
img_tensor = img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis = 0)
img_tensor /= 255.

prediction = model2.predict(img_tensor)
print(valid_types[np.argmax([prediction])])
plt.imshow(img_tensor[0])
plt.show()
```

0.4s

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

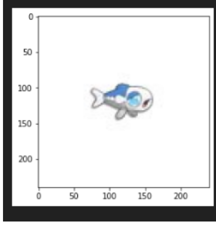


```python
img = load_img("./pokemon/test/Water/wishiwashi-solo.jpg", target_size=(240, 240))
img_tensor = img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis = 0)
img_tensor /= 255.

prediction = model2.predict(img_tensor)
print(valid_types[np.argmax([prediction])])
plt.imshow(img_tensor[0])
plt.show()
```

0.4s

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



La mejora en el segundo modelo es que se está usando una capa convolutiva que ha podido ser entrenada con un dataset más amplio por lo que es mejor reconociendo patrones en imágenes que la que se puede entrenar con unos cientos de datos.