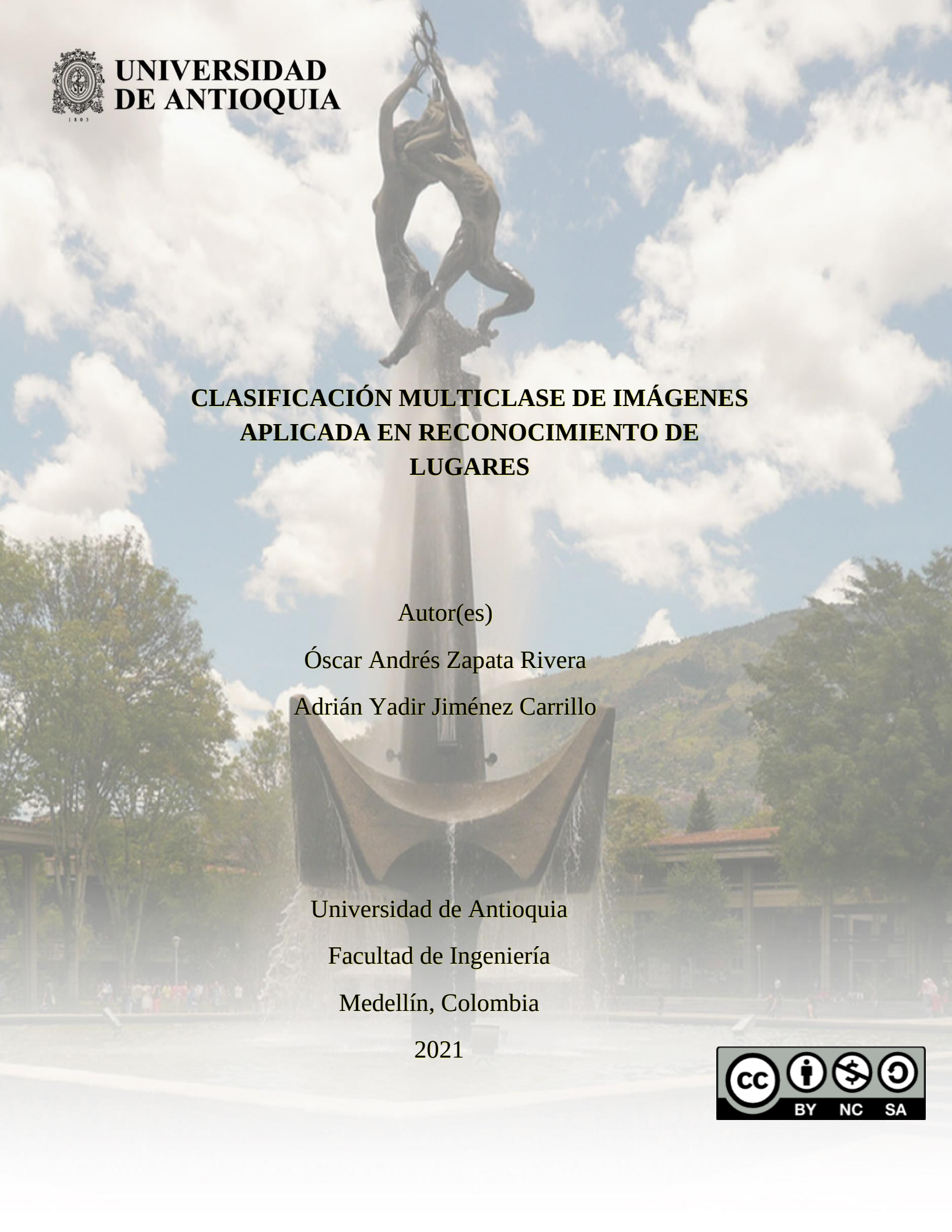

UNIVERSIDAD DE ANTIOQUIA

CLASIFICACIÓN MULTICLASE DE IMÁGENES APLICADA EN RECONOCIMIENTO DE LUGARES

Autor(es)

Óscar Andrés Zapata Rivera

Adrián Yadir Jiménez Carrillo

Universidad de Antioquia

Facultad de Ingeniería

Medellín, Colombia

2021

Clasificación Multiclase de Imágenes Aplicada en Reconocimiento de Lugares

Óscar Andrés Zapata Rivera
Adrián Yadir Jiménez Carrillo

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
Especialista en Análisis y Ciencia de Datos

Asesores (a):
Ingeniera Daniela Serna Buitrago

Línea de Investigación:
Analítica de Datos

Universidad de Antioquia
Facultad de Ingeniería.
Medellín, Colombia
2021.

TABLA DE CONTENIDOS

1. RESUMEN EJECUTIVO 3

2. DESCRIPCIÓN DEL PROBLEMA 5

2.1 APROXIMACIÓN DESDE LA ANALÍTICA DE DATOS 6

3. DATOS 7

3.1 DATOS ORIGINALES 7

3.2 DATASETS 10

4. PROCESO DE ANALÍTICA 11

4.1 Modelo de machine learning clásico: 11

4.2 Modelo de deep learning: 14

5. METODOLOGÍA 17

5.1 BASELINE 17

5.2 ITERACIONES y EVOLUCIÓN 18

5.3 HERRAMIENTAS 21

6. RESULTADOS 22

6.2 CONSIDERACIONES DE PRODUCCIÓN 31

7. CONCLUSIONES 32

8. BIBLIOGRAFIA 33

1. RESUMEN EJECUTIVO

El reconocimiento de imágenes es un problema común entre los desafíos que se encuentran en materia de inteligencia artificial debido a que los seres humanos intrínsecamente tiene como uno de sus sentidos más importantes la visión y por esto se ha construido un mundo lleno de signos visuales para mejorar la comunicación, el entendimiento y la calidad de vida de las personas. Entre muchas de las cosas desarrolladas con el paso del tiempo se tienen aplicaciones en en innumerables áreas como lo son el área industrial, médica, social, en el marketing, en automatización de procesos visuales, entre muchas otras.

En el caso expuesto en este trabajo, se decide abordar el problema de reconocimiento de imágenes enfocado en imágenes de lugares desde diferentes tomas y perspectivas, teniendo como set de datos un grupo de imágenes de diferentes lugares, para así entrenar un modelo que permita la identificación del lugar al que pertenece una nueva imagen ingresada, esto corresponde a un problema de clasificación de imágenes de múltiples clases.

En la ejecución del proyecto se decidió optar inicialmente por modelos matemáticos clásicos, como el "Clasificador Bayesiano Ingenuo" (GNB por sus siglas en inglés), dada la implementación que tiene este algoritmo se deben descomponer inicialmente las imágenes a matrices con datos cuantitativos para poderlas procesar y hacer predicciones sobre estas. Con esto se obtuvo un resultado de predicción bajo causado por el método de descomposición utilizado previo a procesar las imágenes, ya que la representación de una imagen por características numéricas puede quedarse corta ante la realidad y hace complejo la generalización de modelos tradicionales de machine learning en este tipo de problemas.

Debido a la baja generalización del primer modelo también se implementa una red neuronal convolucional que dada su implementación permite el procesamiento de imágenes completas. Se utiliza transfer learning en la inicialización de los pesos de la red para llegar a un resultado satisfactorio que al compararlo con él del método anterior es superior.

En el desarrollo de los métodos se enfrentaron varios problemas, entre ellos la capacidad de procesamiento computacional y el desbalance de los datos, este último ocasionado por la cantidad variada de imágenes que había para los diferentes lugares, donde unos presentaban más datos que otros, esto hacía que al dividir el set de datos no se contará para algunos lugares suficiente información para entrenamiento y validación, como consecuencia aumentaba la probabilidad de obtener predicciones erróneas.

Posterior a la corrida de los modelos, se identificó al analizar las imágenes que muchas de ellas presentaban personas que se identificaban claramente su rostro, por ello se decide crear un paso

intermedio de anonimización de rostros para no incurrir en problemas legales que atentan la privacidad de las personas.

La implementación de lo anterior mencionado puede verse en el siguiente repositorio: https://github.com/andreszr/Multiclass-image-classification

2. DESCRIPCIÓN DEL PROBLEMA

La identificación de lugares y/o de objetos desde diferentes perspectivas es un problema que se ha querido solucionar desde los inicios de la inteligencia artificial y la robótica, los cuales son en esencia campos de conocimiento que buscan replicar y/o automatizar las tareas que puede realizar un ser humano con sus sentidos, es por esto que el reconocimiento visual se convierte en una tarea fundamental en el área. Las personas podemos reconocer un lugar fácilmente al ver una foto de un monumento emblemático como por ejemplo la Torre Eiffel en París o el puente Golden Gate de San Francisco; también lugares más sencillos como la fachada de una casa o una habitación, esta capacidad trasladada al poder de los algoritmos de machine learning no solo facilita la solución de un problema en específico, sino que otorga la capacidad de automatizar y solucionar múltiples problemas del día a día de las personas.

Describiendo ejemplos de lo que esta capacidad de predecir lugares puede lograr se tiene la premisa inicial del reto planteado por Google (Landmark recognition): "*Luego de unas largas vacaciones sería interesante poder saber exactamente a qué lugar corresponden cada uno de los lugares memorables que visitamos*". Sin embargo esta es solo una aplicación de las múltiples que se tienen ya que con las mismas estrategias planteadas en este documento se ha observado el potencial a la hora de identificar diferentes tipos de objetos, por lo tanto se podrían clasificar platos de comida, frutas, animales, flores, entre otros, lo cual a su vez implica la solución de múltiples problemas más en diferentes campos.

Otro de los problemas principales que se ha tenido con este tipo de análisis históricamente es lo difícil que es conseguir una gran cantidad de imágenes etiquetadas con los lugares a los que pertenecen, sin embargo gracias al avance del big data, las arquitecturas distribuidas, el IoT y que las personas están tomando constantemente una gran cantidad de fotos y subiendolas a la red, se hace posible que compañías como Google con toda su infraestructura puedan obtener dataset inmensos de imágenes anotadas con el lugar en donde fue tomada.

El problema se trata como un modelo de clasificación multiclase de computer vision, ya que se quieren reconocer imágenes (lugares) y asignarles una clase (lugar) al cual corresponden, teniendo N número de clases, para esto se tiene inicialmente varias maneras de abordar el problema y se debe realizar un análisis inicial de la cantidad de lugares diferentes, decidir qué tipo de preprocesamiento se realizará sobre las imágenes y posteriormente aplicar machine learning clásico, transfer learning y/o deep learning para poder obtener resultados de clasificación multiclase sobre las imágenes enviadas al modelo.

2.1 APROXIMACIÓN DESDE LA ANALÍTICA DE DATOS

El modelo tiene como aproximación inicial el poder recibir una imagen y retornar al usuario el nombre o etiqueta del lugar/objeto al cual pertenece haciendo posible así identificar estas imágenes desde diferentes tomas o perspectivas, se plantea en el alcance del proyecto adicionalmente anonimizar los rostros de personas ya que muchas de las imágenes del dataset contiene fotos de personas en sitios turísticos. Se plantea también desplegar un API que permita el envío de imágenes para el consumo del modelo fácilmente.

Se definen dos tipos de soluciones de aprendizaje automático para dar respuesta a lo descrito en el párrafo anterior. La construcción de un primer modelo o línea base a partir de un algoritmo de machine learning tradicional, haciendo previamente un preprocesamiento de los datos y un segundo modelo utilizando una red neuronal de deep learning con ayuda de transfer learning.

Para la primera aproximación se debe realizar primero una detección de características o keypoints, los cuales son puntos sobresalientes en una estructura, objeto o lugar para esto se plantea utilizar métodos como el kaze y el orb la cual proporciona la librería de opencv y una vez con los puntos clave identificados hacer un modelo que se entrene con estos puntos claves y probar si es capaz de clasificar nuevas imágenes correctamente.

En la segunda aproximación se procede con una implementación mucho más directa donde no es necesario preprocesar las imágenes, ni hacer extracción de keypoints. Se utiliza una red pre-entrenada, se ajusta el modelo, el batch size, se realiza data augmentation para mover las imágenes en diferentes direcciones y obtener más muestras “virtuales” de los lugares a clasificar, se realiza el ajuste de hiperparametros, se entrena el modelo y se prueban los resultados con los datos de validación.

Pensando un poco más en potenciales aplicaciones de este proyecto a futuro, en el tema del despliegue del modelo es interesante la idea de integrar esto a una aplicación la cual al tomar fotos con un smartphone te retorne el lugar, escultura, objeto, estatua, etc. que estás fotografiando o con los fotogramas de un video para que sea más en tiempo real. También podría traer información asociada al lugar sin tener que buscar en un navegador así como puede ser la descripción básica o historia del lugar.

3. DATOS

3.1 DATOS ORIGINALES

Los datos se recopilaron previamente por parte de Google para el reto de kaggle mencionado previamente (Google landmark recognition) en el cual se proporciona el acceso a los datos con un dataset que contiene tres campos, el id de la foto en especifica (id del registro), la url de la imagen de la cual posteriormente se debe hacer la descarga para tener el dataset de imágenes y una última columna que contiene el id del lugar en el cual se tomó la foto (esto es el landmark_id, se tienen múltiples registros con el mismo landmark_id).

```
df = pd.read_csv('train.csv')
df
```

	id	url	landmark_id
0	97c0a12e07ae8dd5	http://lh4.ggpht.com/-f8xYA5l4apw/RSziSQVaABI/...	6347
1	650c989dd3493748	https://lh5.googleusercontent.com/-PUnMrX7oOyA...	12519
2	05e63ca9b2cde1f4	http://mw2.google.com/mw-panoramio/photos/medi...	264
3	08672eddcb2b7c93	http://lh3.ggpht.com/-9fgSxDYwhHA/SMvGEoltKTI/...	13287
4	fc49cb32ef7f1e89	http://lh6.ggpht.com/-UGAXxvPbr98/S-jGZbyMIPI/...	4018
...	...	...	...
1225024	4bb5a501e5b26a6a	https://lh6.googleusercontent.com/-mRrQU3t5cYw...	9737
1225025	2cd8a404796cfe0e	https://lh6.googleusercontent.com/-0UB5gFx6w7M...	7758
1225026	8733b8b469fb8c1b	http://lh3.ggpht.com/-TDQWNVvJQDI/SI3HZSA4D3I/...	13170
1225027	14dd9e8790397c83	https://lh4.googleusercontent.com/-anV4Xpo0UuM...	5669
1225028	4303049d5a6b5602	https://lh6.googleusercontent.com/-1pe3ldzCDAw...	4987

1225029 rows × 3 columns

Figura 1. Muestra del dataset

Un poco más a profundidad, en google se menciona que el dataset de la competencia y sobre el cual se realiza el modelo es una versión limpia del dataset Google Landmarks Dataset v2 (GLDv2).

El acceso a estos datos es gratuito y público dadas las reglas de la plataforma y la amplia comunidad que tiene. Los datos vienen compilados en un archivo formato .csv (comma separated values), es un archivo de texto donde sus valores vienen separados por comas, y al ser un formato ligero su peso llega a los 124 MB (MegaBytes), cada uno de los registros posee solo tres columnas:

- Id: identificador único del registro, no hay valores repetidos entre las muestras.
- Url: dirección del recurso web donde se encuentra la imagen real del registro.

- Landmark_id: identificador único del lugar al que corresponde la imagen, o dónde fue tomada la fotografía, este valor puede o no estar repetido entre las muestras.

```
df.shape

(1225029, 3)
```

Figura 2. Forma del dataset

También se encuentra que la distribución de las clases es desbalanceada, puesto que de 1225029 imágenes (registros) hay clases que contienen hasta 49451 muestras mientras que también hay clases que poseen solo una muestra, y a la hora de entrenar un modelo esto es un problema.

```
df['landmark_id'].value_counts()

9633     49451
6051     49222
None     34330
6599     22762
9779     18062
         ...
4103         1
7916         1
10331        1
2615         1
12703        1
Name: landmark_id, Length: 14947, dtype: int64
```

Figura 3. Cantidad de muestras por clase

Por otro lado, las imágenes tienen en común el formato en el que vienen luego de descargadas, siendo este .JPEG (Joint Photographic Experts Group), al hablar de su tamaño o resolución, no vienen en un tamaño estándar, al ser imágenes que llegan al dataset de diferentes fuentes, estas vienen en diferentes tamaños (1024 x 693, 500 x 750, 512 x 384, entre otros).

Durante la ejecución, se requiere tomar las muestras anteriores y descargar las imágenes para su procesamiento, a continuación se pueden ver algunos ejemplos de estas imágenes:

Figura 4. Imagen con Id: 97c0a12e07ae8dd5, tamaño original: 1024 x 693 píxeles.

Figura 5. Imagen con Id: f92fa6b19b56aa0a, tamaño original: 512 x 384 pixeles.

En el último caso observado (Figura 5) como muchos otros en el dataset, se observa que hay fotos que incluyen personas que podrían ser reconocidas por sus rostros, y es por esta razón por la cual se tomó la decisión de añadir un paso intermedio que permitiera la anonimización de los rostros para proteger la privacidad de dichas personas, y aunque los modelos generados no serán publicados o utilizados en producción donde esto pueda repercutir en acciones legales, el ejercicio académico debe asemejarse tanto como sea posible a un caso real y esto es sin duda una de los aspectos más importantes.

3.2 DATASETS

Previo al proceso de partición en entrenamiento y validación, se enfrentó un problema adicional y es la capacidad de cómputo a disposición, puesto que trabajar con modelos para el procesamiento de imágenes requiere una arquitectura física robusta que permita la ejecución correctamente lo cual es costoso y adicionalmente se debe tener gran capacidad de memoria para almacenar todas las imágenes durante los diferentes procesos.

Por esto, y teniendo en cuenta el desbalance de las clases, se decidió trabajar con un subconjunto del dataset original, tomando 200 imágenes como muestras, de 10 diferentes clases, para un total de dos mil imágenes que serían procesadas por los diferentes pasos del modelo, en estos pasos se incluye tal vez el más importante del baseline el cual se trata de encontrar los keypoints de cada una de las imágenes ya sea con el método kaze o con el método orb. Posteriormente partiendo de un dataframe con los keypoints identificados se realiza una partición de 70% de las muestras para entrenamiento y 30% para prueba o validación.

```
top10 = df['landmark_id'].value_counts().head(10)
top10
```

```
9633    49451
6051    49222
6599    22762
9779    18062
2061    12999
5554    10802
6651     9296
5376     9048
6696     9031
4352     8821
Name: landmark_id, dtype: int64
```

Figura 6. Top 10 clases con más registros

Para el modelo de deep learning implementando no se hace necesario encontrar los keypoints y se alimenta el modelo directamente con las imágenes .jpg del dataset, se decide descargar estas imágenes en carpetas que indiquen el landmark_id o marca del lugar al que pertenecen el peso de estas 2000 imagenes comprimido es de 2.7 Gb. También se añade un paso de data augmentation para varias las perspectivas de cada imagen y ayudar al modelo a generalizar mejor, se realizan pruebas tanto con los datos sin aumentar y aumentados para ver si se obtienen mejores resultados.

4. PROCESO DE ANALÍTICA

4.1 Modelo de machine learning clásico:

Para el baseline el pipeline que se tiene es el siguiente:

En este primer pipeline el proceso consta de un preprocesamiento de los datos como escoger clases con más cantidad de fotos obtener muestras del top10 de lugares con más imágenes y balancear, realizar la descarga de las imágenes, obtener los keypoints de las imágenes:

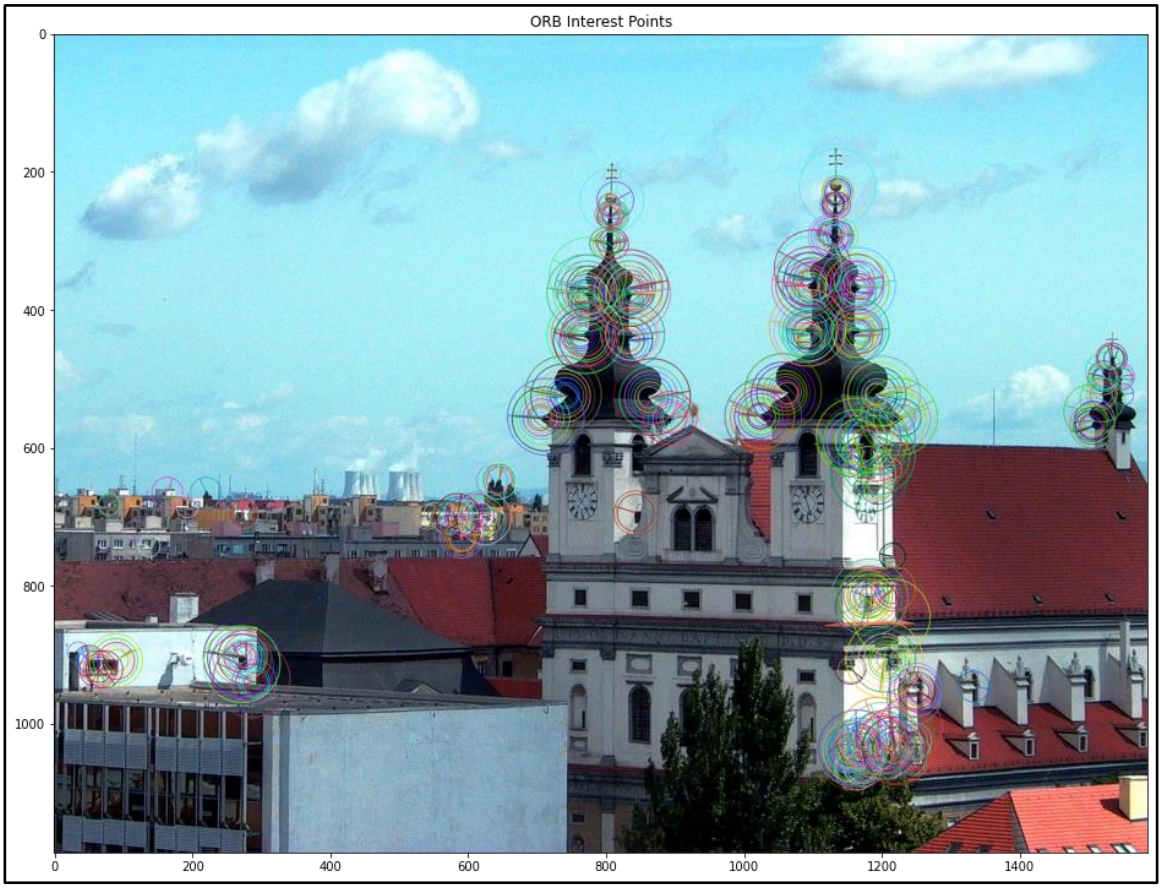

Figura 7. Puntos clave de una imagen con ORB

Luego de obtener los puntos claves de todas las imágenes se deben reorganizar en un dataframe para que sea posible aplicar algún modelo de ML posteriormente y de una manera sencilla.

	0	1	2	3	4	5	6	7	8	9	10	11
0	-0.000664	-0.001693	0.003246	0.009216	-0.011522	-0.005201	0.014313	0.023654	-0.009667	0.002223	0.011904	0.014028
1	0.047247	0.010677	0.127108	0.154313	-0.090806	-0.077606	0.174853	0.177696	-0.035320	-0.021626	0.103468	0.112415
2	0.039447	0.001574	0.069156	0.060533	0.043634	0.029634	0.213416	0.243436	-0.045534	0.038765	0.146393	0.140823
3	-0.033495	-0.018430	0.102787	0.090641	-0.008182	0.054522	0.168051	0.125552	0.013729	0.003688	0.104014	0.090581
4	-0.058567	0.015712	0.085616	0.082889	-0.124639	0.096747	0.281662	0.197722	0.026513	0.039191	0.077575	0.075332
...	...	...	...	...	...	...	...	...	...	...	...	...
1858	-0.025744	0.005915	0.105589	0.044471	-0.011489	-0.015188	0.180928	0.187615	0.065198	-0.020261	0.200281	0.114472
1859	0.031371	0.015269	0.061374	0.037198	-0.003928	0.028889	0.128197	0.105241	0.029753	0.018903	0.099266	0.113066
1860	0.001346	0.012254	0.011030	0.021517	-0.004935	0.043238	0.012949	0.057131	0.000366	0.001647	0.005326	0.012354
1861	0.000367	-0.020990	0.031813	0.044320	0.017838	-0.040300	0.022352	0.043064	0.010054	-0.030730	0.020944	0.035784
1862	-0.033065	0.008566	0.068860	0.037036	0.000282	-0.009813	0.063006	0.121543	0.022752	-0.010540	0.056377	0.152404

1863 rows × 2049 columns

Figura 8. Muestra de datos de los Keypoints

La última columna corresponde al landmark_id, la marca del lugar a la que pertenecen esos key points detectados para no perder de vista el objetivo de clasificar los lugares:

2043	2044	2045	2046	2047	2048
0.231683	0.034195	0.063746	0.062195	0.099304	9633.0
0.046715	-0.011000	-0.025803	0.063860	0.078336	9633.0
0.245408	0.002188	-0.009622	0.036995	0.079269	9633.0
0.145784	-0.007460	0.006521	0.055887	0.101328	9633.0
0.029955	-0.005905	0.017201	0.006913	0.019923	9633.0
...	...	...	...	...	...
0.165726	-0.000170	-0.006420	0.025380	0.117263	4352.0
0.163194	0.001413	-0.052215	0.083665	0.111491	4352.0
0.083210	0.000199	-0.001025	0.026953	0.039787	4352.0
0.072954	-0.011322	0.011342	0.025000	0.027774	4352.0
0.044562	-0.002740	0.004316	0.009896	0.009606	4352.0

Figura 9. Columna 2048 hace referencia al landmark_id

Con estos keypoints se pueden hacer comparaciones de keypoints entre dos imágenes como podemos apreciar en la figura 10 donde comparan los keypoints de la misma imagen.

Figura 10. Cruce de keypoints en una misma foto

Y la figura 11 muestra cómo sería comparar keypoints de imágenes diferentes de un mismo lugar.

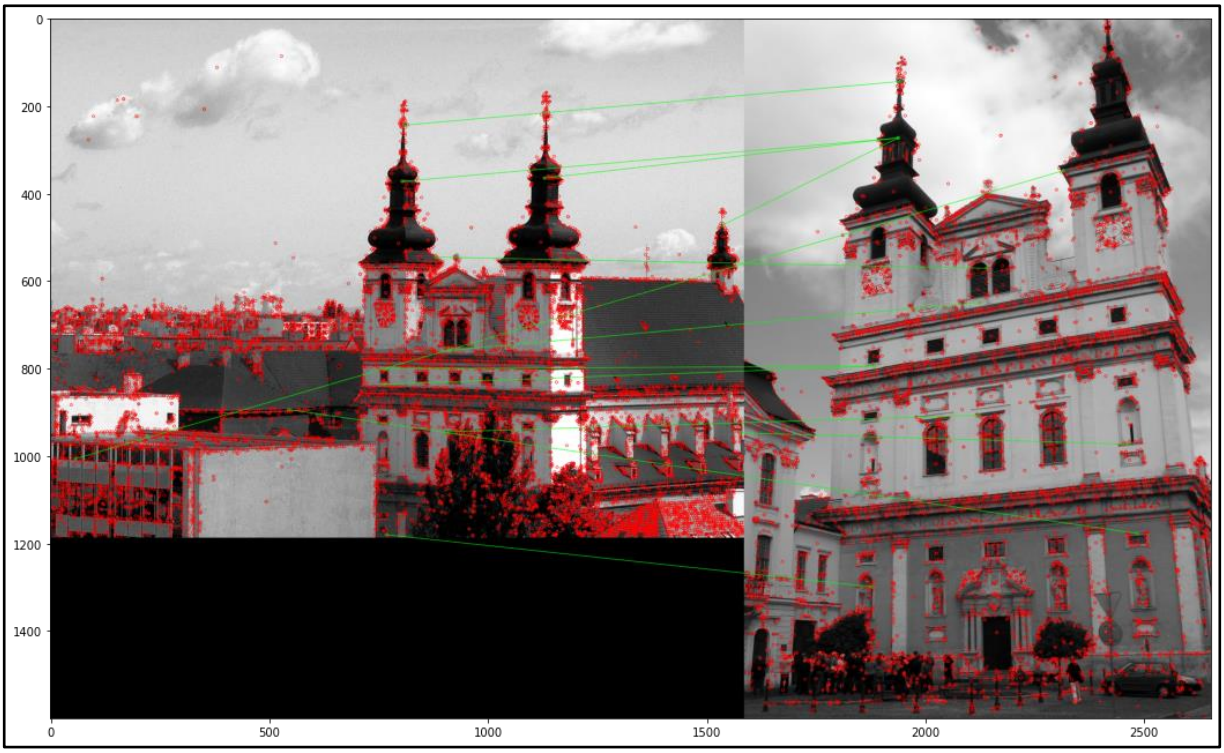

Figura 11. Cruce de keypoints entre dos fotos diferentes del mismo lugar

Una vez con el dataframe de keypoints obtenido se realiza un análisis de componentes principales (pca), para ver si es posible describir esos datos con muchas menos columnas, donde se encuentra que el número óptimo de componente es 10, realizamos la partición de los datos en un 70% - 30% (30 siendo la parte de test)

```
from sklearn.model_selection import train_test_split

Xtr, Xts, ytr, yts = train_test_split(X,Y,test_size=.3)
Xtr.shape, Xts.shape, ytr.shape, yts.shape
```

```
((1304, 2048), (559, 2048), (1304,), (559,))
```

Figura 12. Partición del dataset

Para esta línea base se utiliza el modelo de naive bayes gaussiano de la librería de sklearn (GaussianNB), el cual realiza la clasificación multiclase entre keypoints, para obtener así el modelo entrenado y posteriormente las predicciones contra los datos de prueba.

Para la clasificación del problema con este modelo, se toma como métrica de desempeño el mean accuracy la cual es severa a la hora de medir el desempeño de problemas de clasificación multiclase como el presentado en este proyecto, ya que se requiere que para cada muestra cada etiqueta sea predicha correctamente.

4.2 Modelo de deep learning:

Para el modelo de deep learning implementando el pipeline parte de la descarga de un modelo pre-entrenado del hub de Tensorflow con la idea de hacer transfer learning y aprovechar la capacidad de modelos previamente entrenados con miles o millones de imágenes, para este proyecto se utiliza el modelo MobileNet_v2. Luego de descargar el módulo se continúa por organizar el dataset haciendo un data augmentation usando la función imageDataGenerator de la librería de tensor flow (se deja una variable do_data_augmentation por si se quiere probar el modelo con el aumento de datos y sin el aumento de datos).

```
do_data_augmentation = False
if do_data_augmentation:
    train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rotation_range=40,
        horizontal_flip=True,
        widht_shift_range=0.2, height_shift_range=0.2,
        shear_range=0.2, zoom_range=0.2,
        **datagen_kwargs)
```

Figura 13. Aumento de datos

posteriormente se construye el modelo con sequential en el cual especificamos la capa de input que debe ser igual a la que aceptar el módulo especificado, se agregan dos capas de dropout para no sobreentrenar el modelo, tres capas densas con activación tipo relu y la capa final que debe tener el mismo número de salidas que el número de diferentes locaciones en las imágenes del dataset (se utiliza un regularizador L2).

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dropout_2 (Dropout)          (None, 1280)              0
_________________________________________________________________
dense_4 (Dense)              (None, 20000)             25620000
_________________________________________________________________
dense_5 (Dense)              (None, 1000)              20001000
_________________________________________________________________
dense_6 (Dense)              (None, 200)               200200
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                2010
=================================================================
Total params: 48,081,194
Trainable params: 48,047,082
Non-trainable params: 34,112
_________________________________________________________________
```

Figura 14. Resumen del modelo

Luego se compila el modelo definiendo como métrica de desempeño el accuracy, junto al cross entropy de tipo categórico (también conocido como softmax loss) como función de pérdida, la cual se utiliza en

problemas de múltiples clases (2 o más) utilizando esta función de pérdida se obtiene la pérdida entre las etiquetas y las predicciones. El optimizador utilizado en la red neuronal es una optimización de momentum, gradiente descendente estocástico ó stochastic gradient descent (SGD, lr=0.005, momentum=0.9).

5. METODOLOGÍA

Se utilizó una metodología iterativa, haciendo una primera iteración lo más rápido posible con el fin de tener cada una de las partes del modelo funcionando y poder posteriormente ir cambiando métodos/técnicas más fácilmente, al completar el proceso, justo al final de todas las iteraciones se obtuvieron dos modelos, el modelo base que realiza la solución del problema con técnicas de aprendizaje automático tradicional y un segundo modelo utilizando transfer learning con una red neuronal de deep learning.

5.1 BASELINE

Al realizar un análisis exploratorio para la primera iteración se encontró que en el dataset habían valores nulos en los diferentes campos, estas muestras serían problemáticas ya que si el valor nulo estaba en el campo url, no se tendría la imagen a ser procesada, y si el nulo viene en el campo landmark_id, de nada serviría tener la imagen si no está el valor que lo asocia a una clase o lugar en específico, por consiguiente, se procedió con la eliminación de estos registros con una reducción de 34.330 en el número total, pasando de tener 1225029 a tener 1190699 registros/imágenes.

Por la capacidad de cómputo y tiempos de ejecución, se decide tomar solo un subset de el total de muestras, y para atacar el problema del desbalance se extrae este subconjunto de una manera cuidadosa para evitar problemas a la hora de entrenar el modelo, por lo cual finalmente se toman 10 clases diferentes de los lugares que cuentan con más imágenes, y de cada uno de estos lugares se tomaron 200 registros para tener un total de dos mil datos.

Luego de tener los datos seleccionados se debió proceder a descargar la totalidad imágenes, aquí se presentó un pequeño inconveniente ya que al estar todos estos recursos guardados en la web, el proceso de descarga puede fallar por diferentes motivos, en el experimento realizado se obtuvo un total de 63 imágenes que no pudieron ser descargadas por qué las URLs habían caducado.

Con las imágenes ya descargadas se procedió a pasar cada una de ellas por métodos de extracción de características, método ya implementado por la librería de apoyo que se utilizó durante el proceso, y que consiste en la transformación de las imagenes a vectores de valores numéricos donde se enfatiza el análisis sobre los puntos clave de la imagen, dejando un vector de 2049 características por cada imagen.

Dado que el volumen de los datos sigue siendo bastante alto, se utiliza sobre estos un análisis de componentes principales (PCA por sus siglas en inglés), método que sirve para reducir la dimensionalidad de los datos dejando los mismos descritos por nuevas variables no correlacionadas.

Por último, se realiza la partición de los datos para luego pasar la matriz resultante por un modelo GNB (Gaussian Naive Bayes) para su entrenamiento y posterior validación con un score y una matriz de confusión para ver de una manera más gráfica cómo se están comportando las predicciones.

5.2 ITERACIONES y EVOLUCIÓN

Se realizó un par de interacciones más con el mismo proceso que el primero (ML tradicional), variando la manera de extraer los keypoints (método kaze y método orb), variando la cantidad de imágenes sin ser esto una cambio considerable por las limitantes de procesamiento, también se prueban diferentes cantidades de componentes resultantes al reducir la dimensionalidad con el método PCA y otros modelos de clasificación como el árbol de decisión, sin embargo los resultados siempre entregaron modelos a los cuales les costaba generalizar el problema.

Para aumentar considerablemente estos resultados en posteriores iteraciones se decidió cambiar drásticamente la metodología y utilizar un modelo basado en redes neuronales convolucionales de deep learning, se realiza transfer learning con un modelo pre entrenado con el fin de usar una red que haya sido entrenado de una manera más potente en el problema de clasificar imágenes y aprovechar esas capas para obtener los resultados buscando en el problema en específico.

Adicional en una iteración adicional, también se decidió realizar una anonimización de rostros con dos métodos diferentes que llevan a cabo este proceso, todo con el fin de guardar la privacidad de las personas. Dicho proceso fue realizado con ayuda de la librería opencv para Python y constaba de dos pasos, el primero de ellos, a través de un modelo de red neuronal profunda pre-cargado, se recorren las imágenes una a una con el fin de detectar en cuales de ellas habían rostros presentes y con cuanto porcentaje de confiabilidad, y ya con los rostros identificados se procede a anonimizar esta sección de las imágenes, para el cual se utilizaron dos métodos diferentes, uno de blur o difuminado y otro más fuerte con pixelado.

A continuación se presentan varios ejemplos de anonimización con el proceso de difuminado, teniendo a la izquierda la imagen original y a la derecha la imagen luego de pasarla por el proceso:

Figura 15. Ejemplo 1 anonimización con método de difuminado

Figura 16. Ejemplo 2 anonimización con método de difuminado

Figura 17. Ejemplo 3 anonimización con método de difuminado

En esta última imagen se puede observar que hay rostros presentes que no fueron anonimizados y esto se debe a que el modelo no identificó dichos rostros con una confiabilidad mayor al cincuenta por ciento, que fue el límite utilizado para el proceso.

A continuación, se presentan las mismas imágenes pero con el proceso de pixelado, donde se observa también, que el problema en la última imagen persiste ya que el proceso para reconocer los rostros fue el mismo para ambos métodos, sin embargo esta técnica de anonimización es mucho más fuerte dejando el rostro totalmente irreconocible.

Figura 18. Ejemplo 1 anonimización con método de pixelado

Figura 19. Ejemplo 2 anonimización con método de pixelado

Figura 20. Ejemplo 3 anonimización con método de pixelado

5.3 HERRAMIENTAS

El lenguaje de programación que se utilizó durante todo el proceso fue Python, ya que es un lenguaje comúnmente utilizado para la implementación de modelos de inteligencia artificial, cuenta con una amplia comunidad y un gran número de librerías que facilitan el proceso al tener implementados métodos y modelos en su totalidad con apoyo de librerías como sklearn, tensor flow, caffe, entre muchas otras.

La herramienta principal que se utilizó para la realización de los experimentos es Google Colab, que es un servicio de Google que provee la infraestructura para la ejecución del código de manera gratuita en la web lo que permite trabajar de manera colaborativa y desligarse del hardware a la hora de procesar la información y entrenar los modelos.

También es importante mencionar que en algunas iteraciones de prueba para realizar el despliegue del modelo se utilizó la suite de azure-ml.

Adicional a esto se utilizaron diferentes librerías de Python durante todo el proceso como:

- Numpy
- Pandas
- OpenCV
- Tensorflow
- urllib
- tensorflow_hub
- Sklearn
- pydrive
- cv2
- PIL
- matplotlib
- scipy
- pickle
- itertools

6. RESULTADOS

Modelo de machine learning clásico:

Para la toma de resultado de la línea base (GaussianNB) primero se obtuvieron todas las predicciones de los datos de entrenamiento y de test luego medimos el accuracy con el método score que viene en la clase de gaussianNB, se hizo tanto para el modelo sin PCA, como para el modelo con PCA obteniendo mejores resultados en el modelo con PCA (los resultados siempre están alrededor del 0.27 de accuracy en la fase entrenamiento y un 0.23 de accuracy en la fase validación), estos resultados fueron poco satisfactorios puesto que clasificar con un porcentaje tan bajo de confianza no es garantía para la toma de decisiones o el despliegue de un algoritmo funcional.

```
pca = PCA(n_components=5)
pca.fit(Xtr)

Xt_tr = pca.transform(Xtr)
Xt_ts = pca.transform(Xts)
dt.fit(Xt_tr,ytr)
ypreds_tr = dt.predict(Xt_tr)
ypreds_ts = dt.predict(Xt_ts)
ypreds_tr.shape, ypreds_ts.shape
np.mean(ytr==ypreds_tr),np.mean(yts==ypreds_ts)
```

```
(0.26630851880276285, 0.259391771019678)
```

Figura 21. Aplicación de PCA

También se realiza la predicción sin aplicar PCA el cual muestra unos resultados muy buenos en el conjunto de datos de entrenamiento (0.7382), sin embargo peores resultados en los datos de prueba (0.1896).

```
dt = GaussianNB()
dt.fit(Xtr, ytr)
dt.score(Xtr, ytr), dt.score(Xts, yts)
```

```
(0.7382962394474291, 0.18962432915921287)
```

Figura 22. Ajuste del modelo GaussianNB

Esto se puede observar de una mejor manera al observar el comportamiento del modelo con diferentes números de características principales entregadas por el PCA:

```
plt.plot(cs, dtr, label="train")
plt.plot(cs, dts, label="test")
plt.xlabel("n components")
plt.ylabel("% acierto")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8057ecb5d0>
```

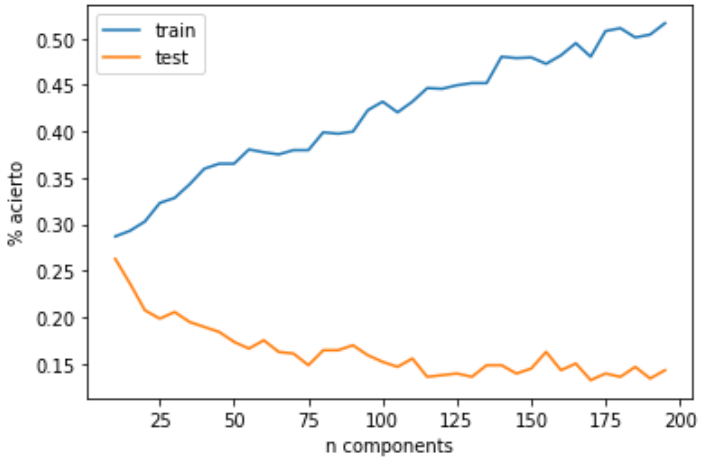

Figura 23. % Acierto de acuerda a la cantidad de características principales

Adicionalmente se probaron también otros clasificadores de ML como el árbol de decisiones, pero la baja generalización persiste incluso obteniendo peores resultados que el GaussianNb, como se puede observar en la siguiente imagen:

```
[52] from sklearn.naive_bayes import GaussianNB
     from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier()
     dt.fit(Xtr, ytr)
     dt.score(Xtr, ytr), dt.score(Xts, yts)

     (1.0, 0.1037567084078712)
```

Figura 24. Prueba del modelo Decision tree classifier

A continuación se puede observar en la figura 25 la matriz de confusión normalizada y sin normalizar del modelo base. Debajo de las imágenes de las matrices se puede observar el resultado general del accuracy y que tanto está fallando las clasificaciones de las clases el modelo, 26% de accuracy y 73% de fallo en la clasificación lo cual nuevamente confirma la baja generalización del modelo.

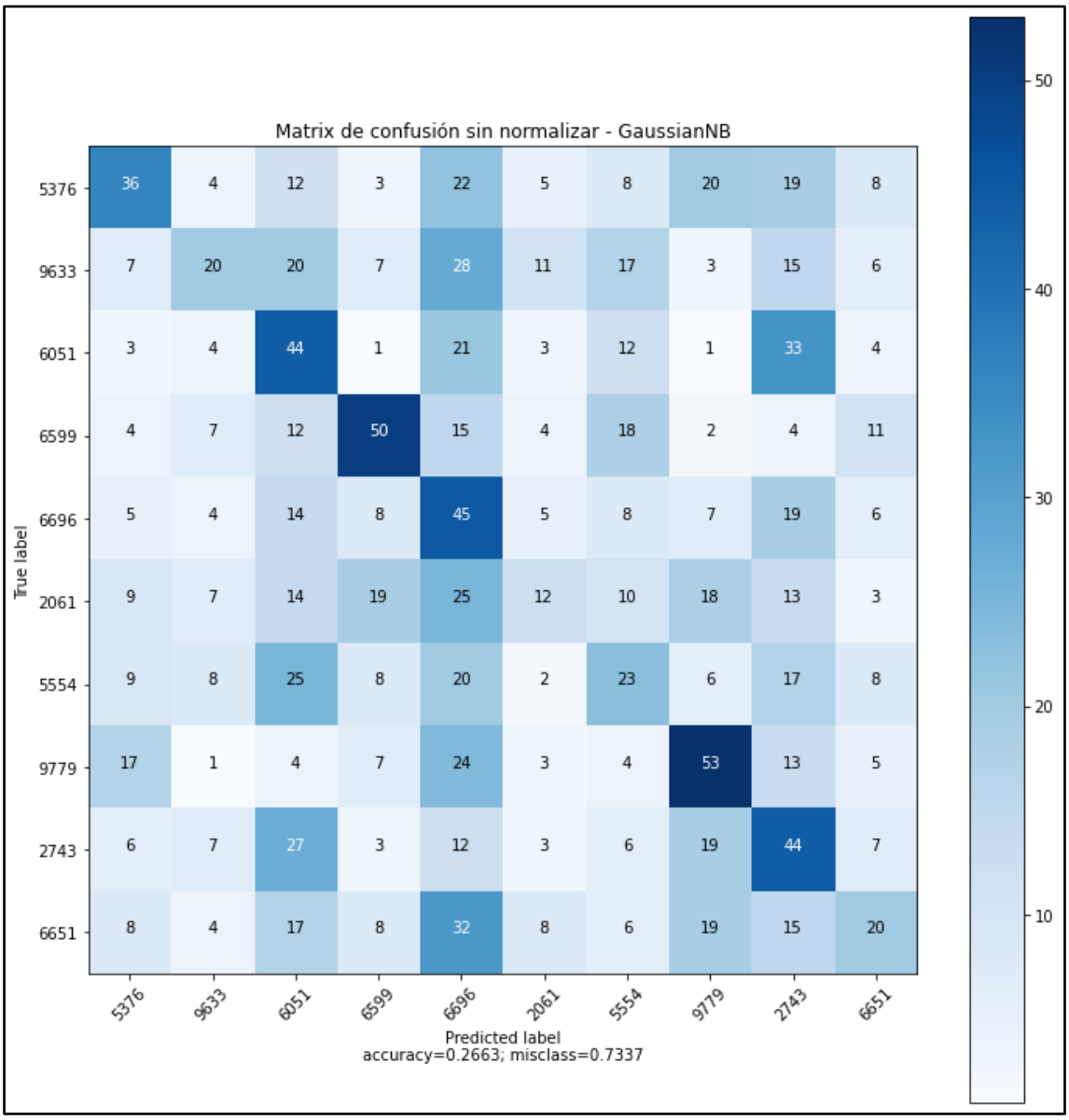

Figura 25. Matriz de confusión sin normalizar - GaussianNB

En la matriz de confusión normalizada de la figura 26 se puede apreciar más fácilmente el porcentaje de acierto en cada clase de acuerdo a la diagonal principal y el fallo de clasificación en el resto de recuadros.

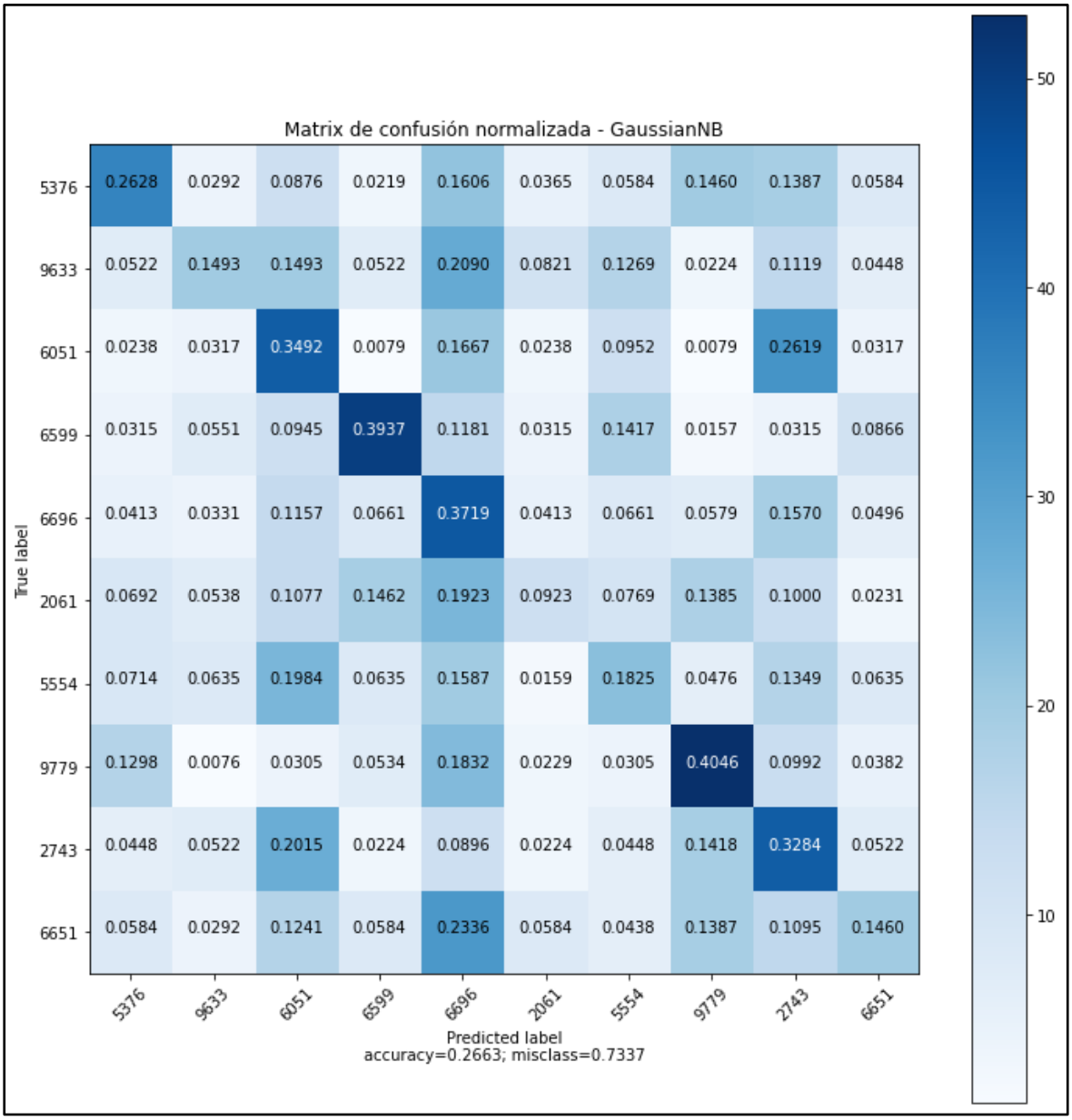

Figura 26. Matriz de confusión normalizada - GaussianNB

Para los 10 lugares clasificados también se puede observar en la figura 27 un reporte donde se muestran el comportamiento de las métricas de precisión, recall y F1 para cada una de las clases clasificadas:

	precision	recall	f1-score
5376	0.23	0.35	0.28
9633	0.23	0.33	0.27
6051	0.18	0.37	0.25
6599	0.21	0.09	0.13
6696	0.36	0.40	0.38
2061	0.35	0.26	0.30
5554	0.44	0.39	0.41
9779	0.26	0.15	0.19
2743	0.30	0.15	0.20
6651	0.21	0.18	0.19

Figura 27. Reporte de métricas del modelo GaussianNB

Modelo de deep learning:

En el modelo final se utiliza la red pre-entrenada de MobileNetV2 y a la hora de compilación del modelo se utiliza la métrica de accuracy, junto a la función de pérdida categoricalCrossEntropy y cómo optimizar el stochastic gradient descent:

```
model.compile(
    optimizer = tf.keras.optimizers.SGD(lr=0.05, momentum=0.9),
    loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True, label_smoothing=0.1),
    metrics = ['accuracy']
)

steps_per_epoch = train_generator.samples // train_generator.batch_size
validation_steps = valid_generator.samples // valid_generator.batch_size
hist = model.fit(train_generator,
                 epochs = 20,
                 steps_per_epoch = steps_per_epoch,
                 validation_data = valid_generator,
                 validation_steps = validation_steps).history
```

Figura 28. Función de pérdida, optimización, métricas y entrenamiento del modelo de deep learning

Se puede observar a la hora de entrenar la red como esta cambia a medida que pasan las 20 épocas definidas y cómo se va ajustando cada vez mejor al problema en específico hasta obtener unos resultados de generalización sobre el problema interesantes del 88% de accuracy en el conjunto de validación y la función de pérdida se baja de un 7 inicial a 0.94 al final del entrenamiento, esto se puede ver más claramente en la figura 29.

```
Epoch 1/20
46/46 [==============================] - 124s 3s/step - loss: 1.7531 - accuracy: 0.5421 - val_loss: 7.4111 - val_accuracy: 0.1165
Epoch 2/20
46/46 [==============================] - 114s 2s/step - loss: 1.4717 - accuracy: 0.6749 - val_loss: 6.4357 - val_accuracy: 0.1051
Epoch 3/20
46/46 [==============================] - 114s 2s/step - loss: 1.3519 - accuracy: 0.7379 - val_loss: 9.4059 - val_accuracy: 0.1051
Epoch 4/20
46/46 [==============================] - 115s 2s/step - loss: 1.2523 - accuracy: 0.7974 - val_loss: 8.2929 - val_accuracy: 0.1250
Epoch 5/20
46/46 [==============================] - 115s 3s/step - loss: 1.0288 - accuracy: 0.8611 - val_loss: 2.6257 - val_accuracy: 0.4148
Epoch 6/20
46/46 [==============================] - 115s 3s/step - loss: 1.0282 - accuracy: 0.8611 - val_loss: 2.8395 - val_accuracy: 0.4545
Epoch 7/20
46/46 [==============================] - 115s 3s/step - loss: 0.9992 - accuracy: 0.8761 - val_loss: 3.3470 - val_accuracy: 0.2756
Epoch 8/20
46/46 [==============================] - 115s 3s/step - loss: 0.8797 - accuracy: 0.9076 - val_loss: 2.1459 - val_accuracy: 0.5398
Epoch 9/20
46/46 [==============================] - 114s 2s/step - loss: 0.8556 - accuracy: 0.9185 - val_loss: 1.6908 - val_accuracy: 0.5540
Epoch 10/20
46/46 [==============================] - 113s 2s/step - loss: 0.8003 - accuracy: 0.9425 - val_loss: 2.0346 - val_accuracy: 0.5227
Epoch 11/20
46/46 [==============================] - 113s 2s/step - loss: 0.7596 - accuracy: 0.9555 - val_loss: 1.1830 - val_accuracy: 0.8097
Epoch 12/20
46/46 [==============================] - 113s 2s/step - loss: 0.7053 - accuracy: 0.9747 - val_loss: 1.4380 - val_accuracy: 0.6903
Epoch 13/20
46/46 [==============================] - 113s 2s/step - loss: 0.7437 - accuracy: 0.9658 - val_loss: 1.5840 - val_accuracy: 0.6676
Epoch 14/20
46/46 [==============================] - 113s 2s/step - loss: 0.7081 - accuracy: 0.9719 - val_loss: 1.1154 - val_accuracy: 0.8097
Epoch 15/20
46/46 [==============================] - 113s 2s/step - loss: 0.7007 - accuracy: 0.9754 - val_loss: 1.3129 - val_accuracy: 0.7301
Epoch 16/20
46/46 [==============================] - 113s 2s/step - loss: 0.7139 - accuracy: 0.9740 - val_loss: 1.2909 - val_accuracy: 0.7528
Epoch 17/20
46/46 [==============================] - 113s 2s/step - loss: 0.7225 - accuracy: 0.9660 - val_loss: 1.1362 - val_accuracy: 0.8068
Epoch 18/20
46/46 [==============================] - 113s 2s/step - loss: 0.6808 - accuracy: 0.9856 - val_loss: 1.0754 - val_accuracy: 0.8210
Epoch 19/20
46/46 [==============================] - 113s 2s/step - loss: 0.6620 - accuracy: 0.9856 - val_loss: 0.9057 - val_accuracy: 0.8892
Epoch 20/20
46/46 [==============================] - 112s 2s/step - loss: 0.6480 - accuracy: 0.9911 - val_loss: 0.9400 - val_accuracy: 0.8892
```

Figura 29. Evolución del entrenamiento del modelo de DL

En la figura 30 se observa cómo el comportamiento de la función de pérdida tienda a bajar considerablemente tanto para el entrenamiento (línea azul), como para los datos de validación (línea naranja):

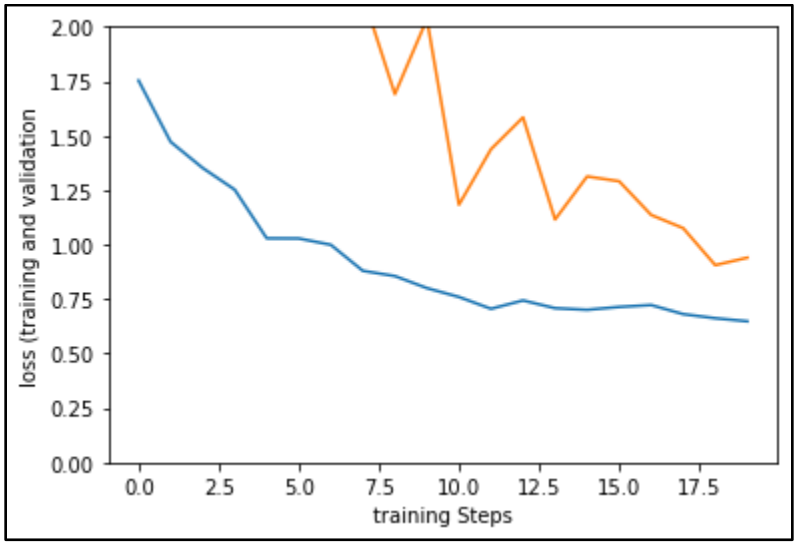

Figura 30. Evolución de la función de pérdida por épocas

En la figura 31 también se muestra una gráfica con la evolución de la métrica de accuracy a medida que pasan las épocas de entrenamiento de la red neuronal, la línea azul representa el comportamiento sobre los datos de entrenamiento y la línea naranja sobre los datos de validación

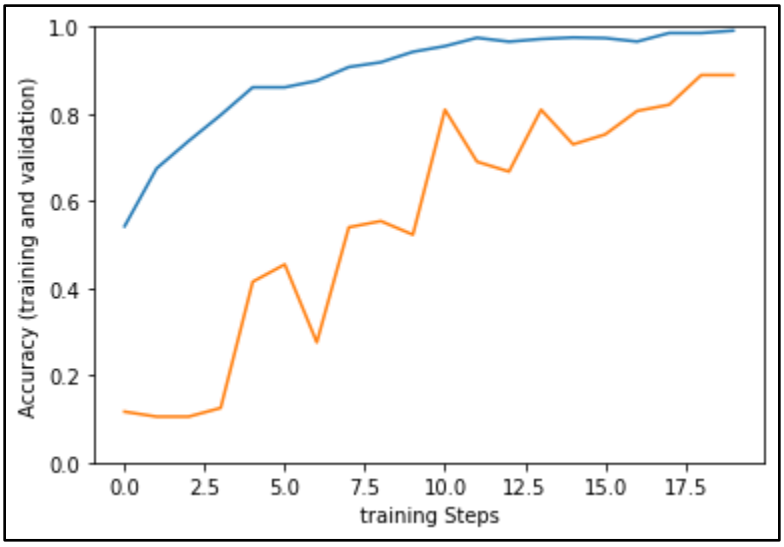

Figura 31. Evolución del accuracy por época

En la figura 32 se tiene la matriz de confusión sin normalizar del modelo de deep learning.

Debajo de las imágenes de las matrices se puede observar el resultado general del accuracy y que tanto está fallando las clasificaciones de las clases el modelo, con un 88% de accuracy y 11% de fallo en la clasificación lo cual evidencia la superioridad de este modelo a la hora de hacer clasificaciones de imágenes respecto al modelo de línea base.

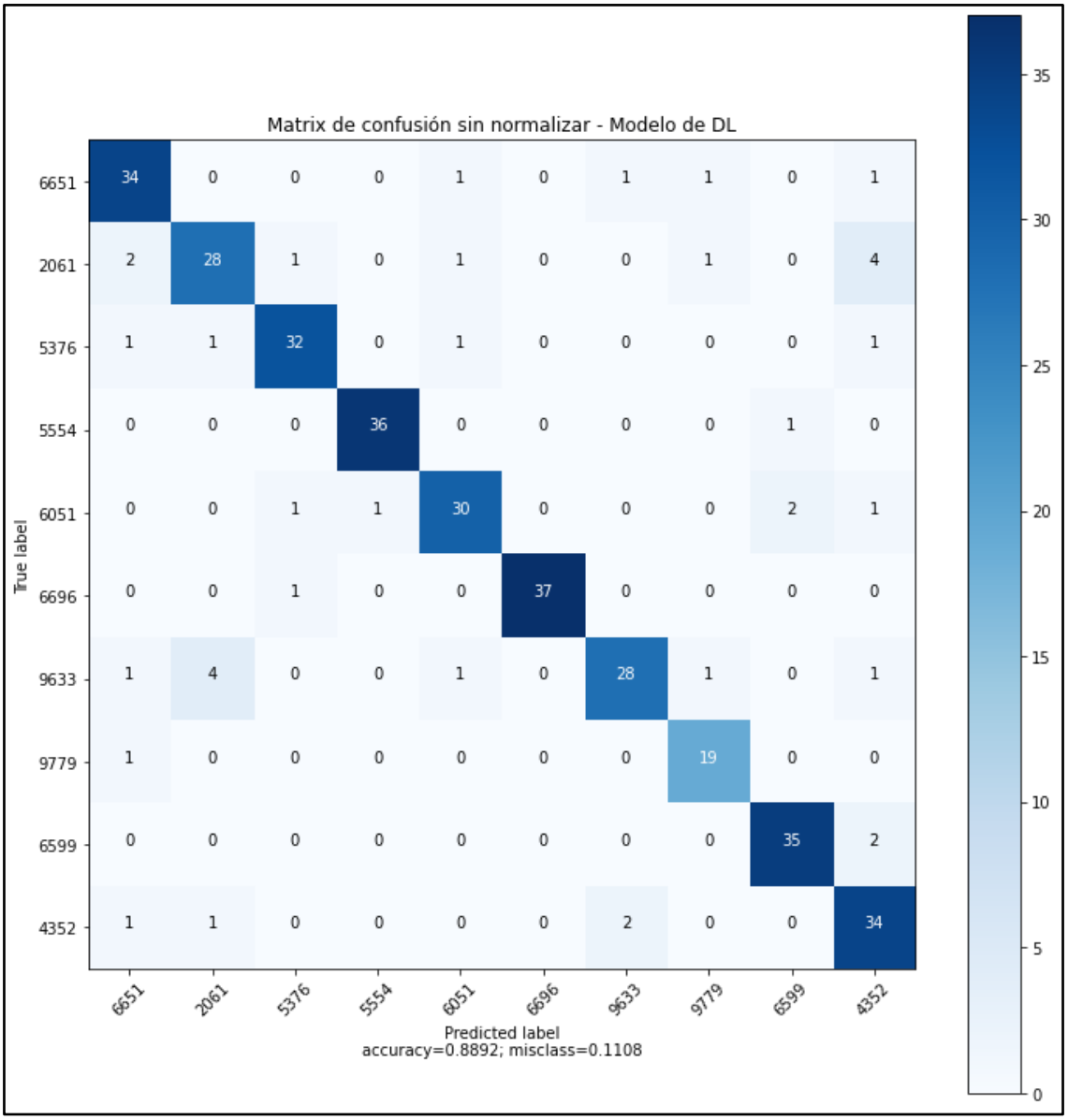

Figura 32. Matriz de confusión sin normalizar - modelo de DL

En la figura 33 se muestra la misma matriz de confusión pero esta vez con los datos normalizados, se puede evidencia que la clase donde tuvo una clasificación menor fue del 75% y las clases donde mejor obtuvo resultados llegaron a un accuracy del 97%

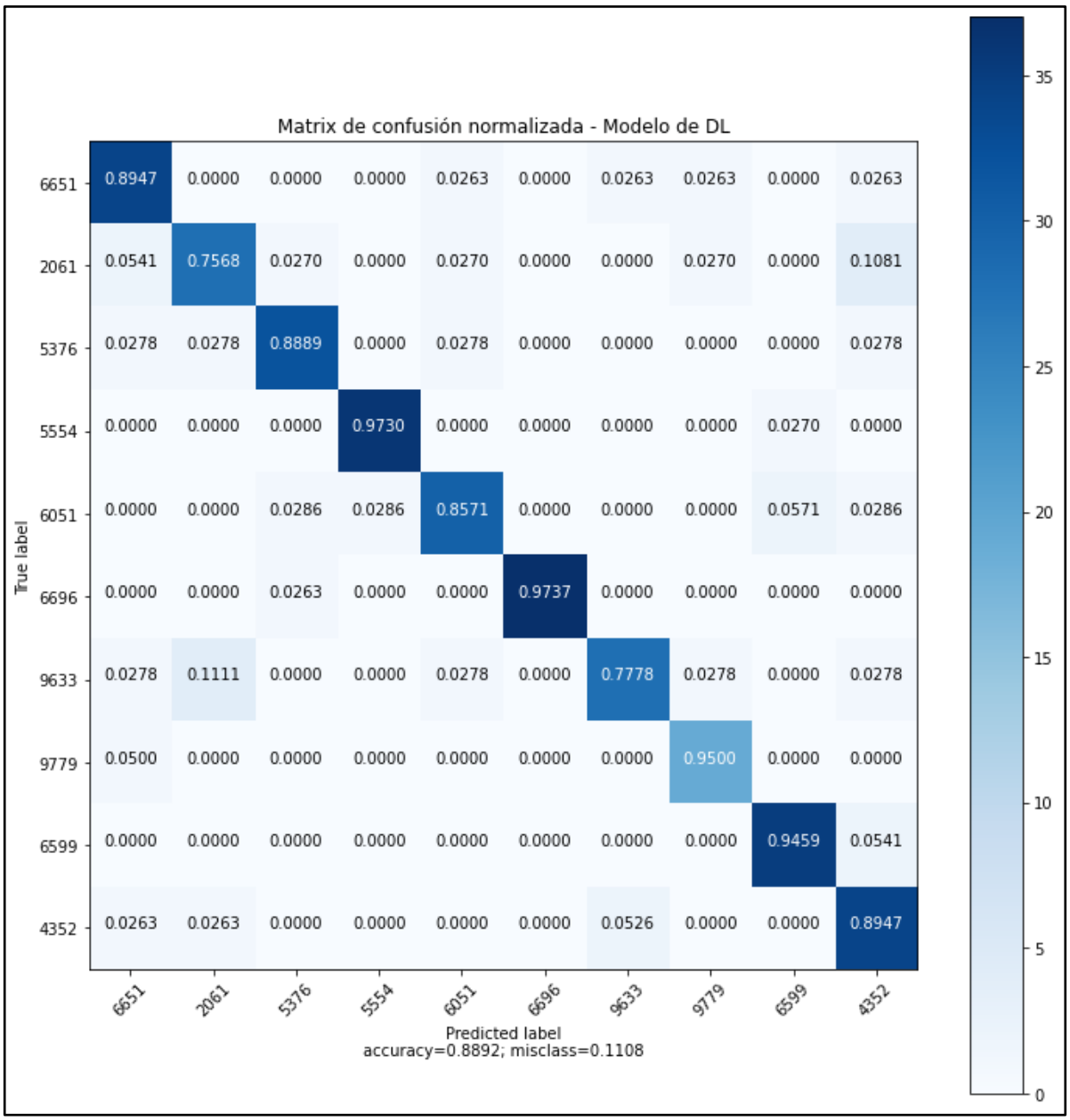

Figura 33. Matriz de confusión normalizada - modelo de DL

Finalmente se tiene el reporte de métricas (precisión, recall y el f1-score), se observan muy buenos resultados en general para el problema planteado en el ejercicio

	precision	recall	f1-score
6651	0.85	0.89	0.87
2061	0.82	0.76	0.79
5376	0.91	0.89	0.90
5554	0.97	0.97	0.97
6051	0.88	0.86	0.87
6696	1.00	0.97	0.99
9633	0.90	0.78	0.84
9779	0.86	0.95	0.90
6599	0.92	0.95	0.93
4352	0.77	0.89	0.83

Figura 34. Reporte de métricas - modelo de DL

6.2 CONSIDERACIONES DE PRODUCCIÓN

Se debe considerar el almacenado del dataset en la nube periódicamente con ingestas en batch y hacer la conexión al entrenamiento modelo para realizar entrenamientos periódicos con nuevos datos cada cierto tiempo y mejorar así su funcionamiento, en el desarrollo del proyecto se guardó un directorio de imágenes en el data-lake de azure (blob storage) y se realizo la conexión al codigo de entrenamiento del modelo que estaba almacenado en azure-ml, esto también permite la posibilidad de hacer experimentos y comparar resultados entre diferentes ejecuciones.

Desplegar un modelo de sklearn o machine learning clásico tiene diferencias importantes comparado con el despliegue un modelo de tensor flow o deep learning, en el primer caso basta con guardar el modelo en un archivo .pkl (pickle), crear un endpoint y enviar datos en formatos json por este endpoint que retornara la predicción. En el caso del modelo de tensorflow, el guardado del modelo se puede hacer de múltiples maneras y en diferentes formatos, también se debe tener en cuenta si se utiliza un modelo pre-entrenado de tensor flow hub, para poder desplegar este modelo con azure-ml fue necesario guardar el modelo en formato .h5 y tener en cuenta que como se utiliza transfer learning a la hora de cargar el modelo se debe hacer de la siguiente manera especificando el hub.kerasLayer:

```
model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'mic_model.h5')
tf.keras.models.load_model(model_path,custom_objects={'KerasLayer':hub.KerasLayer})
```

Figura 35. Carga del modelo guardado en formato .h5

A la hora de envío de datos al ser imágenes, se sugiere que las imágenes sean codificadas en base64 y el endpoint realice la decodificación de la imagen previo a la predicción.

7. CONCLUSIONES

Luego de ver los resultados una de las conclusiones más importantes que deja el desarrollo del proyecto es que generalizar modelos de reconocimiento de imágenes con modelo de machine learning tradicional es complejo, el método de extracción de características tiene resultados interesantes y puede utilizarse para realizar análisis particulares sobre una imagen, también por los resultados obtenidos con la red neuronales pre-entrenada este sería el enfoque recomendando al abordar este tipo de problemas de reconocimiento entre imágenes o lugares sobretodo por que es difícil competir con la cantidad de imágenes con las que estos modelos pre-entrenados han sido entrenados.

El proceso de entrenamiento requiere una capacidad de cómputo considerable, cuando se tienen múltiples clases obtener un modelo con un alto porcentaje de acierto a la hora de clasificar es una tarea compleja, ya que necesita tener un dataset bastante grande y balanceado para que el modelo cumpla con lo deseado, y a medida que aumentan las clases y las muestras de cada clase, el procesamiento se hace más pesado y este demandará mejor máquina y mayor tiempo de ejecución.

El proceso de deep learning que se ejecutó aunque más costoso computacionalmente ya que procesa las imágenes en su formato original y con toda la información que esto posee, es mucho más confiable ya que brinda mayor porcentaje de acierto (accuracy) a la hora de clasificar una imagen entre los sitios que se buscan.

En el proceso de anonimización, ambos métodos (difuminado y pixelado), cumplen con su función perfectamente, pero el método de pixelado es más fuerte al ser aplicado y distorsiona casi por completo el rostro haciendo muy difícil la identificación de una persona en la imagen, se deben realizar ajustes del modelo de acuerda a los rostros contenidos en las imágenes del dataset ya que al ser personas de diferentes regiones, razas o géneros puede producirse un sesgo.

En el despliegue del modelo se debe tener muy en cuenta el tema del guardado en un archivo ya que para modelos de tensor flow hay varias maneras de guardarlo, también si se utiliza una red pre-entrenada tener en cuenta el método de carga de acuerdo a lo especificado y tener en cuenta las complejidades técnicas que conlleva enviar datos de imágenes a través de un endpoint, se puede lograr codificando la imagen o almacenando la imagen en un storage en la nube, para su consulta.

8. BIBLIOGRAFIA

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, & Liang-Chieh Chen. (2018, 01 13). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*.

https://arxiv.org/abs/1801.04381

T. Weyand, A. Araujo, B. Cao, & J. Sim,. (2020). *Google Landmarks Dataset v2 - A Large-Scale Benchmark for Instance-Level Recognition and Retrieval*. Proc ICCV'20

https://www.kaggle.com/c/landmark-recognition-challenge/data