Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Inteligencia Artificial para la ciencia de datos TC3007C

Módulo 2: Implementación de un modelo de Deep Learning

Alejandro López Hernández A01733984

4 de noviembre de 2022

Descripción del problema

Sin duda alguna Pokémon ha sido un fenómeno mundial; generaciones crecieron viendo la serie animada, jugando los videojuegos, leyendo el manga, coleccionando cartas, o teniendo mercancía como juguetes, ropa, llaveros, etcétera. Conforme siguen saliendo nuevos juegos y series animadas, cada vez la lista de Pokémon se hace más grande, a tal punto que a veces resulta difícil recordar el nombre de algún Pokémon viendo su imagen. Es por ello que podría emplearse la Inteligencia Artificial para crear un modelo capaz de obtener el nombre de un pokémon a partir de su imagen.

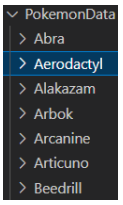
Dataset empleado

Principalmente, se tenía planeado emplear un dataset completo con 12000 imágenes con los primeros 151 pokémon de la región de Kanto; pero por cuestiones de tiempo y poder de cómputo, se optó por emplear sólo 15 pokémon: los 3 iniciales con sus evoluciones, los 5 pokemon legendario y Pikachu. Por lo tanto, el dataset original pasó de 12000 imágenes a 1154 imágenes de 15 clases diferentes.
Link al dataset de Kaggle:

https://www.kaggle.com/datasets/unexpectedscepticism/11945-pokemon-from-first-gen

Manejo de imágenes

La estructura del dataset corresponde a un directorio que contiene directorios, que a su vez contienen imágenes de un pokémon según el nombre de la carpeta en la que se encuentren. La estructura es similar a la siguiente:



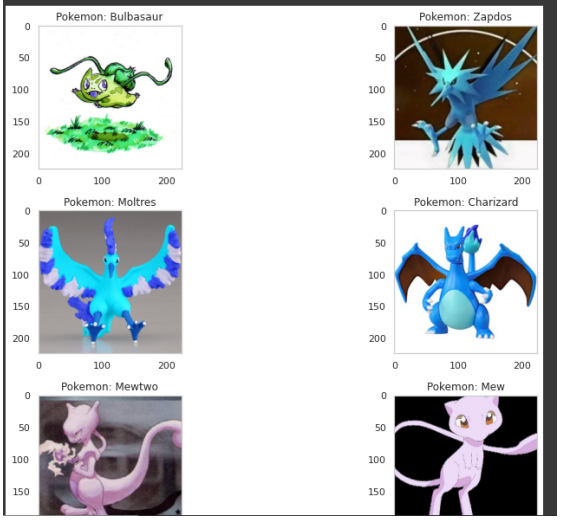
Una vez considerando dicha estructura, una alternativa óptima para generar un set de datos utilizable para el modelo sería capturar todas las imágenes en un arreglo y etiquetarlos en otro arreglo según la carpeta de la que provino, tal y como se muestra en el fragmento de abajo

```python
#funcion para generar un dataset de imagenes usable a partir de un directorio con directorios de imagenes
def make_train_data(poke_name, DIR):
  for img in tqdm(os.listdir(DIR)):
    label = poke_name
    path = os.path.join(DIR, img)
    img = cv2.imread(path, cv2.IMREAD_COLOR)
    img = cv2.resize(img, (img_size, img_size))
    X.append(np.array(img))
    Z.append(str(label))
```

```python
i = 1
#para cada pokemon en la lista se hace llamado a la funcion
for pokemon in pokemon_names:
  dir = './PokemonData/' + pokemon
  make_train_data(pokemon, dir)
  print(i , ' :' , len(X))
  i = i + 1
```

Con tal de que las imágenes se agruparon correctamente, se hace una demostración gráfica del arreglo de imágenes, seleccionando unas cuantas al azar

```python
#demostracion grafica del dataset
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=rn.randint(0,len(Z))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('Pokemon: '+Z[l])
        ax[i,j].grid(None)
plt.tight_layout()
```



Luego, se aplicó Label Encoding al arreglo de etiquetas con tal de asignarles valores numéricos a los nombres de los pokemon y así poderlos clasificar de una mejor manera. Después de este proceso, se determinaron 15 clases para las imágenes; es decir, 15 posibles nombres de pokémon. Seguido de ello, se transformó el arreglo de imágenes en un arreglo de numpy, así como también se ajustó proporcionalmente el valor dentro de cada imagen. Sabiendo que el código RGB de cada compuesto de un color va de 0 a 255, se dividieron todos los valores del arreglo entre 255 para así tener una escala de 0

a 1 en esos datos; logrando que el modelo pueda procesar datos aparentemente más ligeros.

```
#label encoding para las etiquetas
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,15)
#transformacion de los valores de las imagenes a escala 0 - 1
X=np.array(X)
X=X/255
```

Finalmente con el dataset general generado y segmento en X para imágenes e Y para sus etiquetas, se hace el split de test y train haciendo uso de la función correspondiente de la librería de scikit learn.

```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.20,random_state=42) #split del dataset en train y test
```

Modelo empleado

El modelo de Deep Learning a implementar en Tensorflow/Keras se trata de una Red Neuronal Convolutiva (CNN). Es una arquitectura de red para deep learning que aprende directamente de los datos, sin necesidad de extraer características manualmente.

Estas redes son particularmente útiles para encontrar patrones en imágenes para reconocer objetos, caras y escenas. También resultan eficaces para clasificar datos sin imágenes, tales como datos de audio, series temporales y señales.

La base principal de las CNN se constituye en la convolución. La convolución consiste en tomar un conjunto de pixeles cercanos de la imagen de entrada e ir operando matemáticamente por producto escalar contra una pequeña matriz definida previamente denominada kernel.

Arquitectura del primer modelo

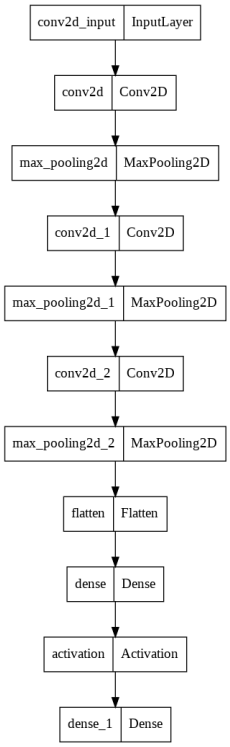
El modelo preliminar consta de una red de 3 capas: 1 de entrada y 2 profundas. Cada capa se compone de la siguiente forma:

- Conv2D: es la capa convolucional en 2 dimensiones. Algunos de sus parámetros son filtro (cantidad de Kernels a emplear en la capa, en este modelo hay de 32,

64 y 128), tamaño del kernel (para el input layer es de 5x5, mientras que para las otras 2 son de 3x3), algoritmo de activación (todas las capas emplean ReLU); así como también tamaño del input que corresponde a la input layer y se refiere al tamaño de las imágenes.

- MaxPooling2D: El pooling es una operación que permite analizar el contenido de una imagen por regiones. En este caso, el tamaño del pooling será de 2x2.

Después de las capas, viene una pila de Flatten. Tal como dice su nombre, se encarga de "aplanar" el input. Asimismo, le sigue 2 capas Dense con una capa de activación entre ellas con algoritmo ReLU de activación general. La última de las 2 capas Dense mencionada emplea el algoritmo de activación softmax a manera de última capa de la red. Una explicación gráfica y el resumen de la CNN se muestran a continuación

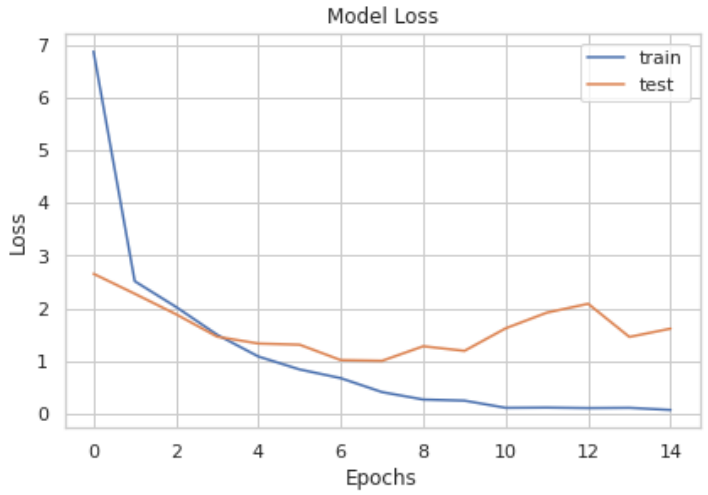


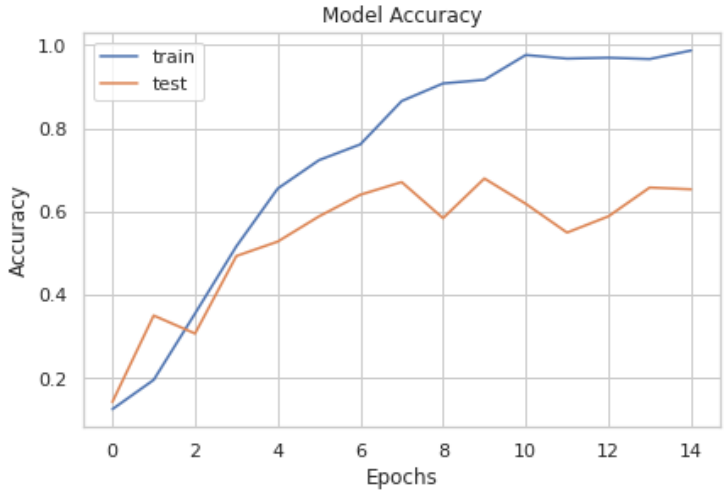

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
 2D)

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 512)               51380736

 activation (Activation)     (None, 512)               0

 dense_1 (Dense)             (None, 15)                7695

=================================================================
Total params: 51,483,215
Trainable params: 51,483,215
Non-trainable params: 0
_________________________________________________________________
```

Con la definición del modelo, se le hizo un ajuste con el dataset de entrenamiento. El ajuste cuenta con los siguientes hiper parámetros: 15 épocas, 128 batches por época, validation_data con el data set de pruebas (x_test, y_test).

Análisis y resultados

Tras poco más de media hora de ejecución, el ajuste de este modelo mostró el siguiente comportamiento. Principalmente se denota una situación común de overfitting dentro de la CNN. Esto quiere decir que el modelo es demasiado bueno para los datos del train pero no resulta consistente con los datos de prueba. A pesar de ello, un punto rescatable es que en ambos sets el modelo muestra una tendencia común de descendencia en cuanto a la pérdida o loss; mientras que se muestra con una relativa alza en términos de la asertividad del modelo conforme pasan las épocas.

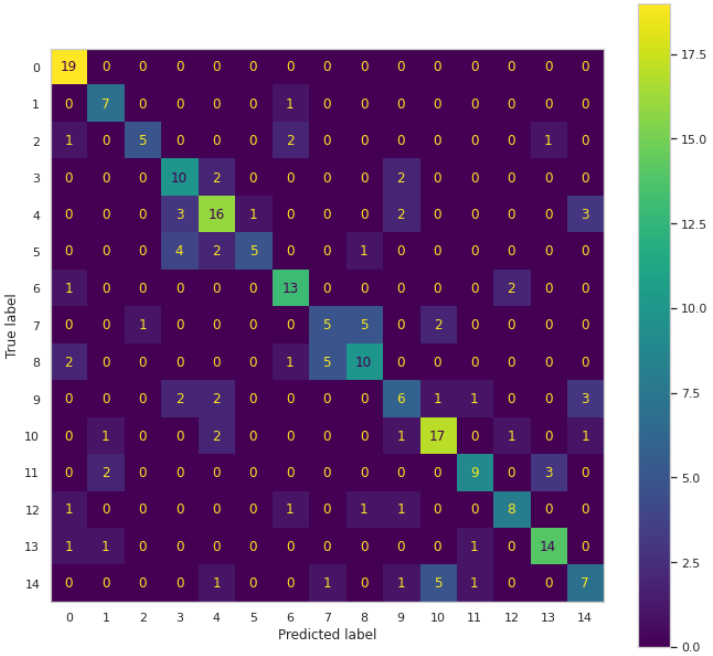




Tras estas gráficas, se obtuvo que el modelo a como está posee un porcentaje de accuracy del 65.36%. Del mismo modo se obtuvo la matriz de confusión del modelo que representa el funcionamiento del modelo respecto a las 15 clases determinadas, donde

se puede ver la diagonal que representa los datos acertados correctamente; mientras que también se denota una dispersión respecto a las clases que no fueron predecidas correctamente.

```
Test loss Model: 1.612815022468567
Test accuracy Model: 0.6536796689033508
```



Mejoramiento del modelo

Data Augmentation

Data augmentation es la generación artificial de datos por medio de perturbaciones o alteraciones hacia los datos originales. Esto permite aumentar tanto en tamaño como en diversidad el dataset de entrenamiento. Esto permite combatir el overfitting que se suele presentar en modelos como el CNN implementado previamente En el caso de este modelo, se emplea una ligera alteración en el zoom, la rotación horizontal, el zoom de cada imagen, etc.

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=False)
datagen.fit(x_train)
```

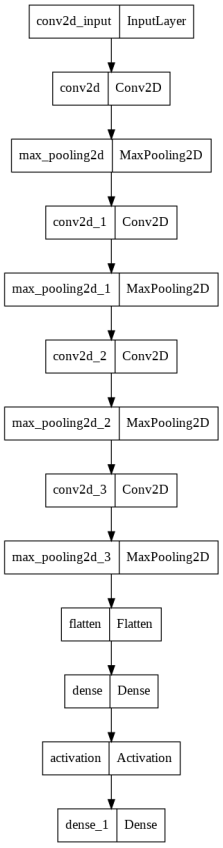
Reduce Learning Rate on Plateau

Se trata de una función de Keras que permite autorregular el learning rate de un modelo en caso de que una determinada métrica no tenga un aumento significativo con el paso de unas cuantas épocas. En este caso, la métrica empleada para esta función es el accuracy de la validación al momento del ajuste del modelo.

```
#Funcion para reducir el learning rate en caso de que un parametro (accuracy d
from keras.callbacks import ReduceLROnPlateau
red_lr= ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
```

Arquitectura del modelo mejorado

El nuevo modelo mantiene la misma base que el modelo previo, con la diferencia de que se añadió una capa extra con un filtro de 32 kernels; resultando en 4 capas ahora. El resto del modelo se mantiene prácticamente igual en términos de estructura; incluso con los mismos parámetros previamente explicados

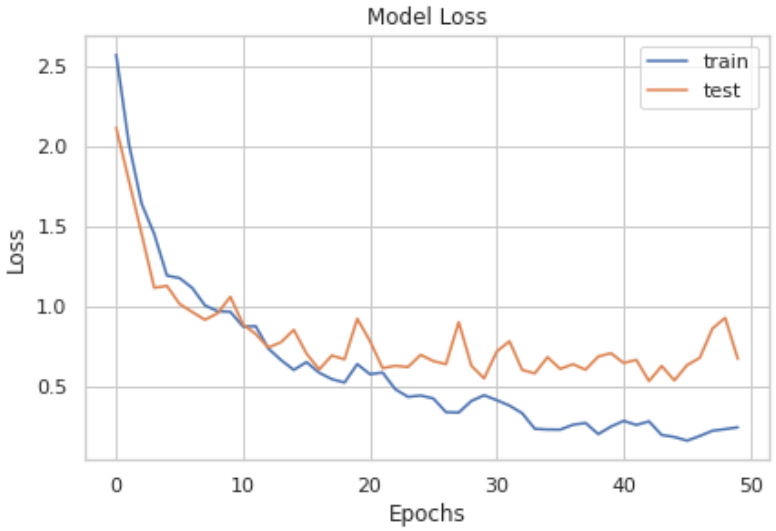


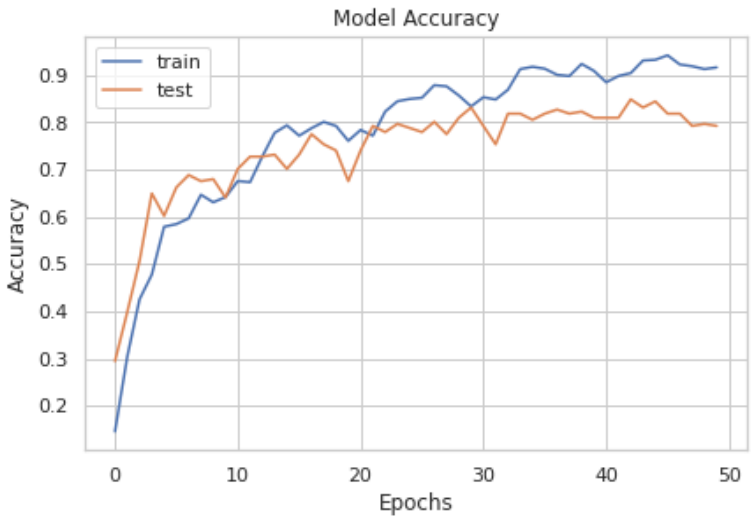

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 activation (Activation)     (None, 512)               0

 dense_1 (Dense)             (None, 15)                7695

=================================================================
Total params: 12,957,295
Trainable params: 12,957,295
Non-trainable params: 0
_________________________________________________________________
```

Con la definición del modelo, se le hizo un ajuste con el dataset de entrenamiento. El ajuste cuenta con los siguientes hiper parámetros: 50 épocas, 100 batches por época, validation_data con el data set de pruebas (x_test, y_test), pasos por épocas definidos y en función del tamaño de x_train sobre la cantidad de batches previamente mencionada.

Análisis y resultados

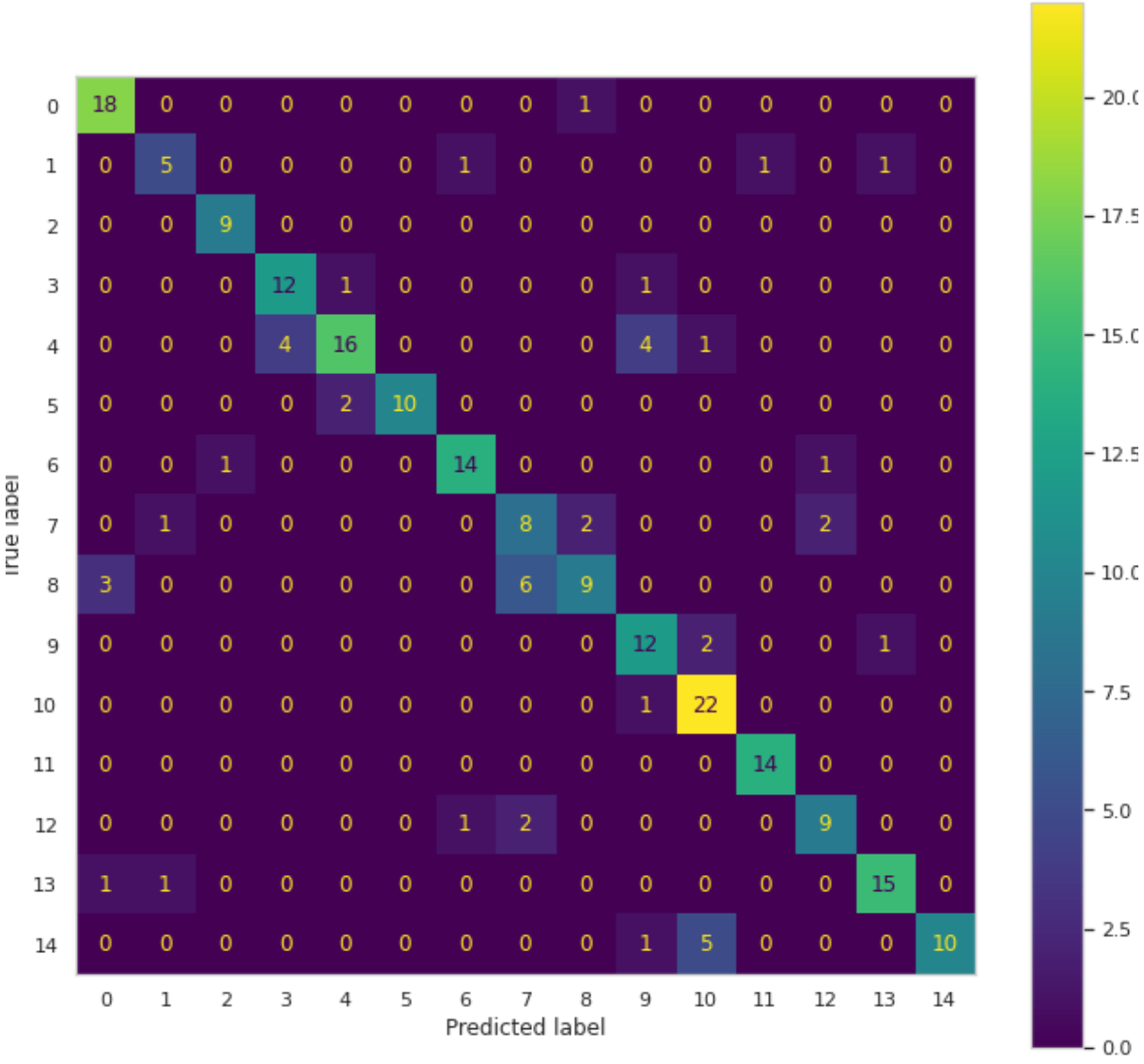
Tras poco más de 1 hora y 40 minutos de ejecución, el ajuste de este modelo mostró el siguiente comportamiento. Como se puede observar, el overfitting ya es significativamente menor respecto al primer boceto del modelo. Esto debido a que el comportamiento del modelo tanto en train como en test se mantiene relativamente similar y con tendencias claras.





Tras estas gráficas, se obtuvo que el modelo a como está posee un porcentaje de accuracy del 79.22%. Del mismo modo se obtuvo la matriz de confusión del modelo que representa el funcionamiento del modelo respecto a las 15 clases determinadas, donde se puede ver la diagonal que representa los datos acertados correctamente; mientras que también se denota una menor dispersión respecto a las clases que no fueron predecidas correctamente respecto al primer modelo.

```
Test loss Model 2: 0.6754796504974365
Test accuracy Model 2: 0.7922077775001526
```



Sin duda hubo una notoria mejoría entre modelos con un ligero ajuste de hiper parámetros. Incluso si se considera la misma época en la que terminó de ajustarse el primer modelo (15), el segundo modelo tiene aproximadamente un 5% más de

accuracy. En adición a ello, fue un acierto la implementación del Data Augmentation con tal de poder enriquecer los datos a partir de la alteración de sus propias características; así como también la autorregulación del Learning Rate con tal de optimizar el performance de cada época

Demostración de modelo mejorado

https://youtu.be/YCNQdehrIqY