



Inteligencia artificial avanzada para la ciencia de datos II (Gpo 502)

**Momento de Retroalimentación: Módulo 2 Implementación
de un modelo de deep learning. (Portafolio
Implementación)**

Benjamín Valdés Aguirre

Alumno

Daniel Munive Meneses

A01734205

04/11/2020

Objetivo

Como objetivo de este entregable se buscaba crear un modelo que permitiera la clasificación de distintos personajes de anime, esto se logra haciendo uso de un modelos de redes neuronales que vaya aprendiendo a identificar los distintos patrones de las imágenes de los personajes, para con esto poder dividirlas de manera correcta.

Dataset

Para este modelo se hizo uso de un dataset de 185 fotos divido de la siguiente forma

Dentro de la carpeta 'Anime Girls'



Nos encontramos con las siguientes carpetas



Siendo que dentro de cada carpeta hay un aproximado de 60 fotos de cada personaje, esto con la excepción de test que es una carpeta con diferentes imágenes que posteriormente pasaría a utilizar para probar mis modelos, las carpetas se ven de la siguiente manera

Archivos

Nombre

Nombre ↓

Yor Forger

X

Detalles Actividad

Usuarios con acceso

No compartido

Administrador acceso

Propiedades del sistema

Tipo Carpeta de Google Drive
Ubicación Anime_Girls
Propietario yo
Modificado 15:53 por mí
Abierto 00:47 por mí
Creado 15:53 con Google Drive Web (Unverified)

Agregar descripción

Archivos

Nombre

Nombre ↓

Micchon Shikimori

X

Detalles Actividad

Usuarios con acceso

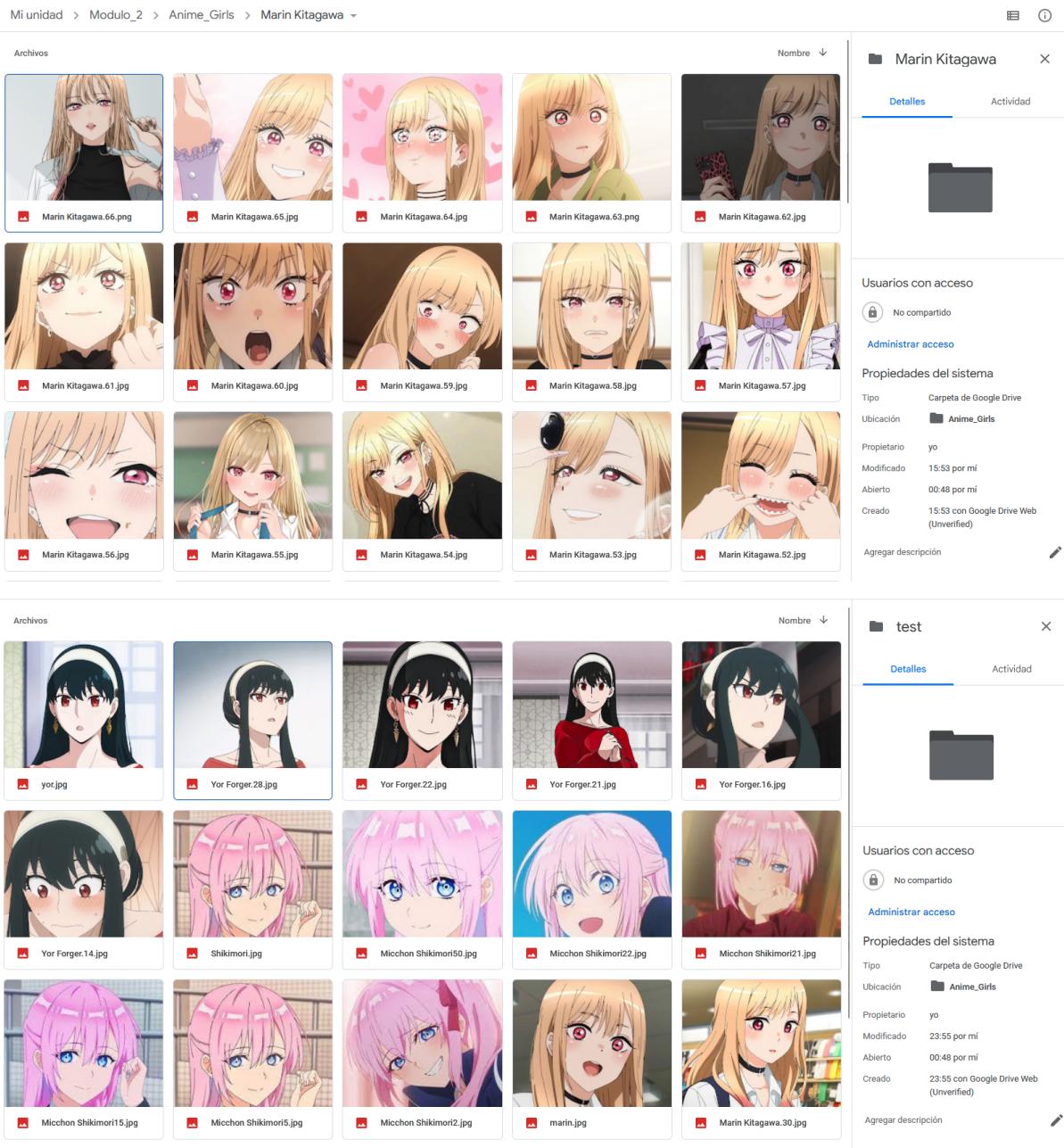
No compartido

Administrador acceso

Propiedades del sistema

Tipo Carpeta de Google Drive
Ubicación Anime_Girls
Propietario yo
Modificado 23:27 por mí
Abierto 00:48 por mí
Creado 23:27 con Google Drive Web (Unverified)

Agregar descripción



Train, Test, Validation

Posteriormente para la división de Train y Test dentro de mí procesamiento de datos, hago uso de la función 'train_test_split'

```
[29] 1 x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,random_state=42)
```

con un porcentaje de de 70% para train y de 30% para test

Para validación dentro de mi modelaje hago uso del parámetro ‘validation_split’, dentro de la función de ‘fit’, donde asingo un porcentaje de 20% del train para este

```
1 #Para entrenar nuestro algoritmo havemo uso de fit (Que significa cada variable se describe en la parte de parametros)
2 history = cnn.fit(x_train,y_train, epochs = epochs, validation_split = 0.2)
3
```

Hiperparametros

Para mis hyperparameter inicialmente se trabajó con los siguientes parámetros

```
1 #Parametro para red neuronal
2 epochs = 15 #Número de veces que se iterara nuestro set de datos
3 height, width = 256, 256 #Tamaño al que se procesaran las imagenes de nuestra database 100px x 100px
4 batch_size = 5 #Número de imágenes que procesara en cada paso
5 steps_per_epoch = 5 #Número de veces que se procesara la informacion en cada época
6 v_steps = 25 #Número de veces que porcesaran con nuestros datos de Validacion
7
8 #Número de filtros de cada convolucion, siendo esto la profundidad de nuestra imagen
9 convolution_1 = 32
10 convolution_2 = 64
11
12 #Tamaño de cada convolucion
13 size_convolution_1 = (3,3)
14 size_convolution_2 = (2,2)
15
16 #Tamaño que se usara para el MaxPooling
17 MaxPooling2D_size = (2,2)
18
19 #Número de clases con las que se va trabajar, en mi clase son 10 ya que se entrenara la red para que esta pueda indentificar a 10 personajes distintos
20 classN = 3
21
22 #Tamaño de ajuste de nuestra red neuronal (mientras mas pequeño mejor)
23 lr = 0.0005
```

sin embargo a razón de mejorar mi modelo realice algunos cambios en estos en ajustes posteriores del modelo

Modelo 1

Para el Modelo 1 que vendría a ser mi modelo ‘mejorado’, hago uso de una red neuronal de distintas capas convolucionales, las cuales quedaron de la siguiente manera

```
#Agregamos la primera capa de convolución
cnn.add(Convolution2D(convolution_1,
                      size_convolution_1,
                      padding='same',
                      input_shape=(height, width, 3), #Input shape normalmente solo se
agrega en la primera capa de la red
                      activation='relu'))

cnn.add(MaxPooling2D(pool_size= MaxPooling2D_size))

#Agregamos la segunda capa de convolución
cnn.add(Convolution2D(convolution_2,
                      size_convolution_2,
                      padding='same',
                      activation='relu'))

cnn.add(MaxPooling2D(pool_size= MaxPooling2D_size))
```

```
cnn.add(Flatten()) #Aquí se trabaja con la imagen después de su proceso la vamos a volver plana osea que únicamente nos vamos a quedar con una dimensión, que tendrá toda la información de nuestra red neuronal

cnn.add(Dense(256, activation= 'relu')) #Una nueva capa con 256 neuronas y con activación relu

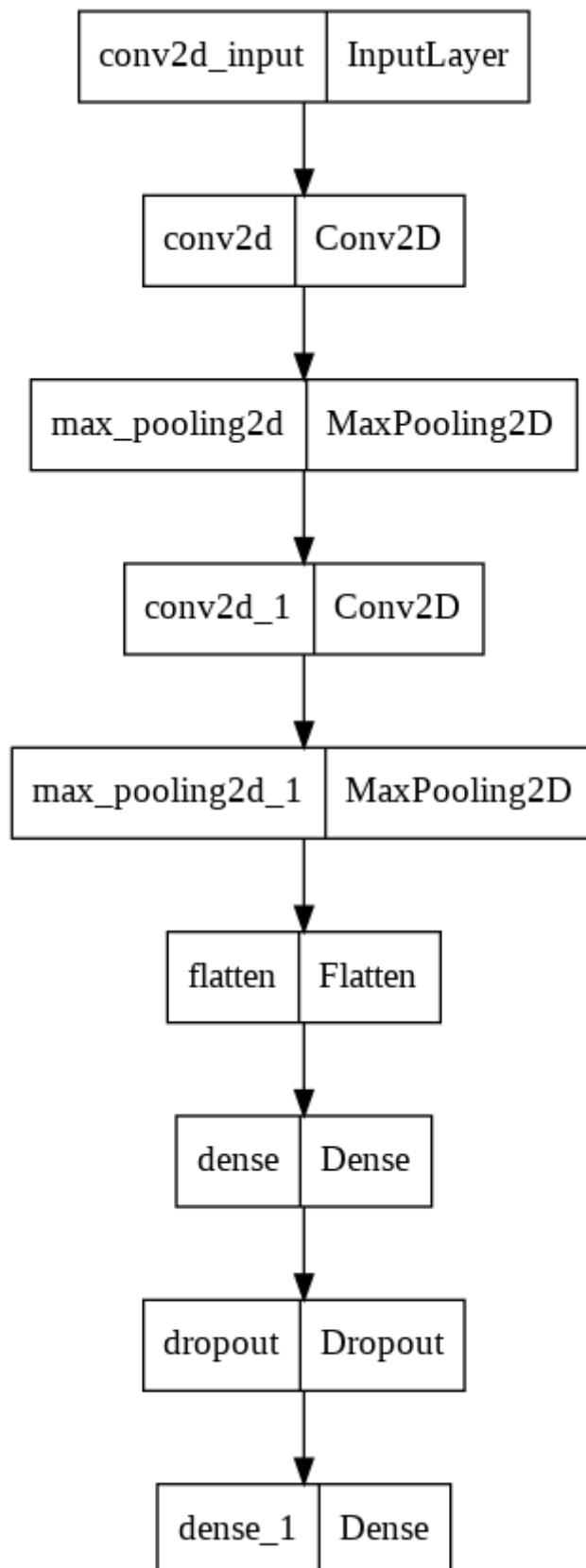
cnn.add(Dropout(0.5)) #Se agrega DropOut para evitar sobre ajustar nuestra red neuronal

cnn.add(Dense(classN, activation='softmax')) #SoftMax nos ayudará a darnos un porcentaje de que tanta probabilidad hay de que la imagen sea un personaje

cnn.summary() #Imprimo un resumen del modelo

cnn.compile(loss='categorical_crossentropy', #Ya que recordemos que nuestra red neuronal es categórica
             optimizer=optimizers.Adam(lr=lr), #Nuestro optimizador tiene un learning rate que definimos en nuestra variable de arriba
             metrics=['accuracy']) #Utiliza como métrica el porcentaje de que tan bien está aprendiendo nuestra red neuronal
```

Estructura de red



```

↳ Model: "sequential"
-----  

Layer (type)          Output Shape       Param #
-----  

conv2d (Conv2D)      (None, 250, 250, 32)    896  

max_pooling2d (MaxPooling2D) (None, 125, 125, 32)    0  

)  

conv2d_1 (Conv2D)     (None, 125, 125, 64)    8256  

max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)    0  

2D)  

flatten (Flatten)     (None, 246016)        0  

dense (Dense)         (None, 256)           62980352  

dropout (Dropout)     (None, 256)           0  

dense_1 (Dense)       (None, 3)             771  

-----  

Total params: 62,990,275  

Trainable params: 62,990,275  

Non-trainable params: 0

```

Hiperparametros

```

1 #Para entrenar nuestro algoritmo havemos de fit (Que significa cada variable se describe en la parte de parametros)
2 history = cnn.fit(x_train,y_train, epochs = epochs, validation_split = 0.2)

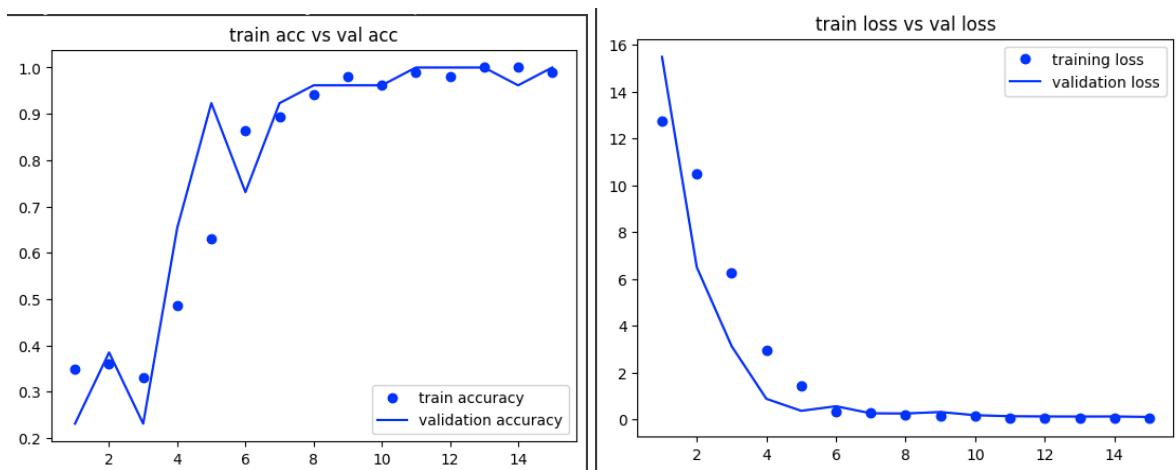
```

Donde no se escribieron los pasos de época y de validación, para que la misma función eligiera los mejores resultados para estos

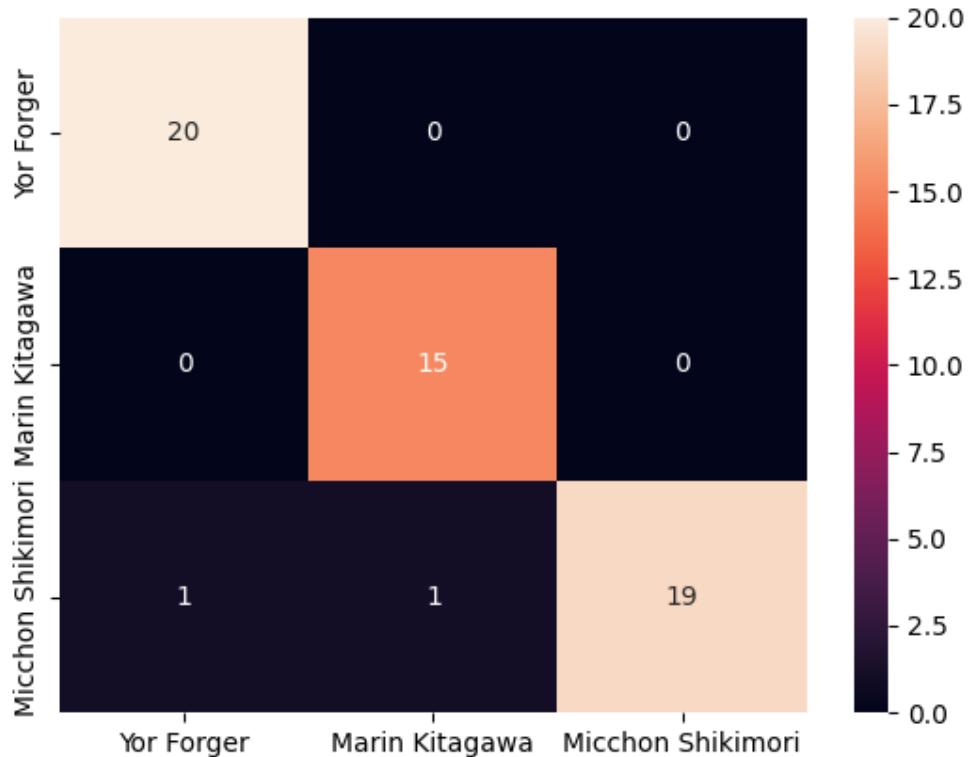
Entrenamiento

```
Epoch 1/15
4/4 [=====] - 16s 4s/step - loss: 12.7469 - accuracy: 0.3495 - val_loss: 15.4835 - val_accuracy: 0.2308
Epoch 2/15
4/4 [=====] - 11s 2s/step - loss: 10.4762 - accuracy: 0.3592 - val_loss: 6.4909 - val_accuracy: 0.3846
Epoch 3/15
4/4 [=====] - 9s 2s/step - loss: 6.2541 - accuracy: 0.3301 - val_loss: 3.1224 - val_accuracy: 0.2308
Epoch 4/15
4/4 [=====] - 9s 2s/step - loss: 2.9517 - accuracy: 0.4854 - val_loss: 0.8693 - val_accuracy: 0.6538
Epoch 5/15
4/4 [=====] - 9s 2s/step - loss: 1.4145 - accuracy: 0.6311 - val_loss: 0.3561 - val_accuracy: 0.9231
Epoch 6/15
4/4 [=====] - 9s 2s/step - loss: 0.3353 - accuracy: 0.8641 - val_loss: 0.5523 - val_accuracy: 0.7308
Epoch 7/15
4/4 [=====] - 9s 2s/step - loss: 0.2969 - accuracy: 0.8932 - val_loss: 0.2494 - val_accuracy: 0.9231
Epoch 8/15
4/4 [=====] - 9s 2s/step - loss: 0.1973 - accuracy: 0.9417 - val_loss: 0.2412 - val_accuracy: 0.9615
Epoch 9/15
4/4 [=====] - 9s 2s/step - loss: 0.1219 - accuracy: 0.9806 - val_loss: 0.3042 - val_accuracy: 0.9615
Epoch 10/15
4/4 [=====] - 9s 2s/step - loss: 0.1344 - accuracy: 0.9612 - val_loss: 0.1681 - val_accuracy: 0.9615
Epoch 11/15
4/4 [=====] - 9s 2s/step - loss: 0.0520 - accuracy: 0.9903 - val_loss: 0.1246 - val_accuracy: 1.0000
Epoch 12/15
4/4 [=====] - 9s 2s/step - loss: 0.0598 - accuracy: 0.9806 - val_loss: 0.1147 - val_accuracy: 1.0000
Epoch 13/15
4/4 [=====] - 9s 2s/step - loss: 0.0333 - accuracy: 1.0000 - val_loss: 0.1106 - val_accuracy: 1.0000
Epoch 14/15
4/4 [=====] - 9s 2s/step - loss: 0.0224 - accuracy: 1.0000 - val_loss: 0.1142 - val_accuracy: 0.9615
Epoch 15/15
4/4 [=====] - 9s 2s/step - loss: 0.0448 - accuracy: 0.9903 - val_loss: 0.0941 - val_accuracy: 1.0000
```

Train vs Val



Matriz de confusión

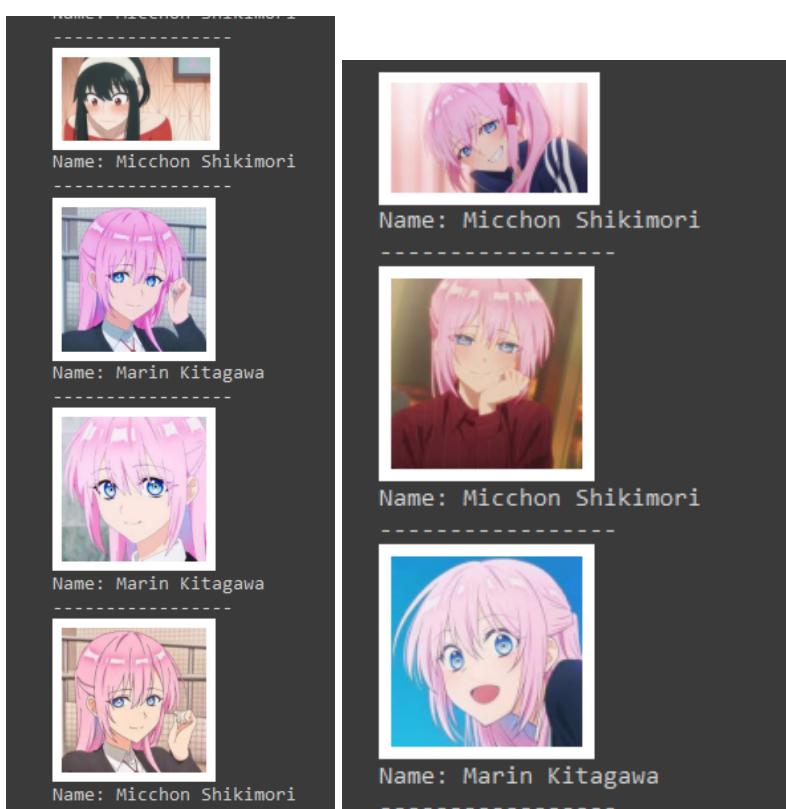
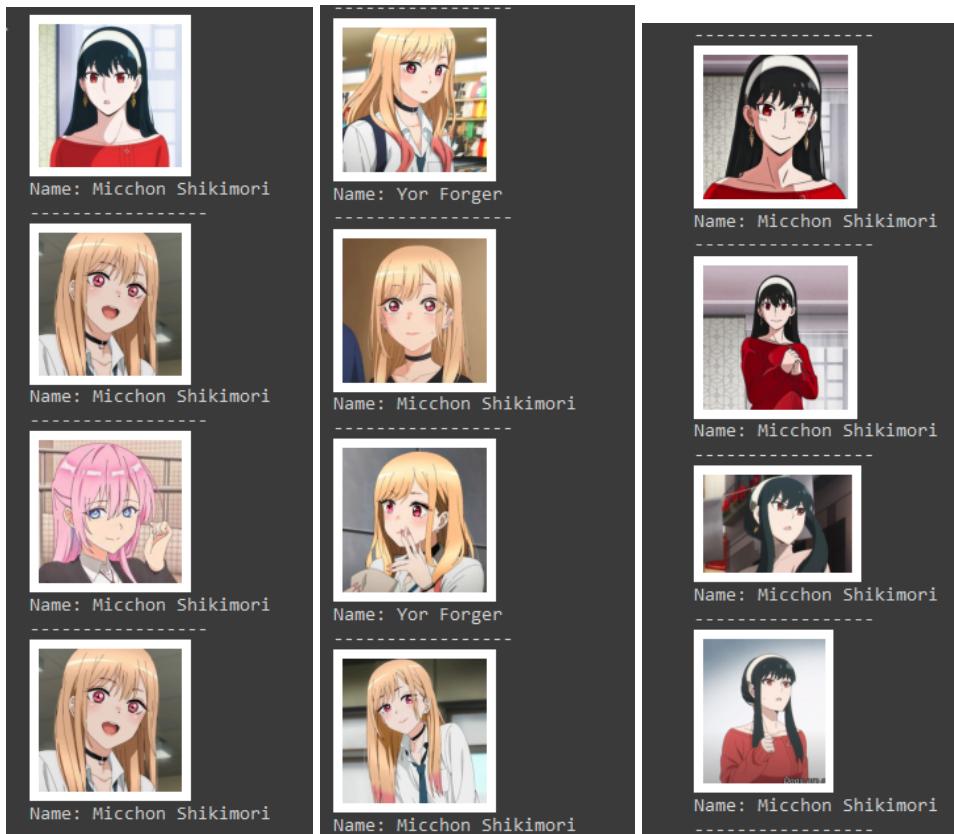


Loss acc - Models acc

```
Test loss Model 2: 0.11337532103061676
Test accuracy Model 2: 0.9642857313156128
```

A pesar de los resultados obtenidos si prestamos atención se puede notar que existe cierto overfitting en nuestro modelo, esto puede ser debido a que el database que estoy utilizando es demasiado pequeño, y el modelo no cuenta con suficientes imágenes para trabajar

Prueba del modelo



¿Qué se podría mejorar?

En cuanto a los resultados obtenidos de las predicciones del modelo podemos notar que no es tan exacta como me gustaría, esto se puede deber a varios factores siendo uno de estos el hecho de que la database con la que se está trabajando es demasiado pequeña en comparación a otras por lo que el modelo no cuenta con la suficiente alimentación como para poder desarrollar una buena predicción, en segundo lugar parece que tiene a tomar más en cuenta las forma que identifica dentro de las imágenes, y siendo que los personajes cuentan con un estilo de dibujo bastante parecido entre sí, el modelo tiende a cometer errores a la hora de clasificarlas, siendo que por esto creo que sería una buena idea aumentar el tamaño del dataframe a una cantidad mayora para que este pueda aumentar su nivel de predicción, así no sería interesante el probar con personajes que sean mucho más distintos entre ellos y así observar si en efecto el modelo puede identificar los personajes entre sí.

Modelo 2

Este Modelo 2, fue el primer modelo que realice, siento que este hice uso de transfer learning con ayuda de VGG16

```
#Modelo anterior con Transfer learning

conv_base= VGG16(weights='imagenet',
                  include_top=False,
                  input_shape = (height, width,3))

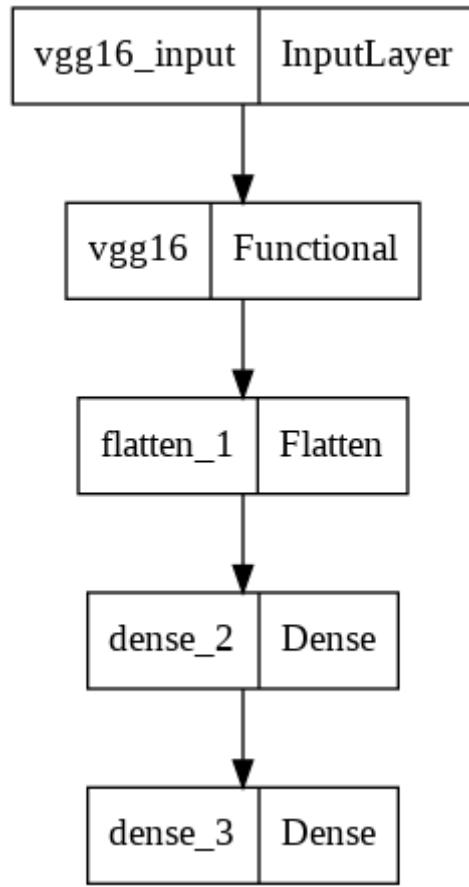
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(3,activation='sigmoid'))

model.summary()

conv_base.trainable = False

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=2e-5),
              metrics=[ 'acc'])
```

Estructura de red



Hiperparametros

```
1 history2 = model.fit(x_train,y_train,
2                      steps_per_epoch = 10,
3                      epochs = 10,
4                      validation_split = 0.2,
5                      validation_steps = 5)
6
```

Entrenamiento

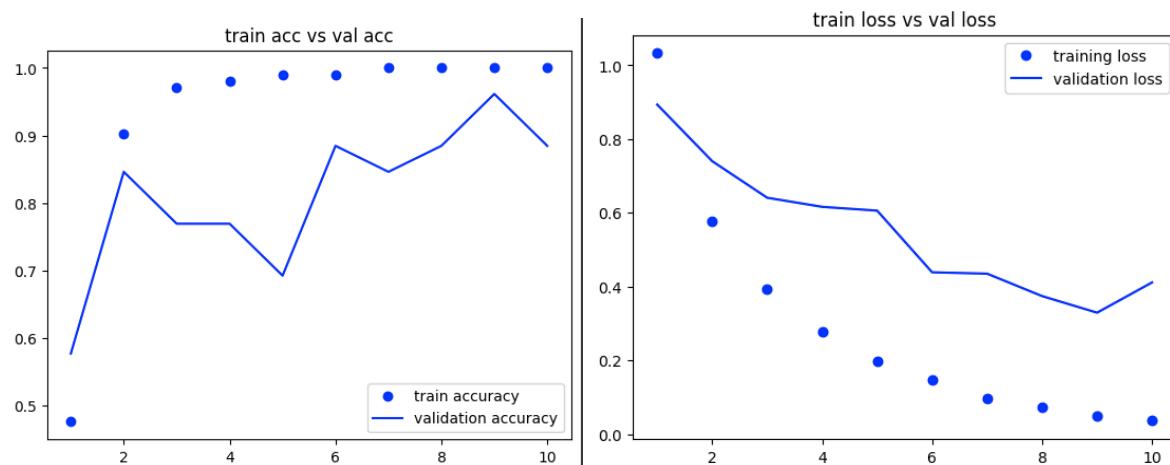
```

→ Epoch 1/10
10/10 [=====] - 85s 9s/step - loss: 1.0325 - acc: 0.4757 - val_loss: 0.8935 - val_acc: 0.5769
Epoch 2/10
10/10 [=====] - 77s 8s/step - loss: 0.5763 - acc: 0.9029 - val_loss: 0.7401 - val_acc: 0.8462
Epoch 3/10
10/10 [=====] - 72s 7s/step - loss: 0.3942 - acc: 0.9709 - val_loss: 0.6409 - val_acc: 0.7692
Epoch 4/10
10/10 [=====] - 78s 8s/step - loss: 0.2765 - acc: 0.9806 - val_loss: 0.6162 - val_acc: 0.7692
Epoch 5/10
10/10 [=====] - 72s 7s/step - loss: 0.1963 - acc: 0.9903 - val_loss: 0.6059 - val_acc: 0.6923
Epoch 6/10
10/10 [=====] - 72s 7s/step - loss: 0.1469 - acc: 0.9903 - val_loss: 0.4386 - val_acc: 0.8846
Epoch 7/10
10/10 [=====] - 78s 8s/step - loss: 0.0974 - acc: 1.0000 - val_loss: 0.4349 - val_acc: 0.8462
Epoch 8/10
10/10 [=====] - 72s 7s/step - loss: 0.0719 - acc: 1.0000 - val_loss: 0.3741 - val_acc: 0.8846
Epoch 9/10
10/10 [=====] - 72s 7s/step - loss: 0.0484 - acc: 1.0000 - val_loss: 0.3291 - val_acc: 0.9615
Epoch 10/10
10/10 [=====] - 82s 8s/step - loss: 0.0357 - acc: 1.0000 - val_loss: 0.4112 - val_acc: 0.8846

```

Como podemos observar en este caso acc, se vuelve 1 desde muy temprano en las épocas, esto significa que hay un overfitting bastante notable en este modelo.

Train vs Val



Loss acc - Models acc

```

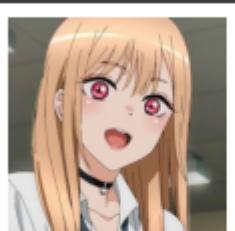
Test loss Model 2: 0.1983238011598587
Test accuracy Model 2: 0.9642857313156128

```

Prueba del modelo



Name: Micchon Shikimori



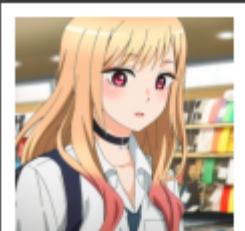
Name: Yor Forger



Name: Marin Kitagawa



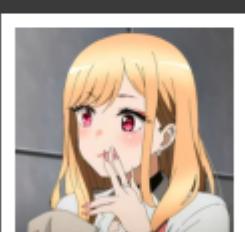
Name: Yor Forger



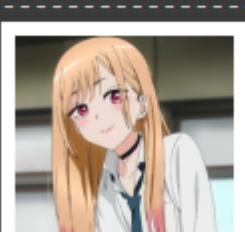
Name: Yor Forger



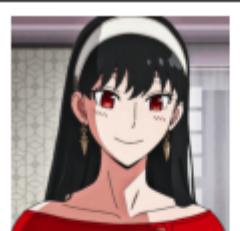
Name: Yor Forger



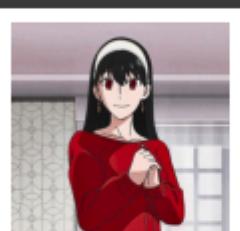
Name: Yor Forger



Name: Yor Forger



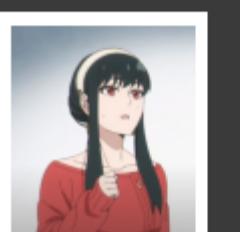
Name: Yor Forger



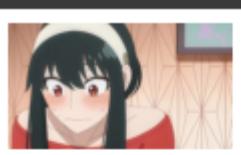
Name: Yor Forger



Name: Micchon Shikimori



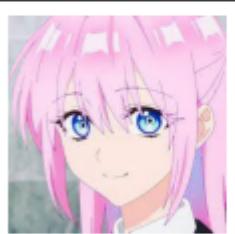
Name: Micchon Shikimori



Name: Micchon Shikimori



Name: Marin Kitagawa



Name: Marin Kitagawa



Name: Marin Kitagawa



Name: Marin Kitagawa



Name: Marin Kitagawa



Name: Marin Kitagawa

¿Qué se podría mejorar?

En este caso el overfitting que existía era muy obvio y a pesar de que se intentó trabajar con distintos hiperparametros, el overfitting siempre era bastante notorio, por lo que este primer modelo llamado 'Modelo 2' fue cambiado a un modelo diferente que vendría a ser el 'Modelo 1'

Se cree que la razón del overfitting es como el database es demasiado pequeño como para poder trabajar con VG16