Práctica de Detección de Rostros Reales y Falsos con Redes Neuronales Convolucionales (CNN) en PyTorch

Memoria - Practica 1 - AA2 - 2023/24 - Giovanni Sgambato, Agata Cavigioli

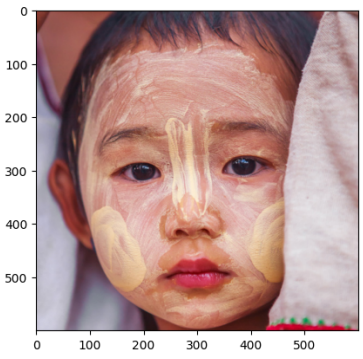
Introducción

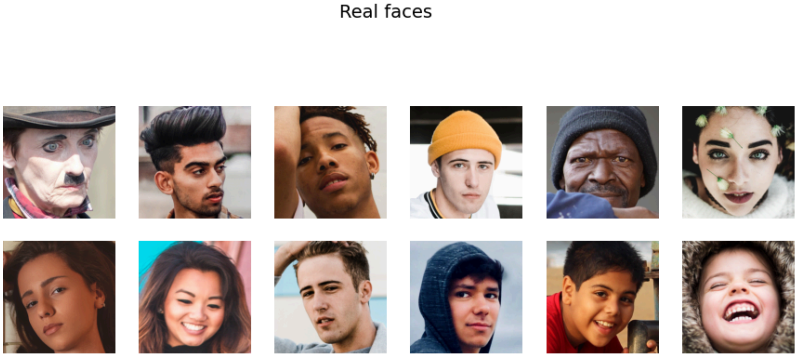
La autenticación facial ha adquirido un papel fundamental, siendo empleada en dispositivos móviles, sistemas de seguridad, biometría y aplicaciones de identificación personal, para asegurar la **integridad** de sistemas y la **privacidad** de los individuos, por esto es necesidad prioritaria de desarrollar modelos de detección precisos y confiables.

En este proyecto, hemos desarrollado una **red neuronal convolucional** (en varias configuraciones) con la capacidad de **clasificar** entre imágenes de **rostros reales** y **falsificados por expertos**.
Se realizaron 2 intentos de **modelo** con diferentes **arquitecturas**, en particular, con un número diferente de capas convolucionales, y varias pruebas de transformaciones en fase de **pre-procesamiento**, para luego evaluar las diferencias de **rendimiento**.
Para este ejercicio, se utilizò un notebook en google **colab** con un entorno python y runtime **T4 gpu**.
La **metodología aplicada** se compone de diferentes etapas:

1) Pre-procesamiento de Datos

Como punto de partida se realizó una primera fase de **pre-procesamiento** del conjunto de datos, que luego pudo utilizarse para realizar todos los pasos posteriores y las distintas alternativas de arquitectura propuestas.
En esta **fase inicial**, se **descarga** y **descomprime** el **conjunto de datos**, para poder utilizar las imágenes de entrenamiento y evaluación del modelo.
Dividir los datos en conjuntos de entrenamiento y prueba, es una práctica esencial para poder entrenar el modelo y medir su rendimiento de manera efectiva.
Uno de los aspectos más cruciales es la **normalización** de las imágenes y su **conversión** en **tensores**, que se aplica para asegurarse de que todas las imágenes tengan el mismo **formato**, lo que facilita que el modelo las procese de manera uniforme y a mejorar la **eficiencia**.
También en una fase preliminar, se implementó un **dataloader** para imprimir algunas muestras de imágenes por **lotes**.





2) Arquitectura del Modelo

Se procede a diseñar unas **arquitecturas** de **Redes** Neuronal Convolucional (CNN), responsable de clasificar las imágenes, a traves proceso experimental que involucra la experimentación de **diversas configuraciones** con el objetivo de determinar la que mejor performe.

3) Entrenamiento

En este step, se elige una **función de pérdida** y un **optimizador,** y se realiza el training del modelo utilizando el conjunto de entrenamiento, ajustando sus parámetros para hacer predicciones más precisas.

4) Evaluacion

La evaluación del modelo implica medir su performance sobre en un conjunto de datos independiente utilizando algunas **métricas** como la precisión (Accuracy, recall, F1-score, support), y ploteando una **matriz de confusión** o algunos gráficos de **curvas caracteristicas**.

Regularizacion

En todas la fases, para intentar **minimizar el sobreajuste**, haciendo que nuestros algoritmos generalicen bien y la red performe correctamente, hemos utilizado varias técnicas de regularizacion que se aplican al modelo o los datos, como:
DropOut, arreglando parametros como el **decaimiento de pesos**, **batch normalization**, y **data augmentation**.

Pruebas de Modelos

Modelo 1

Pre-processing

Despues de descargar el dataset en una carpeta GoogleDrive se ha dividido los datos en conjuntos de entrenamiento y prueba, ejecutando un proceso **normalizacion** de las imágenes y conversión en **tensores**, con los siguiente parametros:
- **Tamaño** de las imágenes: 128x128 pıxeles.
- **Transformaciones** aplicadas: Resize, Normalize, ToTensor.
Se ha realizado un primer modelo aplicando las solas transformaciones

```
#transforma size de imagenes, convierte en tensor, normaliza
data_transforms = transforms.Compose([
    transforms.Resize(INITIAL_SIZE_OF_IMAGES),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
```

- **División** del conjunto de datos en entrenamiento (80%), prueba (10%) y validacion (10%).

Arquitectura

Se ha implementado la siguiente **arquitectura:**

• **Capa de convolución**: 3 canales de entrada, 32 canales de salida, kernel de 3x3, stride de 1, padding de 1.

• **Capa de convolución**: 32 canales de entrada, 64 canales de salida, kernel de 3x3, stride de 1, padding de 1.

• **Capa de pooling** (MaxPool2d) con kernel de 2x2 y stride de 2.

• **Batch Normalization** después de la segunda capa de convolución.

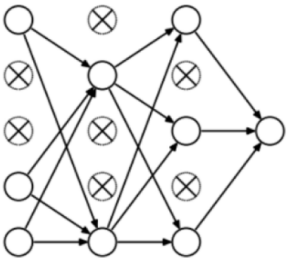
• **Capa completamente conectada** (fc1) con 64x64x64 nodos de entrada y 128 nodos de salida.

• **Capa completamente conectada** (fc2) con 128 nodos de entrada y 2 nodos de salida (clasificaci´on binaria).

Dropout

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 256, 256]             896
         MaxPool2d-2         [-1, 32, 128, 128]               0
            Conv2d-3         [-1, 64, 128, 128]          18,496
         MaxPool2d-4           [-1, 64, 64, 64]               0
            Linear-5                 [-1, 128]       8,388,736
           Dropout-6                 [-1, 128]               0
            Linear-7                   [-1, 2]             258
================================================================
Total params: 8,408,386
Trainable params: 8,408,386
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 30.00
Params size (MB): 32.08
Estimated Total Size (MB): 62.83
----------------------------------------------------------------
```

DropOut



Consiste en apagar o **desconectar** de manera **aleatoria** una parte de las **neuronas** en una **capa** durante el entrenamiento.
"**Eliminar**" valores significa eliminarlos **temporalmente** de la red para el pase de avance actual, junto con todas sus conexiones entrantes y salientes.
El Dropout se puede considerar una entre las **técnicas de inyección de ruido** (noise injection) y puede verse como una inyección de ruido en las unidades ocultas de la red.
En la **práctica**, durante el entrenamiento, una cierta cantidad de salidas de capa se ignoran aleatoriamente (se eliminan) con **probabilidad p**.
La probabilidad de elegir cuántos nodos se deben descartar es el **hiperparametro de la función de descarte,** y se puede aplicar tanto a las capas ocultas como a las capas de entrada.

Al usar el dropout, la **misma capa modifica su conectividad** y **buscará caminos** alternativos para transmitir la información en la siguiente capa.
De esta forma en cada iteración del entrenamiento se realiza una "**vista**" diferente de la capa configurada, y **conceptualmente**, se aproxima al entrenamiento de **múltiples redes** con diferentes **arquitecturas en paralelo**.
Entonces, **cada iteración tendrá un conjunto diferente** de nodos en la red y esto da como resultado un conjunto diferente de salidas.
Esto se puede considerar como una **técnica de conjunto o ensamblado,** que generalmente funcionan mejor ya que capturan **más aleatoriedad**, mejorando las situaciones en las que las capas de la red se adaptan conjuntamente para corregir los errores de las capas anteriores, y haciendo que un modelo resulte más sólido.

En general el DropOut **aumenta la escasez de la red** y resulta muy eficaz como tecnica de regularización.

Entrenamiento

Optimizer
La función optimizadora tiene el objetivo de ajustar **pesos** y **sesgos** de una CNN para **minimizar la función de pérdida** (loss function).
Por tanto, se ha experimentado con diferentes optimizer y hiperparámetros como **SGD**, **Adam** y **AdaDelta**, realizando diferentes pruebas donde este ultimo (con lr=0.1) resultò con resultados mejores.
Al final se utiliza un optimizador Adam el modelo 1 y AdaDelta fue aplicado el Modelo 2.

Se ha realizado el entrenamiento con las siguientes configuraciones:
• **Funcion de perdida =** CrossEntropyLoss
• **Optimizador**: hicimos varios intentos hasta dejar como referencia **Adam** con tasa de aprendizaje de 0.001 y **decaimiento de peso** step LR de 0,001.
• **Numero de epocas =** 10

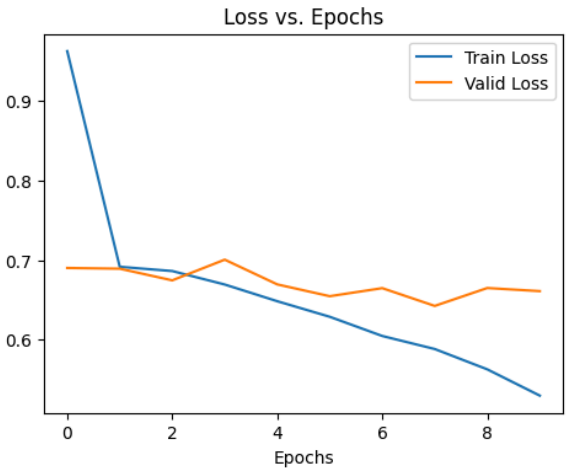
```
Epoch [1/10], Train Loss: 0.9628, Train Acc: 51.90%, Valid Loss: 0.6900, Valid Acc: 53.17%
Epoch [2/10], Train Loss: 0.6915, Train Acc: 53.12%, Valid Loss: 0.6892, Valid Acc: 53.17%
Epoch [3/10], Train Loss: 0.6862, Train Acc: 56.31%, Valid Loss: 0.6745, Valid Acc: 53.66%
Epoch [4/10], Train Loss: 0.6692, Train Acc: 59.38%, Valid Loss: 0.7005, Valid Acc: 51.22%
Epoch [5/10], Train Loss: 0.6482, Train Acc: 63.24%, Valid Loss: 0.6694, Valid Acc: 57.07%
Epoch [6/10], Train Loss: 0.6286, Train Acc: 65.87%, Valid Loss: 0.6545, Valid Acc: 56.10%
Epoch [7/10], Train Loss: 0.6045, Train Acc: 68.75%, Valid Loss: 0.6647, Valid Acc: 58.54%
Epoch [8/10], Train Loss: 0.5881, Train Acc: 68.38%, Valid Loss: 0.6423, Valid Acc: 58.54%
Epoch [9/10], Train Loss: 0.5625, Train Acc: 72.00%, Valid Loss: 0.6648, Valid Acc: 55.61%
Epoch [10/10], Train Loss: 0.5295, Train Acc: 72.86%, Valid Loss: 0.6609, Valid Acc: 56.59%
Entrenamiento finalizado!
```

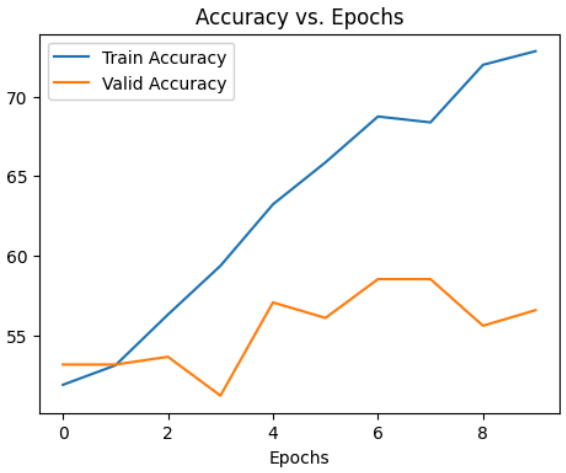
Evaluacion

La evaluación de modelos permite medir la capacidad de generalización de un modelo, y su performance con datos no vistos previamente.
La evaluación de la precisión en este ejercicio se realiza por la fórmula:
Precisión(Accuracy) = TP + TN / Total de Predicciones
Donde:
TP (Verdadero positivos) corresponde a los casos en los que el modelo predijo correctamente una instancia positiva. Es decir, la cara falsa clasificada como falsa.
TN (Verdaderos Negativos) corresponde a los casos en los que el modelo predijo correctamente una instancia negativa. es decir, la cara real clasificada como real.

Matriz de confusión
Una matriz de confusión muestra de manera compacta la cantidad de predicciones correctas e incorrectas realizadas por el modelo a traves de una representación visual.

• **Precision** (Accuracy) en el conjunto de training: 72.8%, overall 56.6%






```
Confusion Matrix
[[59 41]
 [47 57]]
```

```
Classification Report
              precision    recall  f1-score   support

training_fake       0.56      0.59      0.57       100
training_real       0.58      0.55      0.56       104

     accuracy                           0.57       204
    macro avg       0.57      0.57      0.57       204
 weighted avg       0.57      0.57      0.57       204
```

Consideraciones - Modelo 1

• La arquitectura de la red neuronal CNN parece ser demasiado simple, con dos capas convolucionales y de capas lineales.
• La precision en el conjunto de prueba es del 56.6%, y todas la metricas analizadas indican un rendimiento bastante **bajo**.
• En la matriz de confusion se observa que, en su mayoría, el modelo logra clasificar correctamente ambas clases, pero con muchos fallos de muestras.

Con esta consideraciones deducimos que la mejor opción para mejorar la red era con un arquitectura mas articulada o mediante un modelo pre-entrenado con transfer learning.

- Modelo 2.1

Pre-processing

De este modelo custom se realizaron **2 experimentaciones**, en la primera se ha utilizado el mismo pre-processing de antes:
- **Tamaño** de las imágenes: 128x128 pıxeles.
- **Transformaciones** aplicadas: Resize, Normalize, ToTensor
- **División** del conjunto de datos en entrenamiento (80%), prueba (10%) y validacion (10%).

Arquitectura

En este modelo a diferencia del primero, se utilizaron 4 capas convolucionales y batch normalization, con un optimizer Adam y DropOut.

Se ha implementado la siguiente **arquitectura:**

• **Capa de convolución**: 3 canales de entrada, 32 canales de salida, kernel de 3x3, stride de 1, padding de 1.
• **Batch Normalization** después de la primera capa de convolución.

• **Capa de convolución**: 32 canales de entrada, 64 canales de salida, kernel de 3x3, stride de 1, padding de 1.
• **Capa de pooling** (MaxPool2d) con kernel de 2x2 y stride de 2.

• **Capa de convolución**: 3 canales de entrada, 32 canales de salida, kernel de 3x3, stride de 1, padding de 1.
• **Batch Normalization** después de la primera capa de convolución.

• **Capa de convolución**: 32 canales de entrada, 64 canales de salida, kernel de 3x3, stride de 1, padding de 1.
• **Capa de pooling** (MaxPool2d) con kernel de 2x2 y stride de 2.

• **Capa completamente conectada** (fc1) con 64x64x64 nodos de entrada y 128 nodos de salida.
• **Capa completamente conectada** (fc2) con 128 nodos de entrada y 2 nodos de salida (clasificacion binaria).

Dropout

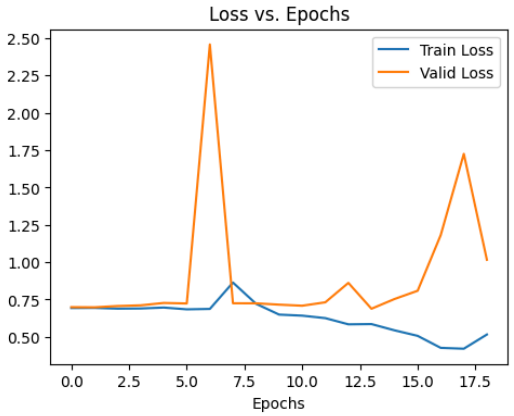
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 256, 256]             896
       BatchNorm2d-2         [-1, 32, 256, 256]              64
            Conv2d-3         [-1, 64, 256, 256]          18,496
         MaxPool2d-4         [-1, 64, 128, 128]               0
            Conv2d-5        [-1, 128, 128, 128]          73,856
       BatchNorm2d-6        [-1, 128, 128, 128]             256
            Conv2d-7        [-1, 256, 128, 128]         295,168
           Dropout-8        [-1, 256, 128, 128]               0
         MaxPool2d-9          [-1, 256, 64, 64]               0
           Linear-10                  [-1, 512]     134,218,240
          Dropout-11                  [-1, 512]               0
           Linear-12                    [-1, 2]           1,026
================================================================
Total params: 134,608,002
Trainable params: 134,608,002
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 176.01
Params size (MB): 513.49
Estimated Total Size (MB): 690.25
----------------------------------------------------------------
```

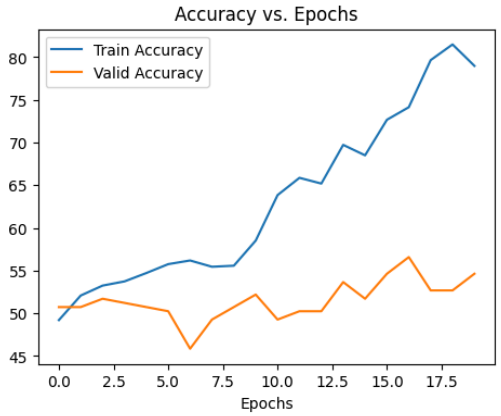
Entrenamiento

```
Epoch [1/20],  Train Loss: 168.1274, Avg Train Loss: 9.8898, Train Acc: 49.20%, Valid Loss: 2.0850 ,Avg Valid Loss: 0.6950, Valid Acc: 50.73%
Epoch [2/20],  Train Loss: 11.7798, Avg Train Loss: 0.6929, Train Acc: 52.08%, Valid Loss: 2.0964 ,Avg Valid Loss: 0.6988, Valid Acc: 50.73%
Epoch [3/20],  Train Loss: 11.7933, Avg Train Loss: 0.6937, Train Acc: 53.25%, Valid Loss: 2.0934 ,Avg Valid Loss: 0.6978, Valid Acc: 51.71%
Epoch [4/20],  Train Loss: 11.7102, Avg Train Loss: 0.6888, Train Acc: 53.74%, Valid Loss: 2.1190 ,Avg Valid Loss: 0.7063, Valid Acc: 51.22%
Epoch [5/20],  Train Loss: 11.7278, Avg Train Loss: 0.6899, Train Acc: 54.72%, Valid Loss: 2.1329 ,Avg Valid Loss: 0.7110, Valid Acc: 50.73%
Epoch [6/20],  Train Loss: 11.8341, Avg Train Loss: 0.6961, Train Acc: 55.76%, Valid Loss: 2.1807 ,Avg Valid Loss: 0.7269, Valid Acc: 50.24%
Epoch [7/20],  Train Loss: 11.6302, Avg Train Loss: 0.6841, Train Acc: 56.19%, Valid Loss: 2.1705 ,Avg Valid Loss: 0.7235, Valid Acc: 45.85%
Epoch [8/20],  Train Loss: 11.6777, Avg Train Loss: 0.6869, Train Acc: 55.45%, Valid Loss: 7.3760 ,Avg Valid Loss: 2.4587, Valid Acc: 49.27%
Epoch [9/20],  Train Loss: 14.6824, Avg Train Loss: 0.8637, Train Acc: 55.58%, Valid Loss: 2.1743 ,Avg Valid Loss: 0.7248, Valid Acc: 50.73%
Epoch [10/20],  Train Loss: 12.2641, Avg Train Loss: 0.7214, Train Acc: 58.52%, Valid Loss: 2.1735 ,Avg Valid Loss: 0.7245, Valid Acc: 52.20%
Epoch [11/20],  Train Loss: 11.0318, Avg Train Loss: 0.6489, Train Acc: 63.85%, Valid Loss: 2.1469 ,Avg Valid Loss: 0.7156, Valid Acc: 49.27%
Epoch [12/20],  Train Loss: 10.9124, Avg Train Loss: 0.6419, Train Acc: 65.87%, Valid Loss: 2.1252 ,Avg Valid Loss: 0.7084, Valid Acc: 50.24%
Epoch [13/20],  Train Loss: 10.6305, Avg Train Loss: 0.6253, Train Acc: 65.20%, Valid Loss: 2.1940 ,Avg Valid Loss: 0.7313, Valid Acc: 50.24%
Epoch [14/20],  Train Loss: 9.9233, Avg Train Loss: 0.5837, Train Acc: 69.73%, Valid Loss: 2.5841 ,Avg Valid Loss: 0.8614, Valid Acc: 53.66%
Epoch [15/20],  Train Loss: 9.9528, Avg Train Loss: 0.5855, Train Acc: 68.50%, Valid Loss: 2.0644 ,Avg Valid Loss: 0.6881, Valid Acc: 51.71%
Epoch [16/20],  Train Loss: 9.2399, Avg Train Loss: 0.5435, Train Acc: 72.67%, Valid Loss: 2.2604 ,Avg Valid Loss: 0.7535, Valid Acc: 54.63%
Epoch [17/20],  Train Loss: 8.6114, Avg Train Loss: 0.5066, Train Acc: 74.14%, Valid Loss: 2.4259 ,Avg Valid Loss: 0.8086, Valid Acc: 56.59%
Epoch [18/20],  Train Loss: 7.2447, Avg Train Loss: 0.4262, Train Acc: 79.66%, Valid Loss: 3.5445 ,Avg Valid Loss: 1.1815, Valid Acc: 52.68%
Epoch [19/20],  Train Loss: 7.1476, Avg Train Loss: 0.4204, Train Acc: 81.50%, Valid Loss: 5.1770 ,Avg Valid Loss: 1.7257, Valid Acc: 52.68%
Epoch [20/20],  Train Loss: 8.7645, Avg Train Loss: 0.5156, Train Acc: 78.98%, Valid Loss: 3.0480 ,Avg Valid Loss: 1.0160, Valid Acc: 54.63%
Entrenamiento finalizado!
```

Evaluacion

Precision (Accuracy) en el conjunto de test: 78.9%, overall 54.6%






```
Confusion Matrix
[[46 44]
 [51 63]]

Classification Report
              precision    recall  f1-score   support

training_fake       0.47      0.51      0.49        90
training_real       0.59      0.55      0.57       114

     accuracy                           0.53       204
    macro avg       0.53      0.53      0.53       204
 weighted avg       0.54      0.53      0.54       204
```

Consideraciones - Modelo 2.1

• La precision en el conjunto de prueba sigue siendo inferior al 60% y la matriz de confusion y todas la metricas analizadas no mejora significativamente.

Con este mismo modelo intentamos realizar Data Augmentation.

Modelo 2.2 con Data Augmentation

Pre-Processing con Data Augmentation

En el segundo intento se realizaron unas transformaciones diferentes para realizar la **data augmentation**:
- **Tamaño** de las imágenes: 128x128 pıxeles.
- **División** del conjunto de datos en entrenamiento (80%), prueba (10%) y validacion (10%).
- **Transformaciones** aplicadas: Resize, Random Horizontal Flip, Random Rotation, ColorJitter, GaussianBlur, ToTensor.

```
# Define the data augmentation transformations
data_transforms1 = transforms.Compose([
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomApply([transforms.GaussianBlur(kernel_size = (7,13), sigma = (6 , 7))]),
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])
```

A través de las transformaciones se ha realizado una **aumentación** del numero de imágenes utilizadas para el entrenamiento es de circa **4x** (de 1632 a 6531 instancias en train_size).

```
print(train_size)
print(valid_size)
print(test_size)
```

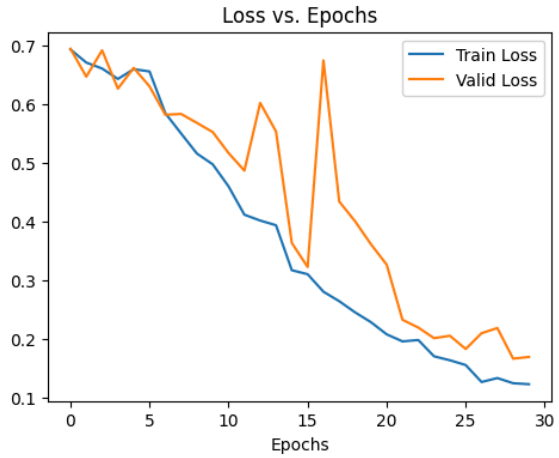
```
/content/gdrive/MyDrive/Colab Notebooks/real_and_fake_face
6531
816
817
```

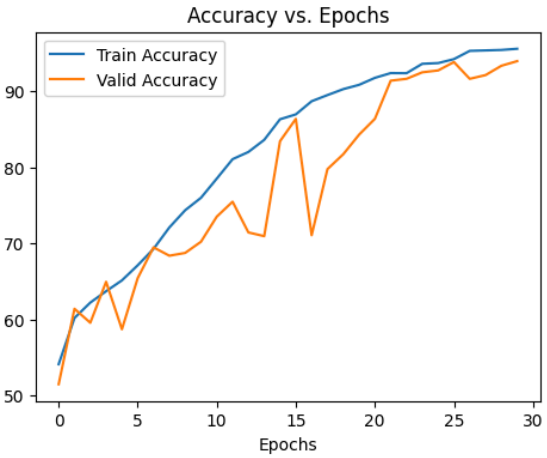
Entrenamiento

```
Epoch [1/30],  Avg Train Loss: 0.6932, Train Acc: 54.11%, Avg Valid Loss: 0.6944, Valid Acc: 51.47%
Epoch [2/30],  Avg Train Loss: 0.6709, Train Acc: 60.19%, Avg Valid Loss: 0.6469, Valid Acc: 61.40%
Epoch [3/30],  Avg Train Loss: 0.6608, Train Acc: 62.21%, Avg Valid Loss: 0.6917, Valid Acc: 59.56%
Epoch [4/30],  Avg Train Loss: 0.6430, Train Acc: 63.71%, Avg Valid Loss: 0.6265, Valid Acc: 64.95%
Epoch [5/30],  Avg Train Loss: 0.6596, Train Acc: 65.14%, Avg Valid Loss: 0.6618, Valid Acc: 58.70%
Epoch [6/30],  Avg Train Loss: 0.6558, Train Acc: 67.13%, Avg Valid Loss: 0.6304, Valid Acc: 65.44%
Epoch [7/30],  Avg Train Loss: 0.5846, Train Acc: 69.28%, Avg Valid Loss: 0.5819, Valid Acc: 69.49%
Epoch [8/30],  Avg Train Loss: 0.5500, Train Acc: 72.10%, Avg Valid Loss: 0.5833, Valid Acc: 68.38%
Epoch [9/30],  Avg Train Loss: 0.5158, Train Acc: 74.37%, Avg Valid Loss: 0.5681, Valid Acc: 68.75%
Epoch [10/30],  Avg Train Loss: 0.4975, Train Acc: 76.01%, Avg Valid Loss: 0.5525, Valid Acc: 70.22%
Epoch [11/30],  Avg Train Loss: 0.4599, Train Acc: 78.53%, Avg Valid Loss: 0.5167, Valid Acc: 73.53%
Epoch [12/30],  Avg Train Loss: 0.4117, Train Acc: 81.09%, Avg Valid Loss: 0.4868, Valid Acc: 75.49%
Epoch [13/30],  Avg Train Loss: 0.4017, Train Acc: 82.04%, Avg Valid Loss: 0.6021, Valid Acc: 71.45%
Epoch [14/30],  Avg Train Loss: 0.3937, Train Acc: 83.63%, Avg Valid Loss: 0.5536, Valid Acc: 70.96%
Epoch [15/30],  Avg Train Loss: 0.3171, Train Acc: 86.34%, Avg Valid Loss: 0.3637, Valid Acc: 83.46%
Epoch [16/30],  Avg Train Loss: 0.3103, Train Acc: 86.99%, Avg Valid Loss: 0.3226, Valid Acc: 86.40%
Epoch [17/30],  Avg Train Loss: 0.2805, Train Acc: 88.72%, Avg Valid Loss: 0.6744, Valid Acc: 71.08%
Epoch [18/30],  Avg Train Loss: 0.2643, Train Acc: 89.53%, Avg Valid Loss: 0.4341, Valid Acc: 79.78%
Epoch [19/30],  Avg Train Loss: 0.2453, Train Acc: 90.31%, Avg Valid Loss: 0.4005, Valid Acc: 81.74%
Epoch [20/30],  Avg Train Loss: 0.2287, Train Acc: 90.87%, Avg Valid Loss: 0.3615, Valid Acc: 84.31%
Epoch [21/30],  Avg Train Loss: 0.2079, Train Acc: 91.79%, Avg Valid Loss: 0.3266, Valid Acc: 86.40%
Epoch [22/30],  Avg Train Loss: 0.1958, Train Acc: 92.42%, Avg Valid Loss: 0.2327, Valid Acc: 91.42%
Epoch [23/30],  Avg Train Loss: 0.1982, Train Acc: 92.41%, Avg Valid Loss: 0.2194, Valid Acc: 91.67%
Epoch [24/30],  Avg Train Loss: 0.1704, Train Acc: 93.65%, Avg Valid Loss: 0.2013, Valid Acc: 92.52%
Epoch [25/30],  Avg Train Loss: 0.1637, Train Acc: 93.74%, Avg Valid Loss: 0.2054, Valid Acc: 92.77%
Epoch [26/30],  Avg Train Loss: 0.1555, Train Acc: 94.26%, Avg Valid Loss: 0.1829, Valid Acc: 93.87%
Epoch [27/30],  Avg Train Loss: 0.1266, Train Acc: 95.35%, Avg Valid Loss: 0.2098, Valid Acc: 91.67%
Epoch [28/30],  Avg Train Loss: 0.1333, Train Acc: 95.41%, Avg Valid Loss: 0.2186, Valid Acc: 92.16%
Epoch [29/30],  Avg Train Loss: 0.1246, Train Acc: 95.47%, Avg Valid Loss: 0.1664, Valid Acc: 93.38%
Epoch [30/30],  Avg Train Loss: 0.1229, Train Acc: 95.62%, Avg Valid Loss: 0.1694, Valid Acc: 94.00%
Entrenamiento finalizado!
```

Evaluacion

Precision (Accuracy) en el conjunto de train: 95.6%, overall 94.8%.






```
Confusion Matrix
[[354  41]
 [ 52 370]]
```

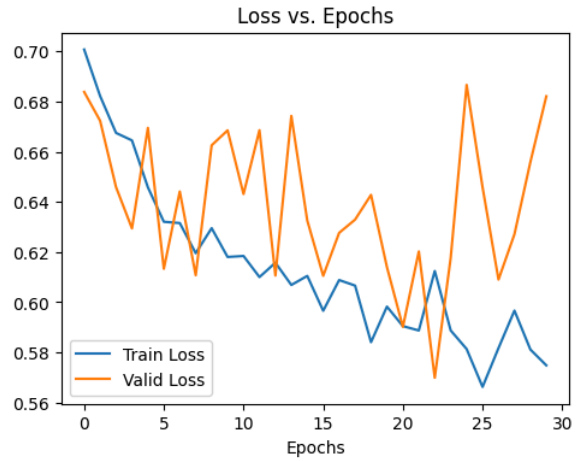
Consideraciones - Modelo 2.2

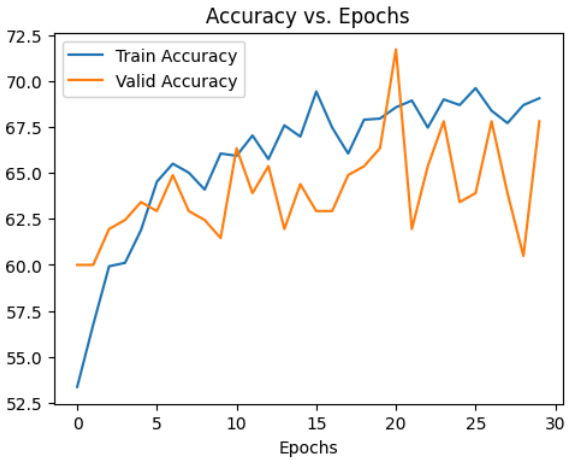
• Con Data Augmentation la mejora en los resultados fue destacable
• Vista la complexidad de la tarea parece necesario utilizar un conjunto de datos mas grande.

Transfer Learning

El Fine Tuning consiste en adaptar una red pre-entrenada a una tarea específica, aprovechando del conocimiento previo de la red para mejorar la capacidad de generalización del modelo.
Se optó por realizar el proceso de ajuste fino con modelos pre-entrenados,eligendo **ResNet18**.

Evaluacion





Conclusiones

En este estudio, se ha desarrollado un modelo de clasificación de caras reales y falsas mediante técnicas de aprendizaje profundo.

El objetivo era de realizar un proceso de exploración y experimentación utilizando dos enfoques distintos:
- realizar modelos con arquitectura custom, desarrollado desde cero (2 intentos)
- utilizar técnica de fine-tuning con un modelo preentrenado.

Después de unos intentos iniciales que mostraron limitaciones en términos de rendimiento, las modificas de arquitecturas, hiperparametros, tecnicas de regularización y sobre todo con data augmentation y transfer learning se han obtenido modelos más efectivo, logrando mejoras significativas en la precisión.
Además, se ha adquirido más comprensión en como tratar con sobreajuste.