



Face Mask Detection

Mauricio Rivera
Braulio Quiñe
Bruno Mixán
Hans Burmester

Índice

- Introducción
- Consideraciones y preprocesamiento
- Modelo implementado
 - DNN
 - CNN
 - AlexNet
 - Lenet
 - VGG-16
 - ResNet
- Resultados
 - Comparación de indicadores
 - Predicción
- Conclusiones y observaciones



Introducción

Problemática

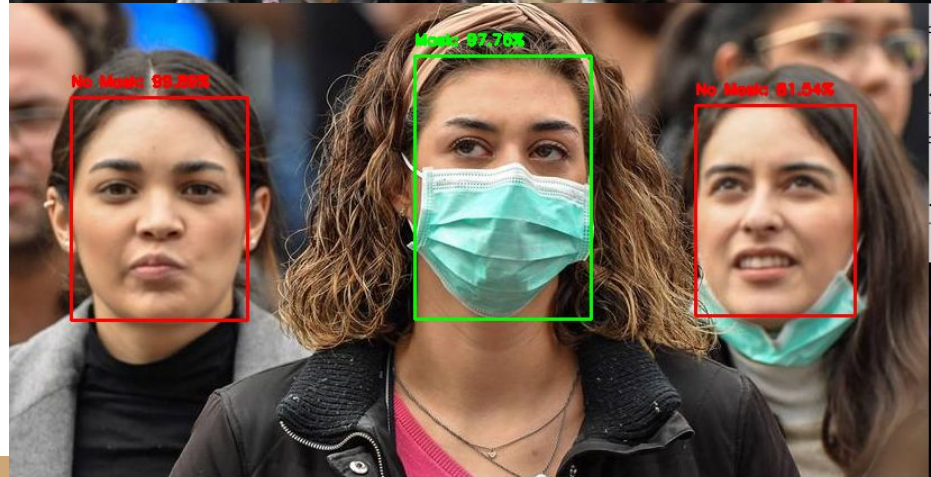



Objetivos

- *Detección de rostros con/sin mascarilla con un accuracy mínimo de 70%
- * Resolver un problema de clasificación binaria
- * Comparar arquitecturas de redes (un total de 4 métodos CNN) y escoger la mejor

Base de datos:

<https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset>





Consideraciones y pre-procesamiento



Es importante considerar que las imágenes utilizadas fueron procesadas con anterioridad para que las imágenes sean cortadas y se vea, más que nada la cara de las personas



Imágenes a escala de grises de 256 x 256



Reducción de la data a procesar

```
#Train: Usando solo 1600 imágenes: 800 de with y 800 de without  
filenames_train = filenames_train[ 0:800]+filenames_train[ 9199:9999]  
GT_ids_train = GT_ids_train[ 0:800]+GT_ids_train[ 9199:9999]
```

```
#Valid: Usando solo 400 imágenes: 200 de with y 200 de without  
filenames_valid = filenames_valid[ 0:200]+filenames_valid[ 599:799]  
GT_ids_valid = GT_ids_valid[ 0:200]+GT_ids_valid[ 599:799]
```

Reshape y normalización

```
# Aumentar 1 dimension para que esté en formato que acepta Keras (escala a grises)
X_train = X_train.reshape(ntrain, npx, npx, 1)
X_valid = X_valid.reshape(nvalid, npx, npx, 1)
X_test = X_test.reshape(ntest, npx, npx, 1)
```

```
print("Tamaño modificado del conjunto de entrenamiento:", X_train.shape)
print("Tamaño modificado del conjunto de validacion:", X_valid.shape)
print("Tamaño modificado del conjunto de prueba:", X_test.shape)
```

```
Tamaño modificado del conjunto de entrenamiento: (1600, 256, 256, 1)
Tamaño modificado del conjunto de validacion: (400, 256, 256, 1)
Tamaño modificado del conjunto de prueba: (992, 256, 256, 1)
```

```
#Normalizando
```

```
X_train = X_train/255.          #el . es para que sea float
X_valid = X_valid/255.
X_test = X_test/255.
```

Uso de parámetros de callback

```
filepath = './my_best_model.epoch{epoch:02d}-loss{val_loss:.2f}.hdf5' #para almacenar cada mejor modelo encontrado con con
cada iteracion

cp = ModelCheckpoint(filepath=filepath,
                    monitor='val_loss',
                    verbose=1,
                    save_best_only=True,
                    mode='min')

#el checkpoint(cp) guardará los parámetros del modelo que generen el mínimo de 'val_loss' y se quedará con esos hasta que aparezca
un menor a loss

es = EarlyStopping(monitor='val_loss',
                  patience=10,
                  mode='min',
                  verbose=0,
                  restore_best_weights=True)

# si "es" detecta 10 veces seguidas que el 'val_loss' está subiendo(malo) entonces parará las iteraciones

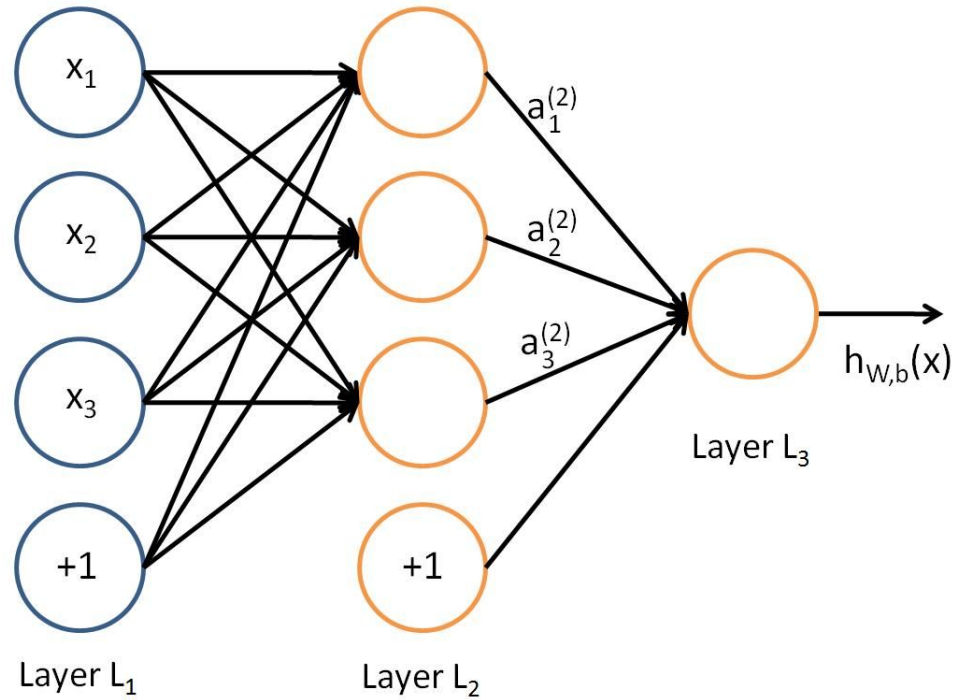
pl = ReduceLROnPlateau(monitor='val_loss',
                      factor=0.2,
                      patience=5,
                      verbose=0,
                      mode='min')

#reduce(multiplica) al 'learning_rate' en un factor de 0.2 cada 5 veces seguidas que el val_loss se reduzca
```




Modelo implementado

Red DNN



DNN

DNN parametrizado

```
model.add(Flatten())  
model.add(Dense(4096))  
model.add(BatchNormalization())  
model.add(Activation('swish'))  
model.add(Dropout(0.5))
```

```
model.add(Dense(10))  
model.add(BatchNormalization())  
model.add(Activation('swish'))  
model.add(Dropout(0.5))
```

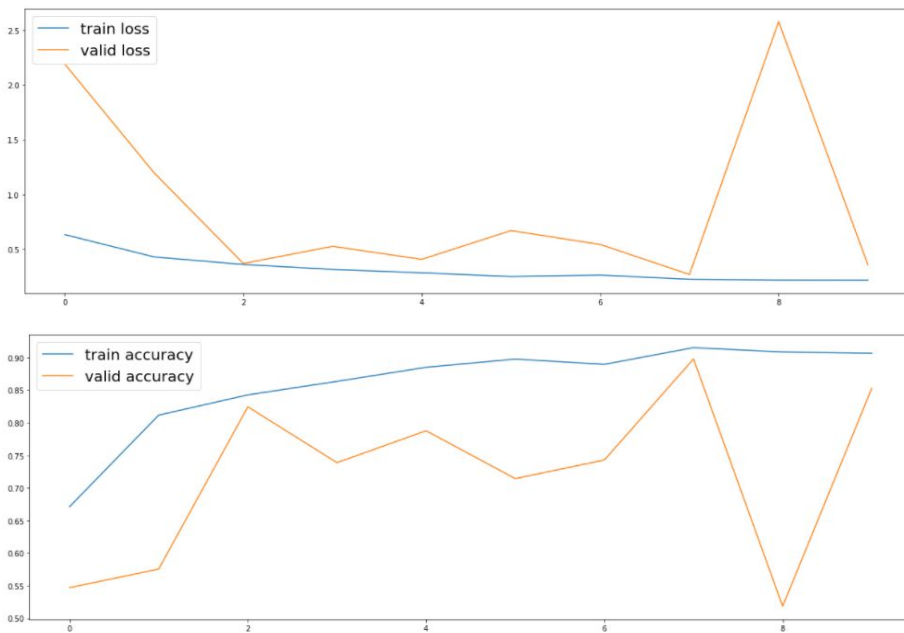
```
model.add(Dense(1))  
model.add(Activation('sigmoid'))
```

DNN simple

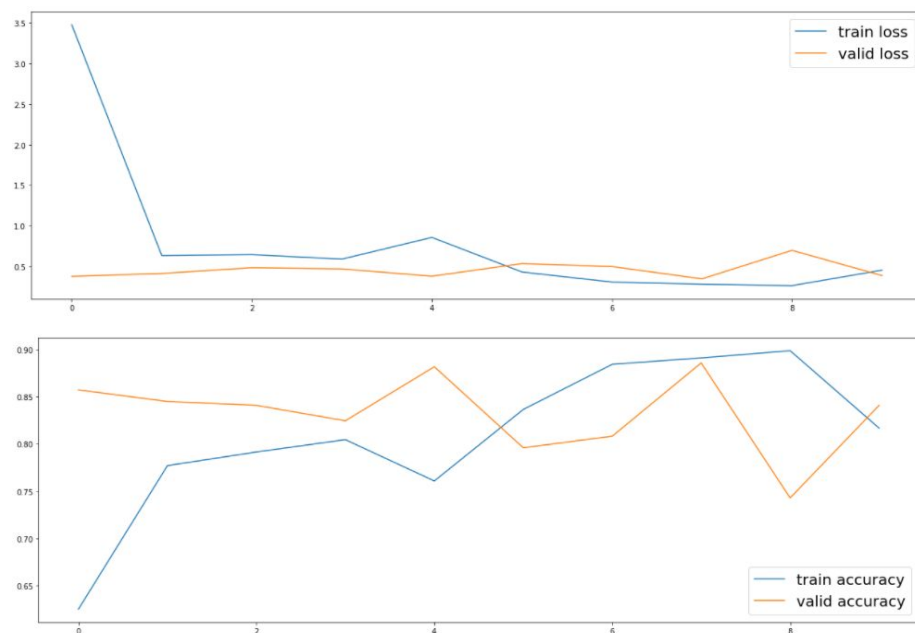
```
model.add(Flatten())  
model.add(Dense(4096, activation='relu'))  
model.add(Dense(10, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))
```


DNN

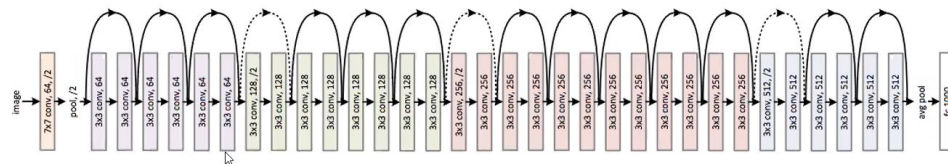
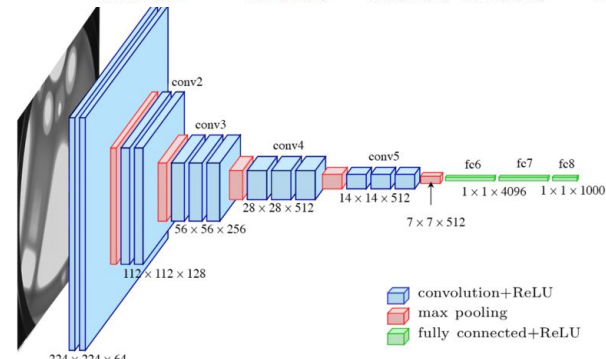
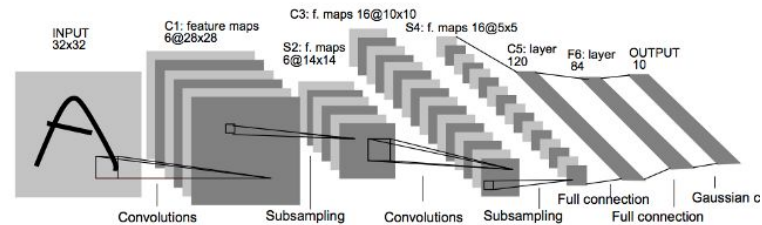
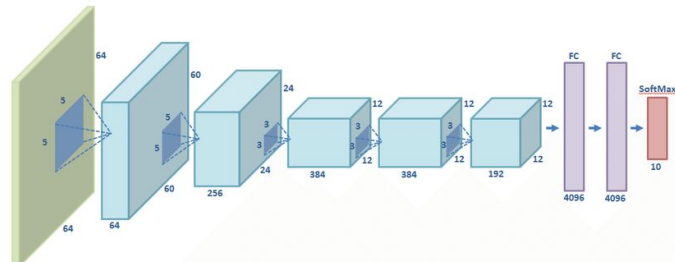
DNN parametrizado



DNN simple



CNN



CNN - AlexNet

```
model = models.Sequential()

model.add(layers.experimental.preprocessing.Resizing(256, 256, input_shape=X_train.shape[1:]))

model.add(layers.Conv2D(96, 11, strides=4, padding='same')) # tamaño = 64x64x96
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D((3,3), strides=2)) # tamaño = 31x31x96

model.add(layers.Conv2D(256, 5, strides=4, padding='same')) # tamaño = 8x8x256
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D((3,3), strides=2)) # tamaño = 3x3x256

model.add(layers.Conv2D(384, 3, strides=4, padding='same')) # tamaño = 1x1x384
model.add(layers.Activation('relu'))

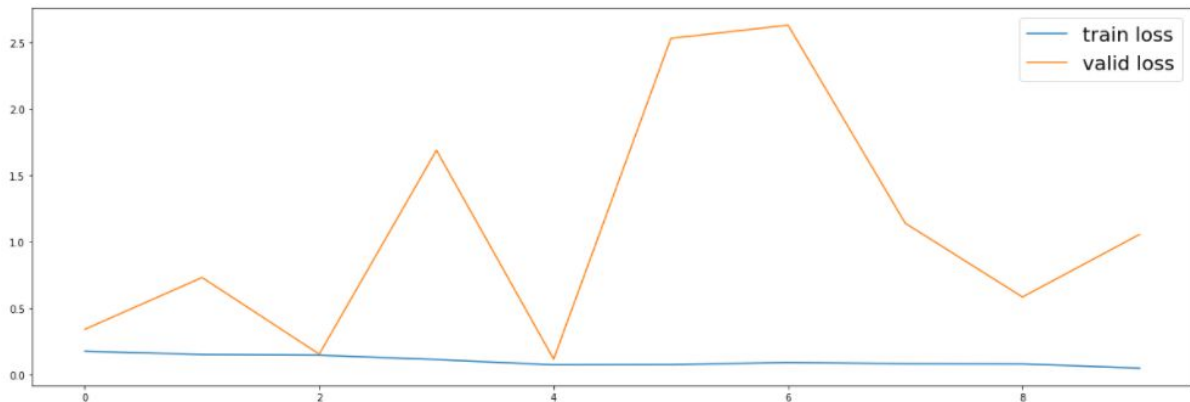
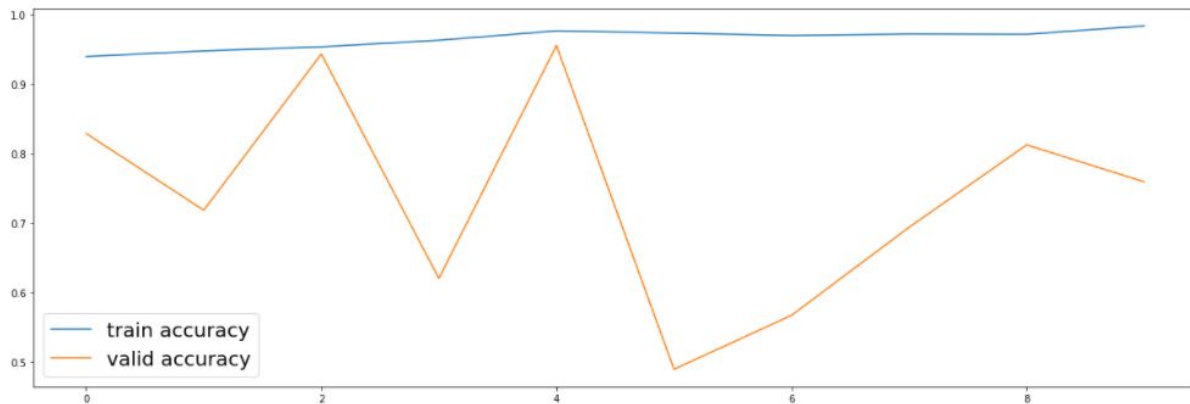
model.add(layers.Conv2D(256, 3, strides=4, padding='same')) # tamaño = 1x1x256
model.add(layers.Activation('relu'))
model.add(layers.Flatten())
```


CNN - AlexNet

```
=====
Total params: 2,505,281
Trainable params: 2,504,385
Non-trainable params: 896
```

Se usó 2697 imágenes para
no gastar la memoria RAM

7min - 10 épocas



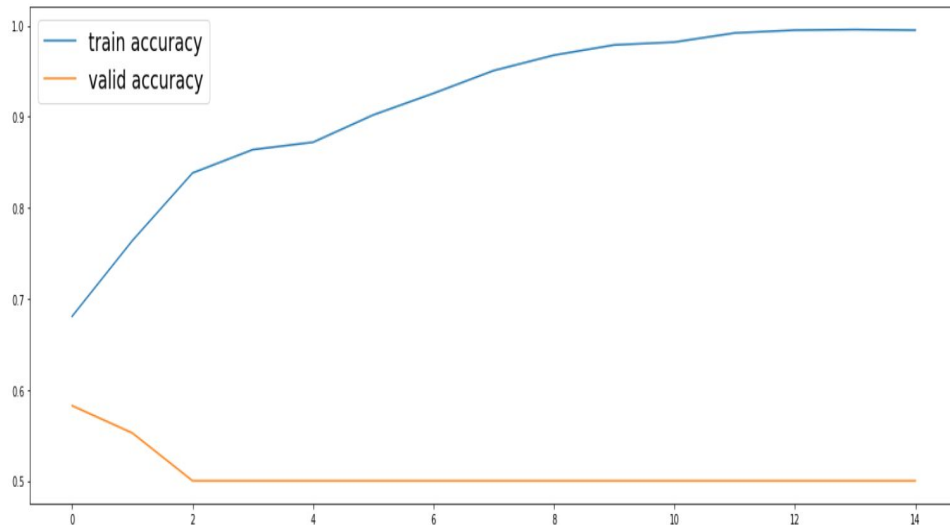
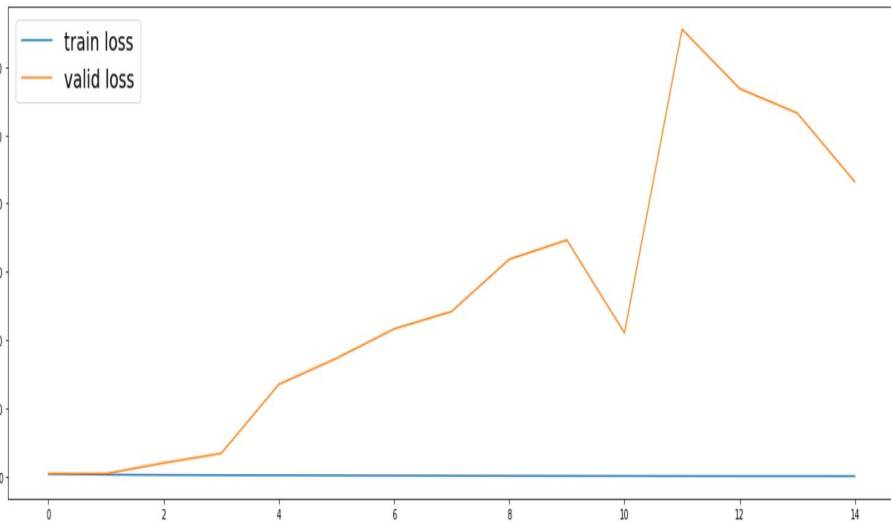
CNN - LeNet

```
model = models.Sequential()  
model.add(layers.Conv2D(6, 5, activation='tanh', input_shape=X_train.shape[1:]))  
model.add(layers.AveragePooling2D(2))  
model.add(layers.Activation('sigmoid'))  
model.add(layers.Conv2D(16, 5, activation='tanh'))  
model.add(layers.AveragePooling2D(2))  
model.add(layers.Activation('sigmoid'))  
model.add(layers.Conv2D(120, 5, activation='tanh'))
```

CNN - LeNet

=====
Total params: 99,894,789
Trainable params: 99,894,021
Non-trainable params: 768
=====

```
Epoch 14/15  
16/16 [=====] - ETA: 0s - loss: 0.0326 - accuracy: 0.9956 - auc: 0.9999  
Epoch 00014: val_loss did not improve from 0.76050  
16/16 [=====] - 5s 319ms/step - loss: 0.0326 - accuracy: 0.9956 - auc: 0.9999 - val_loss: 106.5219 - val_accuracy: 0.5000 - val_auc: 0.5000 - lr: 2.0000e-04  
Epoch 15/15  
16/16 [=====] - ETA: 0s - loss: 0.0298 - accuracy: 0.9950 - auc: 0.9999  
Epoch 00015: val_loss did not improve from 0.76050  
16/16 [=====] - 5s 316ms/step - loss: 0.0298 - accuracy: 0.9950 - auc: 0.9999 - val_loss: 86.3609 - val_accuracy: 0.5000 - val_auc: 0.5000 - lr: 2.0000e-04
```



CNN - VGG-16

```
model = Sequential()
model.add(Conv2D(input_shape=(256,256,1),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=128,kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
model.add(Conv2D(filters=256,kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256,kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=512,kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

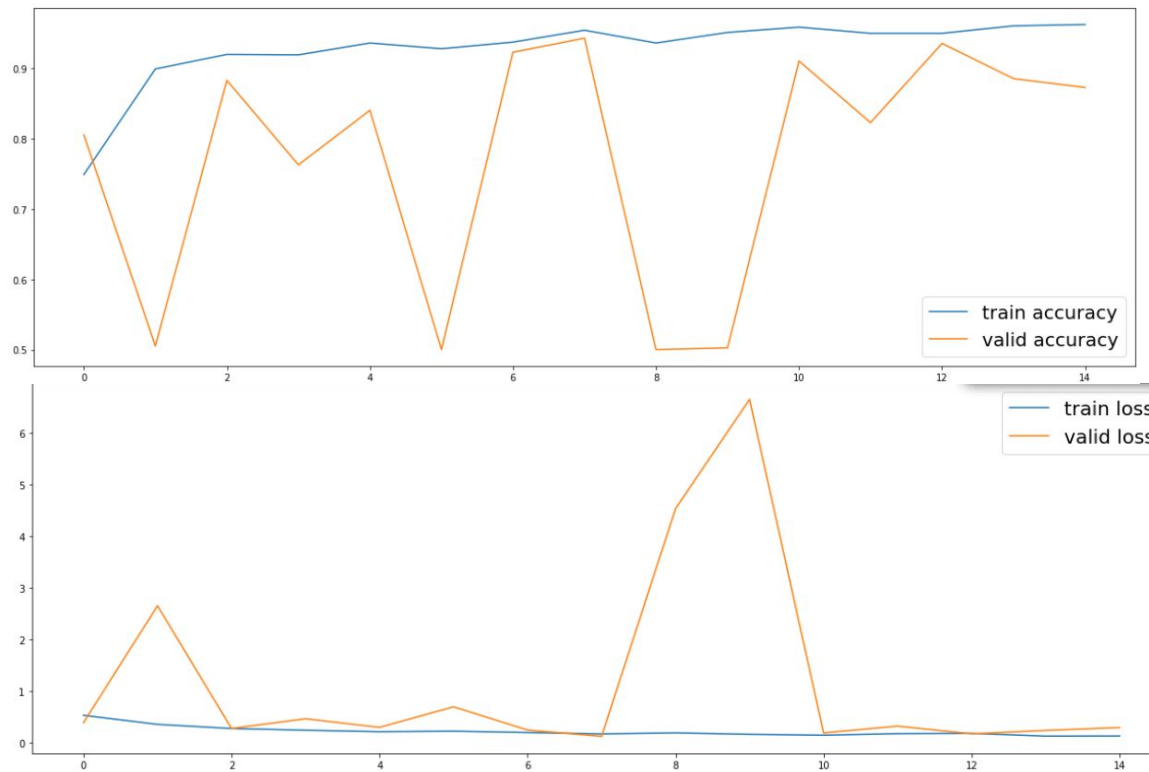
```
model.add(Conv2D(filters=512,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=512,kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```


CNN - VGG-16

20min - 15 épocas

accuracy_train → 96.19 %
accuracy_test → 87.25%

=====
Total params: 148,993,917
Trainable params: 148,985,705
Non-trainable params: 8,212
=====



CNN - ResNet (50 layers)

```
def ResNet50(input_shape=(256, 256, 1), classes=1):
```

```
def convolutional_block(X, f, filters, stage, block, s = 2):
```

```
def identity_block(X, f, filters, stage, block):
```

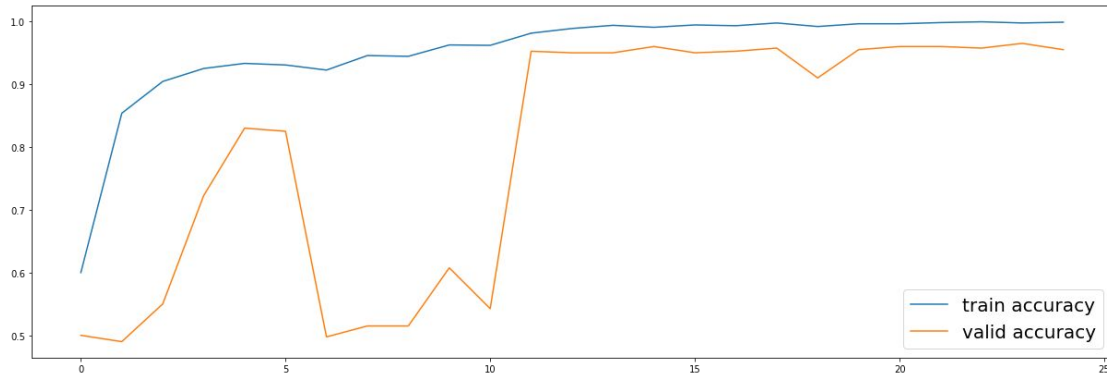
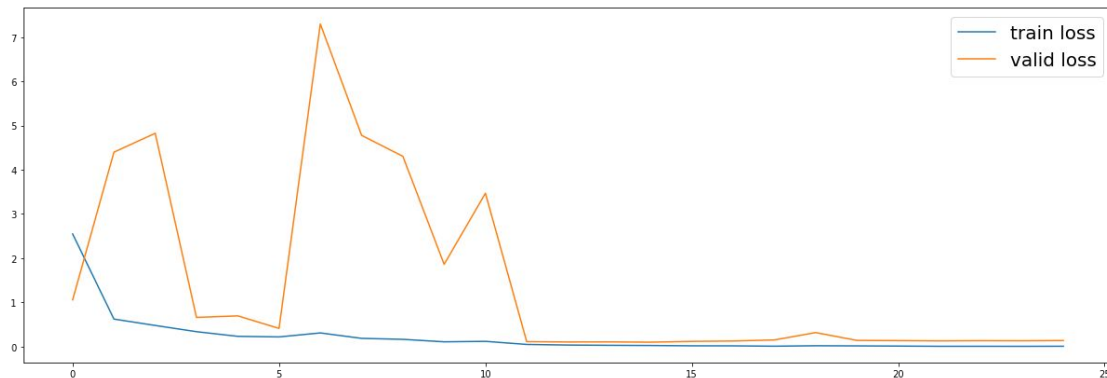
- Zero-padding pads the input with a pad of (3,3)
- Stage 1:
 - The 2D Convolution has 64 filters of shape (7,7) and uses a stride of (2,2). Its name is "conv1".
 - BatchNorm is applied to the channels axis of the input.
 - MaxPooling uses a (3,3) window and a (2,2) stride.
- Stage 2:
 - The convolutional block uses three set of filters of size [64,64,256], "f" is 3, "s" is 1 and the block is "a".
 - The 2 identity blocks use three set of filters of size [64,64,256], "f" is 3 and the blocks are "b" and "c".
- Stage 3:
 - The convolutional block uses three set of filters of size [128,128,512], "f" is 3, "s" is 2 and the block is "a".
 - The 3 identity blocks use three set of filters of size [128,128,512], "f" is 3 and the blocks are "b", "c" and "d".
- Stage 4:
 - The convolutional block uses three set of filters of size [256, 256, 1024], "f" is 3, "s" is 2 and the block is "a".
 - The 5 identity blocks use three set of filters of size [256, 256, 1024], "f" is 3 and the blocks are "b", "c", "d", "e" and "f".
- Stage 5:
 - The convolutional block uses three set of filters of size [512, 512, 2048], "f" is 3, "s" is 2 and the block is "a".
 - The 2 identity blocks use three set of filters of size [512, 512, 2048], "f" is 3 and the blocks are "b" and "c".

CNN - ResNet (50 layers)

```
=====
Total params: 23,614,209
Trainable params: 23,561,089
Non-trainable params: 53,120
=====
```

**17 minutos para
25 épocas**

```
X_train.shape, X_valid.shape
((1600, 256, 256, 1), (400, 256, 256, 1))
```





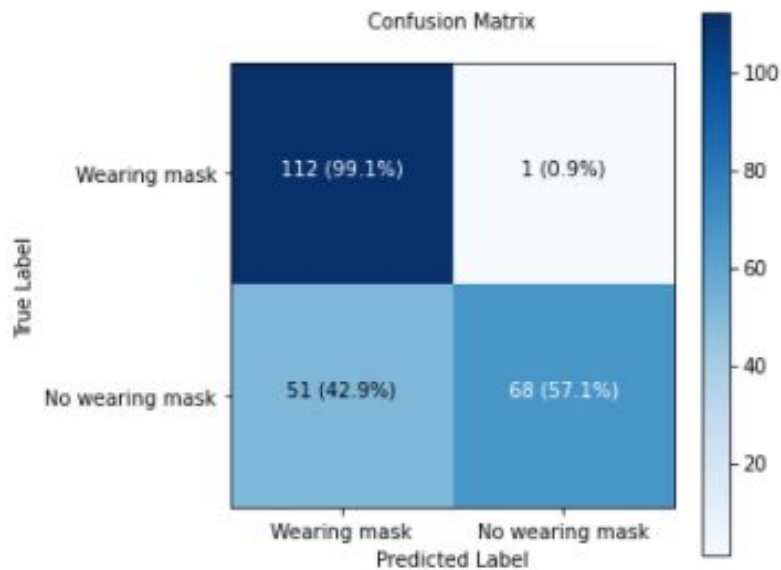
Resultados

Comparación de accuracy y loss

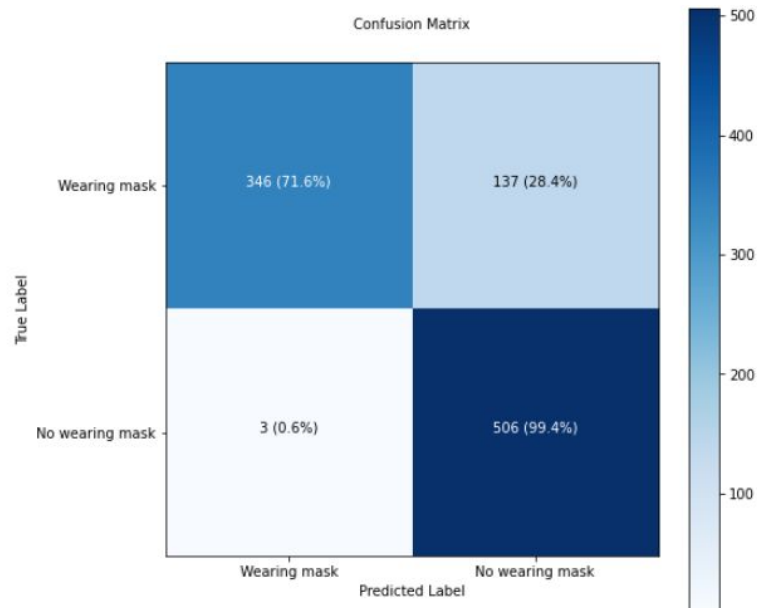
	train_accuracy	valid_accuracy	train_loss	valid_loss	#parámetros
AlexNet	98,33%	75,92%	4.97 %	105.50%	2,505,281
LeNet	99.5%	50%	2.98%	86.39%	99,894,021
VGG 16	96.19 %	87.25%	12.48%	28.94%	148,992,765
ResNet	99.87%	95.50%	0.8%	13.67%	23,614,209

Predicción con modelos

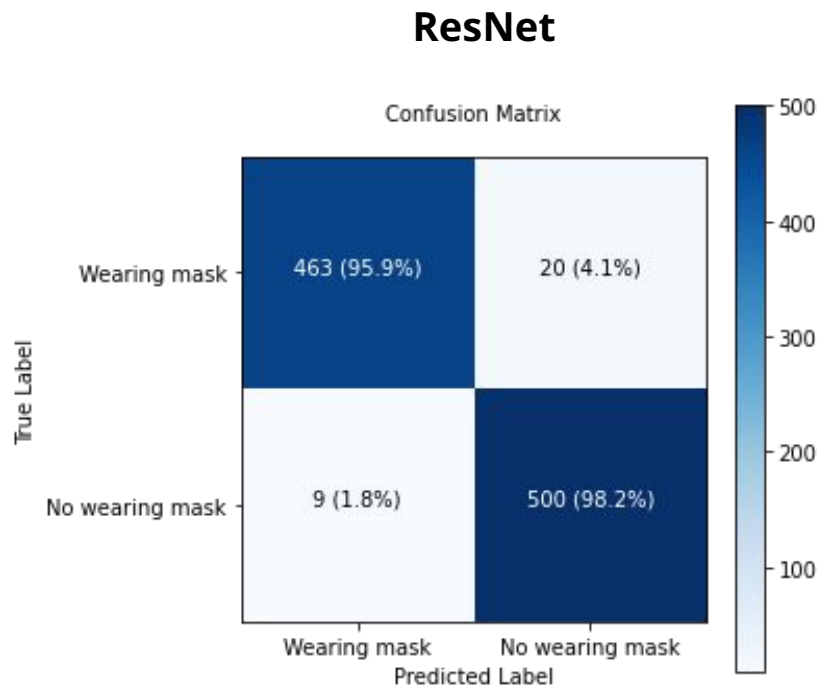
AlexNet



VGG-16



Predicción con modelos





Conclusiones y observaciones

Observaciones

- El uso del GPU, en cuanto a rapidez, y la memoria RAM, en cuanto a cantidad de datos posibles de procesar, obtuvieron mayor relevancia para el desarrollo de nuestro proyecto
- Se observó que es posible realizar el clasificador utilizando únicamente DNN, no obstante dada la enorme cantidad de datos, el programa se reiniciaba en la mayoría de casos
- La cantidad de parámetros afecta en la rapidez de procesamiento de cada época considerablemente.
- La cantidad de parámetros usados está relacionada con el mejor comportamiento del modelo, no obstante, la mayor cantidad de estos no implica ser el mejor de los modelos

Conclusiones

- Se obtuvieron 4 diferentes modelos de red capaces de cumplir con el objetivo de identificar personas con y sin mascarilla obteniendo los siguientes rangos de accuracy entre los cuatro
 - `train_accuracy` = [96.19; 99.87]%
 - `valid_accuracy` = [50; 95.5]%
- La CNN ResNet fue la de mayor precisión, tanto para los datos de validación como para los del entrenamiento. Esto es debido a la complejidad de la convolución en la red neuronal. (Backward propagation en bloques identidad y convolucionales).
- El preprocesamiento correcto de las imágenes para todos los métodos fue independiente a los modelos utilizados, variando únicamente en la cantidad de datos utilizados en la red neuronal. (Eficiencia)
- La historia evolutiva en el ploteo de validación y entrenamiento de la data nos muestra el avance y nos brinda un feedback de la confiabilidad de la red neuronal.
- A mayor cantidad de épocas, mayor estabilización en la exactitud de los datos de validación y en la función de costo