ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

Máster en Big Data: Tecnología y Analítica Avanzada

MACHINE LEARNING II: DEEP LEARNING

Jorge Ayuso Martínez

Francisco Javier Gisbert Gil

Carlota Monedero Herranz

Diego Sanz-Gadea Sánchez

José Manuel Vega Gradit

Madrid
April 2023

Índice

1. Introducción

En esta práctica, se exploran algoritmos de Machine Learning y Deep Learning para clasificar imágenes. Para ello, se proponen dos técnicas: la bolsa de palabras (Bag-of-Words o BoW) y las redes neuronales convolucionales (Convolutional Neural Networks o CNN). El código del proyecto se encuentra en este repositorio de GitHub.

BoW es una técnica de Machine Learning tradicional que se basa en la extracción de características previas a la clasificación de imágenes. Visual BoW (Bolsa de Palabras visual) extrae vectores de características mediante filtros y los agrupa en *clusters* o grupos a través de un algoritmo de aprendizaje no supervisado. Los vectores se reducen a un vocabulario de m *clusters* y se cuentan para crear una firma única para cada imagen. Esto es similar a la comparación de documentos del modelo de BoW de texto, donde se crea un histograma de frecuencia de palabras. Los histogramas normalizados de cada imagen se utilizan como entrada en el modelo de clasificación junto con la etiqueta de clase correspondiente.

Por otro lado, las CNN son una técnica de Deep Learning que se inspira en el funcionamiento del sistema visual humano. Las CNN aprenden automáticamente las características relevantes de las imágenes a través de capas convolucionales, y luego utilizan estas características para clasificar las imágenes.

Además de las técnicas de BoW y CNN mencionadas, de manera adicional hemos entrenado un modelo de Vision Transformers. Esta arquitectura ha ganado una gran popularidad gracias a su alto rendimiento del Procesamiento de Lenguaje Natural, siendo la base de aplicaciones basadas en grandes modelos de lenguaje como ChatGPT. Los transformers son una técnica de Deep Learning que utiliza una arquitectura basada en atención para procesar secuencias de datos, como texto o imágenes. En el caso de las imágenes, los transformers aprenden a atender a diferentes partes de la imagen para extraer características relevantes. Esto se puede entender como un proceso análogo a la atención humana, en el que prestamos más atención a ciertas partes de una imagen que a otras.

Una vez que se han clasificado las imágenes utilizando estas técnicas, se procede a generar imágenes sintéticas utilizando una red generativa antagónica profunda, Deep Convolutional Generative Adversarial Network (DCGAN). Esta técnica de Deep Learning permite generar imágenes sintéticas que parecen auténticas a ojos de observadores humanos, lo que la convierte en una técnica muy interesante para la creación de contenido generado automáticamente.

En resumen, en esta práctica se han explorado técnicas de Machine Learning y Deep Learning para la clasificación de imágenes, y se ha utilizado una red generativa antagónica profunda para generar imágenes sintéticas.

2. Bag of Words

En este apartado exploraremos el modelo de Bolsa de Palabras visual (Visual BoW), un método que combina la extracción de características a partir filtros de con un modelo de Machine Learning tradicional (como puede ser SVM) para realizar la clasificación de imágenes.

2.1. Introducción

El método de BoW se basa en una serie de etapas realizadas de manera secuencial. En primer lugar, se realiza la extracción de n vectores de características de longitud definida mediante filtros (como pueden ser SIFT, SURF, ORB o KAZE). Una vez extraídos los vectores para cada una de las imágenes del conjunto de entrenamiento, se procede a aplicar una técnica de clustering no supervisado que permita reducir el espacio de características que utilizar como input para el modelo de clasificación. De este modo, se consigue reducir los n vectores de características a un *vocabulario* compuesto por m clusters de las mismas. Seguidamente, para cada una de las imágenes, se realiza un conteo del número de características pertenecientes a cada uno de los clusters generados, dando lugar a una *firma* única para cada una de dichas imágenes. Esta aproximación es análoga a la comparación de documentos dentro de un modelo de BoW tradicional, donde se realiza un histograma de frecuencia de las palabras que aparecen en cada uno de los documentos analizados para luego comparar ambos histogramas y determinan la proximidad léxica entre los mismos. Dichos histogramas se encuentran normalizados en cada una de sus bins (lo cual se consigue dividiendo el conteo por el número total de clusters realizados). Finalmente, se emplean el vector con los valores de conteo normalizados generados en el paso anterior como input para el modelo de clasificación, así como la etiqueta de clase correspondiente a cada una de las imágenes.

2.2. Parámetros clave y modificaciones realizadas

Algunos de los parámetros clave a considerar dentro del diseño de un clasificador BoW incluyen:

- **Tamaño del vocabulario:** Un vocabulario visual demasiado pequeño resulta en una generalización demasiado alta, que no permite al algoritmo diferenciar con precisión las características de cada una de las clases. Esto resulta especialmente crítico en el caso de problemas multiclase, especialmente si hay varias clases que resultan similares entre sí, puesto que se requiere un nivel de precisión alto para poder diferenciar detalles críticos a la hora de clasificarlas. Por otra parte, un vocabulario demasiado grande implica un aumento de la posibilidad de *overfitting* del modelo, que impide la generalización a ejemplos no vistos durante el entrenamiento.
- **Feature extractor:** El algoritmo empleado para realizar la extracción inicial de características a partir de las imágenes de entrada resulta igualmente importante, puesto que es en última instancia aquel capaz de determinar los keypoints que van a emplearse para la construcción de los clusters y posteriores variables de input del modelo. El algoritmo más popular para realizar esta tarea es 'SIFT'. Sin embargo, existen otros algoritmos, tales como FAST, ORB, SURF o KAZE que presentan diferencias en su forma de calcular las características. En este proyecto comparamos la extracción de características realizada con los algoritmos SIFT, KAZE y FREAK (en este último empleando FAST para realizar la extracción de los keypoints de la imagen).

- **Tipo de modelo de clasificación:** El modelo escogido para realizar la clasificación de las imágenes resulta del mismo modo importante, puesto que determinados modelos son más adecuados para determinadas tareas. Por ejemplo, en el caso de un problema de clasificación con una frontera lineal, aproximaciones como SVM resultanóptimas, dado que permiten una separación limpia de los grupos. Sin embargo, otros modelos como los basados en árboles resultan ineficientes en casos de frontera lineal, ya que se encuentran limitados a realizar cortes perpendiculares al espacio de entrada de las variables, siendo más adecuados para problemas en los que existen fronteras complejas y no lineales. Por ello, se propone la comparación de 2 tipos de modelo diferentes: 'SVM' y 'Random Forest', ambos implementados en la librería de 'cv2'.

En la realización de este apartado de la práctica se ha seguido un proceso iterativo. En primer lugar se ajustó un modelo de BoW sencillo que pudiera servir como referencia. Después, se implementaron las mejoras anteriormente descritas para tratar de mejorar la calidad del modelo, realizando un seguimiento de los errores para ajustar el modelo de forma progresiva. A continuación describiremos las principales pruebas realizadas.

2.3. Modelo de SVM

2.3.1. Primer modelo

La primera combinación sobre la cual vamos a iterar es una combinación de SIFT como extractor de características y SVM como clasificador, empleando KMeans como algoritmo de clustering. En este caso, emplearemos un tamaño de vocabulario de 100, el tamaño predeterminado.

El rendimiento de este primer modelo es bastante deficiente, dado que se obtiene un **accuracy inferior al 0.6 en el conjunto de entrenamiento**, y de **0.46 en el conjunto de validación**. La inspección visual de los resultados revela que el modelo es capaz de distinguir espacios abiertos de espacios cerrados, pero falla sistemáticamente a la hora de realizar diferenciaciones más finas entre dichos espacios. Por ejemplo, es común que el modelo confunda espacios como Office y Bedroom, Living Room o Store, pero es infrecuente encontrar fallos evidentes tales como confundir Bedroom con Forest. Sin embargo, estas equivocaciones siguen ocurriendo en algunos casos, lo que evidencia que el modelo no se encuentra optimizado.

La aproximación evidente para solucionar este tipo de problema es realizar un aumento del vocabulario, lo cual creemos que permitirá al modelo aprender patrones sutiles en las imágenes y hacer mejores distinciones entre las distintas clases. Empezaremos con un incremento drástico del tamaño de vocabulario (500).

2.3.2. Aumento del tamaño del vocabulario

Para este segundo modelo, incrementamos el tamaño del vocabulario y el número de iteraciones del SVM. Podemos ver como con un vocabulario mucho más extenso hay un aumento significativo (del 88 %) del rendimiento en el conjunto de training. Sin embargo, la mejora de accuracy en el conjunto de validación es tan solo parcial, pasando de un 0,46 a un 0.54. Como era de esperar, un aumento demasiado drástico del tamaño de vocabulario conlleva un incremento sustancial del sobreajuste a los datos, como se puede apreciar en la diferencia de rendimiento entre el conjunto de validación y el conjunto de entrenamiento (**0.54 de accuracy en validación** frente a **0.88 de accuracy en entrenamiento**). Por otra parte, el coste computacional es a su vez mucho más elevado, puesto que el proceso de encontrar los clusters ha consumido aproximadamente media hora, mientras que en el caso anterior pudimos obtener las features en menos de 10 minutos.

2.3.3. Experimentos con diferentes kernels de SVM

En este apartado vamos a explicar las pruebas que realizamos tras el tamaño de vocabulario, que consistieron en probar diferentes kernels de SVM en el clasificador en conjunción con un tamaño de vocabulario intermedio (250 palabras) para reducir el sobreajuste.

- RBF: Se consigue reducir el sobreajuste y se obtiene un ligero aumento del rendimiento en el dataset de validación frente al primer modelo, pero no se observa ninguna mejora significativa frente al segundo modelo (con vocabulario de 500 palabras). La performance es de **0.5 de accuracy en validación** y **0.67 en entrenamiento**.
- Basado en histogramas: Iteremos una vez más con otro tipo de kernel, esta vez basado en la intersección de histogramas. Observamos un rendimiento similar al obtenido con el kernel anterior, obteniéndose un ligero un aumento en la performance del modelo en el conjunto de train, pero este no se ve acompañado por una mejora sustancial de la performance en validation (**0.5 de accuracy en validación** y **0.76 en entrenamiento**).

2.3.4. Experimentos con diferentes matchers

Otra de las opciones a testar es el tipo de 'matcher' que empleamos a la hora de realizar la comparación de las features extraídas en la imagen. Por defecto, la clase 'image_classifier' emplea 'FLANN' como feature matcher, pero podemos comparar los resultados con el uso de otro matcher (en este caso, 'Brute-Force').

Como no hemos observado diferencias significativas en la performance del modelo variando el kernel de SVM, nos decantaremos por mantener el kernel lineal. Además, también trataremos de aumentar ligeramente el vocabulario, pasando de 250 a 300. De nuevo se ve acrecentado el sobreajuste al conjunto de entrenamiento (**0.5 de accuracy en validación** frente a **0.88 de accuracy en entrenamiento**). Es decir, aunque hay un incremento de la performance en el conjunto de train, esto no se ve reflejado en un aumento en la performance en el conjunto de test. Además, es posible que este aumento se deba no al cambio de matcher, sino al cambio en el tamaño del vocabulario.

2.3.5. Cambios en el algoritmo de feature extraction

Otro de los posibles cambios a implementar es utilizar un algoritmo diferente para la extracción de features. Existen muchas opciones a considerar para realizar esto, pero en este caso exploraremos dos: KAZE y FREAK. También tratamos de explorar el uso de SURF, un algoritmo bastante popular, pero en la actualidad se encuentra patentado, por lo que la librería opencv ha dejado de incluirlo y por lo tanto no hemos podido probarlo.

- FAST + FREAK: En el caso de FREAK, usaremos este algoritmo únicamente como extractor de los descriptores, mientras que los keypoints serán extraídos con FAST. El rendimiento es de **0.5 de accuracy en validación** y **0.88 en entrenamiento**, lo cual demuestra un gran sobreajuste de nuevo
- KAZE: Este modelo tiene un rendimiento algo pobre, de (**0.37 de accuracy en validación** y **0.58 en entrenamiento**).

De estas pruebas concluimos que no se obtiene mejora alguna al realizar el cambio de Feature Extractor.

2.4. Modelo de Random Forest

Una vez explorado el abanico de posibilidades ofrecido por diferentes combinaciones de parámetros, procedemos a cambiar el modelo empleado para realizar la clasificación de las imágenes. En este caso, hemos utilizado Random Forest. Los resultados obtenidos son los peores hasta el momento, con un (**0.36 de accuracy en validación** y **0.53 en entrenamiento**)

2.5. Comparación de resultados

Modelo	Vocabulario	Accuracy Train	Accuracy Test	Tiempo
SVM + SIFT	100	0.6	0.47	6 mins
SVM + SIFT	500	0.88	0.54	30 mins
SVM + FAST/FREAK	250	0.88	0.5	15 mins
SVM + KAZE	250	0.59	0.37	17 mins
RF + SIFT	250	0.53	0.36	15 mins

Tabla 1: Resultados de los principales modelos de Bag of Word.

Como podemos observar, el modelo que consigue mejores resultados en el conjunto de validación es el clasificador SVM que emplea SIFT como algoritmo para extraer las features, utilizando un tamaño de vocabulario de 500. Sin embargo, estos resultados se consiguen a costa de un alto nivel de sobre entrenamiento, lo cual sería subóptimo desde el punto de vista de la capacidad de generalización del modelo a datos que no ha visto anteriormente.

3. Clasificación de imágenes

En este apartado, aplicaremos modelos de redes neuronales convolucionales para clasificar imágenes. Exploraremos dos tipos de arquitecturas, las redes neuronales convolucionales y los transformers. Utilizaremos transfer learning, que consiste en utilizar una red neuronal previamente entrenada en una tarea similar para reutilizar su conocimiento en una nueva tarea de clasificación de imágenes. El conjunto de entrenamiento se utilizará para ajustar los pesos de la red y el conjunto de validación se utilizará para evaluar la capacidad de generalización del modelo y prevenir el sobreajuste. Con el objetivo de obtener el mejor rendimiento posible, modificaremos diversos paramétros de la red, y elegiremos el mejor modelo de cara a la competición, así como el que creemos que tiene una mayor capacidad de generalización y que por lo tanto desplegaríamos en un entorno real.

3.1. Análisis Exploratorio de Datos

En primer lugar realizaremos un analisis exploratorio de los datos, respresentando la cantidad de imágenes por clase. Las distribución de las imágenes en el conjunto de entrenamiento (2985 imágenes) y validación (1500 imágenes) es la siguiente:

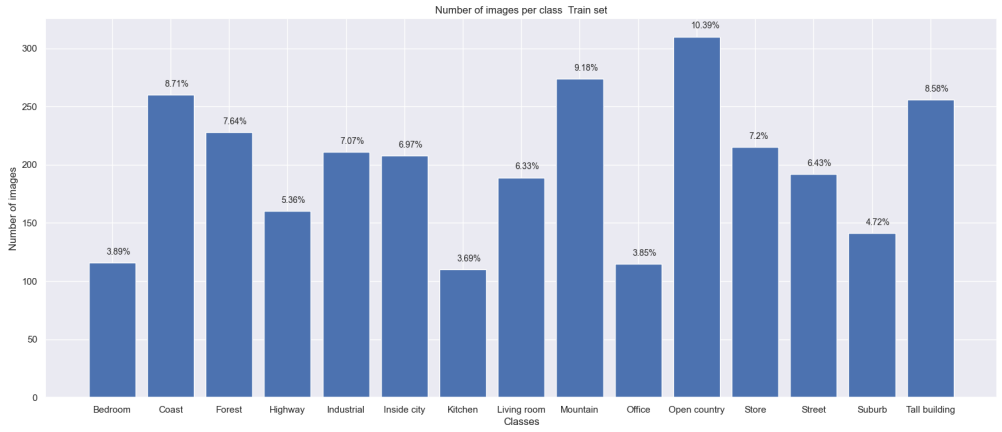

Figura 1: Distribución de las clases de entrenamiento

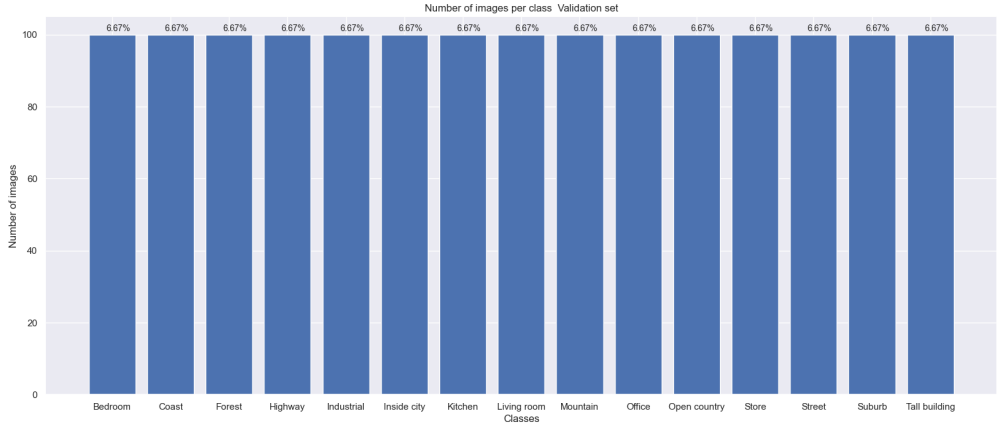

Figura 2: Distribución de las clases de validación

3.2. CNN

Hemos implementado estos modelos en TensorFlow. Si bien existen otras librerías de Deep Learning como Pytorch, la cual usaremos más adelante, hemos decidido utilizar Tensorflow en este apartado para que facilitar la realización de pruebas y generación de logs automática mediante unas ligeras modificaciones sobre el código original. A través de la clase CNN del archivo cnn.py, generamos una carpeta por cada modelo entrenado, donde se guardará la información del modelo en .h5, las gráficas de entrenamiento y los logs. Además se generará un archivo .xlsx con información general de las pruebas de cada modelo, de esta manera podremos comparar los modelos fácilmente.

Hemos realizado pruebas con todos los modelos disponibles, (i) 'DenseNet121' (ii) 'DenseNet169' (iii) 'DenseNet201' (iv) 'InceptionResNetV2' (v) 'InceptionV3' (vi) 'MobileNet' (vii) 'MobileNetV2' (viii) 'NASNetLarge' (ix) 'NASNetMobile' (x) 'ResNet50' (xi) 'VGG16' (xii) 'VGG19' y (xiii) 'Xception'. También hemos modificado parámetros como son el numero de capas finales, el `Drop-Out`, el numero de neuronas por capa, el learning rate, la `Data Augmentation`, el `Batch Size`, la función de activación de cada capa densa antes de llegar a la `softmax` de multiclasificación, descongelación de últimas capas convolucionales, las épocas, la paciencia del `early stopping`. El único hiperparámetro que ha permanecido fijo durante las distintas pruebas es el optimizador. Hemos utilizado `ADAM` puesto que su descenso automático de `learning rate` según se acerca a la solución y el hecho de que sea uno de los optimizadores más usados nos ha convencido.

Se han entrenado 45 modelos con una GPU 6GB `NVIDIA GeForce RTX 3060`, cuyos resultados pueden verse en detalle en GitHub, en el fichero de excel overall_results.

3.2.1. Resultados del entrenamiento

Durante el entrenamiento de los distintos modelos nos hemos dado cuenta de los siguientes aspectos:

- En general, si cogemos muchos lotes provoca que haya más oscilaciones, si se utiliza todo va muy lento o más `batch size`, es mas suave yendo más lenta cada época. No hemos observado que este parámetro mejore los modelos.
- Cuando se aplica el `learning rate` y se para debido a que no se ha mejorado la métrica de monitorización - en este caso la perdida en la validación `val_loss` aunque también se podría modificar a otras métricas del proceso de entrenamiento como es la accuracy en validación `val_acc`- nos quedamos con los pesos de la época que mejor minimización del `val_loss` ha dado lugar.
- Las gráficas de los modelos que se producen al entrenar que monitorizan la perdida y el accuracy en validación y entrenamiento, no corresponden con la métrica de predicción del accuracy sobre el train, debido al `Data Augmentation`. En el entrenamiento se incluye `Data Augmentation` de las imágenes pero en la predicción se ignora y solo se hace sobre las reales. Por otro lado, el valor de predicción de la validación si que corresponde con alguna época existente debido a que en validación no se realiza ningún tipo de `Data Augmentation`.
- En general, el aumento de las épocas sin un `early stopping` muy restrictivo, provoca sobre entrenamiento, generando un modelo menos generalizador.

- La mayor diferencia en la obtención de los mejores modelos radica en el modelo base. Los que mejor se han comportando ha sido `InceptionResNetV2` y `MobileNetV2`, así como variar el numero de neuronas.
- La modificaciones de las funciones de activación en la que hemos probado la `leaky relu` la cual puede tomar valores negativos no han marcado una diferencia.
- Añadir excesivas capas no produce una mejora en el modelo, genera mayor sobre entrenamiento en nuestras pruebas.
- La modificaciones del `Data Augmentation` no han supuesto una mejora reseñable desde las características iniciales, pese a que pensábamos que podría ser un elemento que variar, importante para sacar mejores modelos, la configuración original es la que mejor resultados nos ha dado.
- El modelo con mayor numero de parámetros es el `InceptionResNetV2`. Y el de menor el `MobileNetV2`. Aun así el `MobileNetV2` se ha comportado de los mejores.

De entre todos los modelos entrenados, hemos escogido dos:

Models	Acc Train	Acc Val	Layers (DropOut)	Epochs	Early Stopping	LRate	Time
InceptionResnet_10	0.971	0.926	7 (0.1)	70	15	0.0001	54.34m
InceptionResnet_18	0.945	0.917	7 (0.1)	70	15	0.00001	60.75m

Para mas información de las características del modelo, estan definidas en el .xlsx de pruebas de Github overall_results.

En primer lugar, tenemos el modelo que mejor accuracy obtiene en el set de validación, **InceptionResnetV2_10**, el cual es el elegido para la competición o reto al maximizar la accuracy en validación. En segundo lugar, tenemos **InceptionResnetV2_18** el modelo que debería tener un mejor desempeño en caso de ser puesto en producción, ya que es el que presenta un menor sobreajuste - su accuracy en validación y entrenamiento son los más balanceados.

3.2.2. Análisis de entrenamiento de CNN

En esta sección, haremos un análisis de errores para diagnosticar los modelos escogidos.

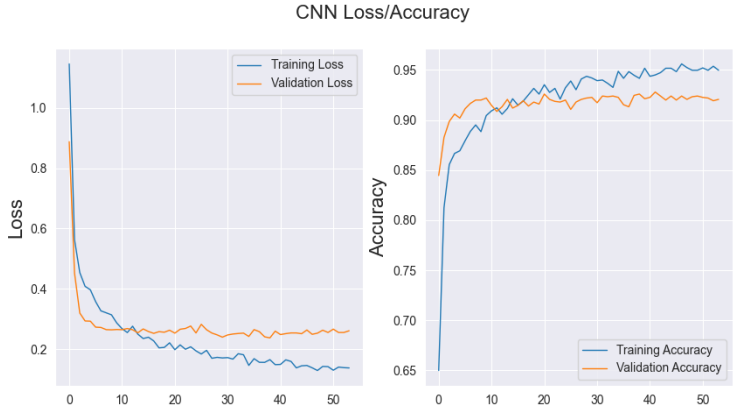

Figura 3: Evolución Loss/Accuracy en el entrenamiento para InceptionResNetV2_10

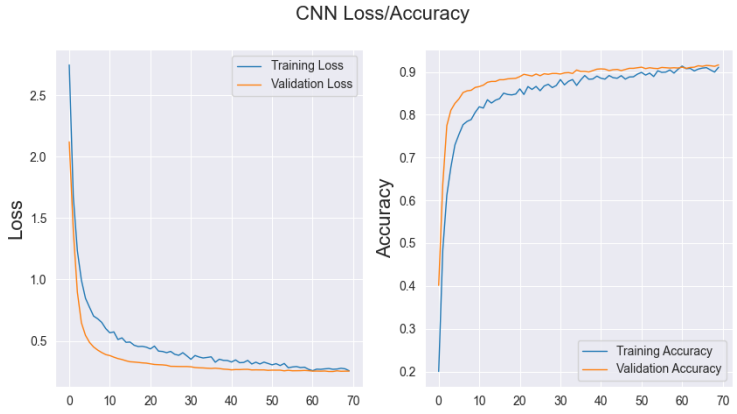

Figura 4: Evolución Loss/Accuracy en el entrenamiento para InceptionResNetV2_18

En las imágenes (3) y (4) vemos claramente como el modelo `InceptionResNetV2\_18` al tener un `learning rate` menor, consigue llegar pausadamente hasta el valor óptimo con muy poco overfitting. Sin embargo, pese a tener una accuracy mayor en validación, vemos como el modelo `InceptionResNetV2_10` satura en dicha métrica por la época 10 y desde ese punto comienza a overfittear.

Ahora analizaremos sus puntos débiles centrándonos en las clases donde menos acierta y entraremos en detalle de algunas de estas imágenes. Comenzamos mostrando la matriz de confusión de modelo `InceptionResnetV2 18`.

KNOWN/PREDICTED	Bedroom	Coast	Forest	Highway	Industrial	Inside city	Kitchen	Living room	Mountain	Office	Open country	Store	Street	Suburb	Tall building
Bedroom	85	0	0	0	0	0	0	15	0	0	0	0	0	0	0
Coast	0	93	0	2	1	0	0	0	1	0	3	0	0	0	0
Forest	0	1	89	0	0	0	0	0	0	0	10	0	0	0	0
Highway	0	0	0	97	0	0	0	0	0	0	0	0	3	0	0
Industrial	0	0	0	1	95	1	0	0	1	0	1	0	0	0	1
Inside city	0	0	0	2	0	87	0	0	0	0	0	1	4	1	5
Kitchen	0	0	0	0	0	1	96	2	0	0	0	1	0	0	0
Living room	7	0	0	0	0	0	1	88	0	4	0	0	0	0	0
Mountain	0	0	0	0	0	0	0	0	96	0	4	0	0	0	0
Office	0	0	0	0	0	0	2	4	0	92	0	2	0	0	0
Open country	0	10	3	2	0	0	0	0	5	0	79	1	0	0	0
Store	0	0	0	0	0	2	1	1	0	0	0	95	1	0	0
Street	0	0	0	0	0	5	0	0	0	0	0	0	95	0	0
Suburb	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
Tall building	0	0	0	0	0	2	0	0	0	0	0	0	0	0	98

Figura 5: matriz de confusión de modelo InceptionResnetV2_18

Vemos Fig.(5) que el modelo se equivoca principalmente cuando la clase real es Bedroom, prediciendo Living Room, cuando la clase real es Forest, prediciendo Open Country y cuando la clase real es Open Country prediciendo Coast. Dado el parecido de estas clases, no es sorprendente que el modelo pueda fallar en algunas imágenes dudosas.

Si analizamos las probabilidades de cada clase para algunas imágenes concretas, podemos diferenciar dos casos. Cuando se equivoca rotundamente y asigna una probabilidad cercana a 1 a la clase equivocada, o cuando predice con un 0.5 para la clase correcta y otro 0.5 para la clase equivocada. Veamos algunos ejemplos ilustrativos para analizar estos errores.

Las imagenes de la figura (6) son un ejemplo de aquellas donde el modelo duda entre varias de las clases y finalmente se equivoca.

Figura 6: Imagenes 27 y 275

Para la imagen 27, la etiqueta real es Bedroom, pero el modelo predice un 0.07 Bedroom, 0.31 Kitchen y 0.62 Living. Posiblemente por tratarse de una habitación muy recargada de muebles y plantas.

Para la imagen 275, la etiqueta real es Coast, pero el modelo predice un 0.37 Coast y un 0.61 Open Country. Esta vez el modelo duda porque tenemos elementos en la imagen que podrían pertenecer a ambas clases, incluso un humano podría dudar al clasificar esta imagen.

Figura 7: Imagenes 69 y 55

Las imágenes de la figura (7) son un ejemplo de aquellas donde el modelo falla rotundamente asignando una probabilidad muy alta a la clase equivocada.

Para la imagen 69, la etiqueta real es Coast, pero el modelo predice un 0.87 Highway por la presencia del puente, además como el agua es una textura muy uniforme, puede que falle al no detectar bordes propios de costas.

Para la imagen 55, la etiqueta real es Bedroom, pero el modelo predice 0.98 para LivingRoom. Este caso es dudoso porque ni siquiera nosotros diríamos que esta imagen es de una habitación, ya que a diferencia de estas imágenes, en esta no vemos niguna cama o elementos propios de este tipo de estancia.

Podemos concluir por tanto que muchos de estos problemas derivan de querer clasificar una imagen dentro de una determinada etiqueta y puede haber problemas si aparecen en la misma imagen elementos típicos de otras clases o simplemente que una imagen debería pertenecer a varias clases.

3.3. Vision Transformer

3.3.1. Introducción

De manera adicional a las redes neuronales convolucionales, se ha entrenado un modelo de Vision Transformers. La arquitectura Transformer, que fue presentada en el artículo 'Attention is All You Need' en 2017, ha revolucionado el mundo del Deep Learning, especialmente en el campo del Procesamiento del Lenguaje Natural. Como un gran modelo de lenguaje basado en la arquitectura GPT-3.5, ChatGPT es la aplicación basada en la arquitectura Transformer más popular del momento. Además de ChatGPT, muchas otras aplicaciones reconocidas, como BERT de Google, la serie GPT de OpenAI y RoBERTa de Facebook, se basan en la arquitectura Transformer para lograr resultados de vanguardia en tareas de NLP. Además, la arquitectura Transformer también ha tenido un gran éxito en el campo de la Visión por Computador, como lo demuestra el éxito de modelos como ViT y DeiT en ImageNet y otros benchmarks de reconocimiento visual.

La principal innovación de la arquitectura Transformer es la combinación del uso de representaciones basadas en atención y un estilo de procesamiento similar al de una red neuronal convolucional (CNN). A diferencia de las redes neuronales convolucionales tradicionales (CNN) que se basan en capas convolucionales para extraer características de las imágenes, los Transformers utilizan mecanismos de atención (auto-atención, atención multi-cabezal) para enfocarse selectivamente en diferentes partes de una secuencia de entrada.

La principal ventaja de los Transformers sobre las CNN tradicionales es que pueden capturar de manera más efectiva las dependencias a largo plazo en los datos. Esto es especialmente útil en tareas de visión por computadora donde una imagen puede contener objetos que están dispersos por toda la imagen, y donde las relaciones entre objetos pueden ser más importantes que los propios objetos. Al atender a diferentes partes de la imagen de entrada, los Transformers pueden aprender eficazmente a extraer estas relaciones y mejorar el rendimiento en tareas como la detección y segmentación de objetos.

3.3.2. Implementación

Para la implementación del modelo hemos utilizado un modelo pre-entrenado de la librería HuggingFace, en conjunción con Pytorch Lightning. El modelo ha entrenado sido entrenado en una GPU 6GB `NVIDIA GeForce GTX 1070`.

HuggingFace es una comunidad y biblioteca líder de software de código abierto que ha ganado una atención significativa en los últimos años por sus contribuciones a la democratización de la

inteligencia artificial. La biblioteca proporciona modelos pre-entrenados, conjuntos de datos y una suite de herramientas que hacen que sea más fácil para los desarrolladores construir y desplegar aplicaciones de inteligencia artificial. Una de las contribuciones más significativas de HuggingFace es el desarrollo de la biblioteca Transformers, que facilita el trabajo con modelos basados en Transformer como BERT y GPT, entre muchos otros

PyTorch Lightning es una biblioteca de Python de código abierto que proporciona una interfaz de alto nivel para PyTorch. Este framework liviano y de alto rendimiento organiza el código de PyTorch para desacoplar la investigación de la ingeniería, haciendo que los experimentos de Deep Learning sean más fáciles de leer y reproducir.

3.4. Comparativa de resultados

Por último, comparamos los mejores modelos de CNN con el modelo de transformer.

Models	**Acc Train**	**Acc Val**	**Tiempo**
InceptionResnet_10	0.971	0.926	54.34m
InceptionResnet_18	0.945	0.917	60.75m
Vision Transformer	0.95	0.92	76m

El primer modelo de `InceptionResnet_10` es el que obtiene un mejor rendimiento en términos de precisión de entrenamiento y validación, y además su tiempo de entrenamiento es más corto (si bien esta diferencia puede deberse a que su entrenamiento ha sido en distintos dispositivos). No obstante, el modelo Vision Transformer también muestra un rendimiento excelente, y un menor sobreajuste al conjunto de entrenamiento. Por lo tanto, el mejor modelo en un entorno real sería el modelo de Vision Transformer.

4. Generación sintética de imágenes

En este último apartado de la práctica, exploraremos cómo generar imágenes sintéticas utilizando una red generativa antagónica profunda, Deep Convolutional Generative Adversarial Network (DCGAN). Se han realizado pruebas sobre los conjuntos de datos de MNIST y Fashion MNIST, dos conjuntos de datos muy conocidos en el campo del aprendizaje profundo, utilizados comúnmente para la tarea de clasificación de imágenes. El conjunto de datos MNIST consiste en 70,000 imágenes en escala de grises de 28x28 píxeles que representan dígitos escritos a mano del 0 al 9. Por otro lado, Fashion MNIST es un conjunto de datos similar a MNIST, pero en lugar de dígitos escritos a mano, consiste en 70,000 imágenes en escala de grises de 28x28 píxeles que representan diez categorías de ropa diferentes, como camisetas, pantalones, abrigos y zapatos, entre otros. Este conjunto de datos se utiliza comúnmente como una alternativa a MNIST, ya que es más desafiante y realista dado que sus imágenes son de mayor complejidad.

4.1. Introducción

Las redes generativas adversarias (GANs) son un tipo de red neuronal que fueron propuestas por primera vez por Ian Goodfellow en 2014, aunque no fue hasta hace unos pocos años que su utilización se empezó a popularizar, dados los problemas que este tipo de modelos mostraban durante el entrenamiento de los mismos. Estas tienen numerosas aplicaciones, entre las que destaca la obtención de datos sintéticos (*data augmentation*) para incrementar el tamaño de los conjuntos de entrenamiento.

Este tipo de redes están compuestas a su vez por dos redes neuronales:

- **Generador**: Esta red neuronal recibe como input un vector de números aleatorios de un tamaño concreto (típicamente sigue una distribución gaussiana), y genera como resultado una imagen.
- **Discriminador**: El discriminador es una red neuronal convolucional al uso, que en este caso trata de clasificar imágenes en dos categorías (es decir, lleva a cabo una clasificación binaria) entre imágenes reales e imágenes falsas o *fake* (aquellas generadas por el generador).

La arquitectura de las DCGANS presenta una estructura similar a la que se muestra en la siguiente imagen:

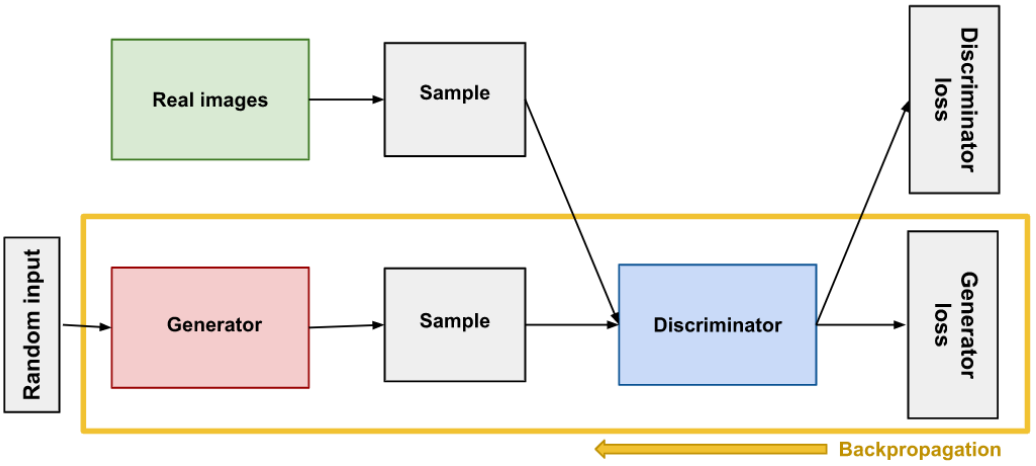

Figura 8: Arquitectura de la DCGAN

Como se puede apreciar en la imagen anterior, se tienen dos conjuntos de imágenes con las cuales se entrena el discriminador. Por un lado, un conjunto de imágenes reales, y por el otro, imágenes generadas por el generador de forma iterativa según este va aprendiendo a generar imágenes cada vez más realistas.

Durante el proceso de entrenamiento de una DCGAN, el generador intenta generar imágenes lo más parecidas posibles a la realidad con el objetivo de engañar al discriminador, mientras que este último intenta ser capaz de clasificar correctamente las imágenes. El proceso de entrenamiento de una DCGAN se puede resumir en los siguientes pasos:

1. En primer lugar se selecciona un subconjunto o mini-batch de imágenes reales de forma aleatoria del dataset de entrenamiento (es decir, del conjunto de imágenes reales disponible).

2. Se genera un subconjunto o mini-batch del mismo tamaño, pero en este caso conformado por imágenes fake generadas por el generador (inicialmente el generador no ha sido entrenado por lo que generará imágenes muy poco realistas).

3. Ambos subconjuntos de datos son proporcionados al discriminador, y este clasifica las imágenes en dos categorías: imágenes reales e imágenes fake.

4. Se calculan los errores de clasificación del discriminador, y mediante el algoritmo de backpropagation se actualizan los parámetros del discriminador, de forma que en cada iteración mejore su capacidad de clasificar correctamente las imágenes reales y las imágenes fake.

5. Respecto al generador, este también actualiza sus parámetros, pero éste solo tiene en cuenta el error de clasificación del discriminador para las imágenes fake (es decir, como de bien o mal lo ha hecho el discriminador con las imágenes que el generador había generado).

Es importante destacar que mientras que el objetivo del discriminador es minimizar su función de pérdida, el objetivo del generador es maximizarla. El modelo coverge cuando el generador produce imágenes perfectas, de forma que el discriminador no es capaz de distinguir entre ambas (es decir, este tiene una precisión del 50 %). Sin embargo, no es sencillo llegar a la convergencia, ya que es relativamente sencillo que el discriminador se vuelva mucho mejor que el generador. Por ello, es necesario limitar la capacidad del discriminador.

4.2. Resultados del modelo implementado en Tensorflow

En este apartado mostraremos algunas imágenes generadas por nuestro modelo para ambos conjuntos de datos, y analizaremos el realismo de estas así como si hay una diferencia en rendimiento para uno y otro conjunto de datos. Comenzaremos mostrando los resultados para el conjunto de datos de MNIST:

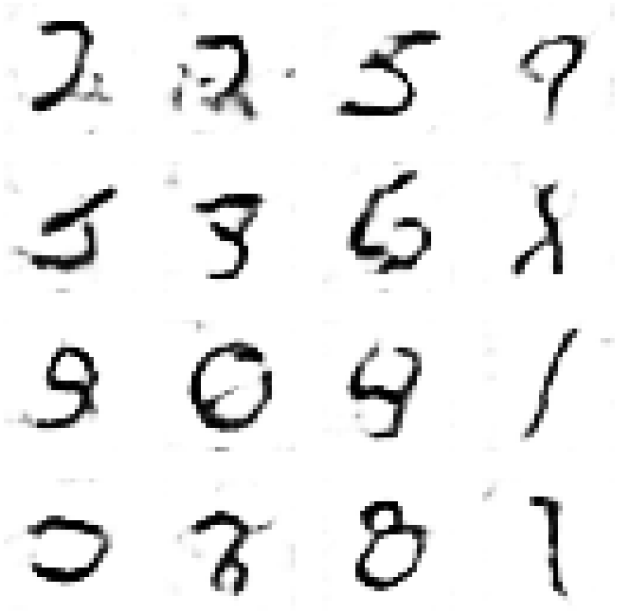

Figura 9: Resultados del modelo implementado en Tensorflow para MNIST

Podemos comprobar que algunas imágenes generadas por el generador son relativamente realistas, si bien otras no se parecen a ningún número real. También se puede observar que hay algo de ruido en los dígitos, especialmente en torno a los bordes.

Figura 10: Resultados del modelo implementado en Tensorflow para Fashion-MNIST

Vemos que en este caso el generador, pese a haber sido entrenado durante el mismo número de epochs, genera peores resultados que en el caso anterior. Esto se debe a que el conjunto de datos de Fashion MNIST es más complejo que el de MNIST, y por tanto el generador no ha sido capaz de aprender a generar imágenes tan realistas como en el caso anterior.

4.3. Resultados del modelo implementado en PyTorch

Se ha desarrollado un código equivalente al mostrado en clase con la librería TensorFlow, pero en este caso haciendo uso de PyTorch. De esta forma, se pueden comparar las diferencias en cuanto a la implementación de los modelos en ambas librerías.

Si bien los métodos utilizados en la clase son los mismos que en la clase de TensorFlow proporcionada, ha sido realizar numerosos cambios para que el código pudiera ser ejecutado en PyTorch. Se puede observar como PyTorch permite una implementación a más bajo nivel, ofreciendo una mayor capacidad de control al usuario sobre las implementaciones de los modelos. Sin embargo, esto puede hacer que la curva de aprendizaje sea más elevada que en el caso de TensorFlow.

Adicionalmente, se ha modificado levemente la arquitectura de ambos modelos (generador y discriminador), para así comprobar si se observa alguna diferencia significativa con respecto al modelo implementado en TensorFlow. En Pytorch se emplea ReLU como función de activación para el generador, y se agrega una capa convolucional de 32 filtros al discriminador que no se incluye en el código en TensorFlow, de forma que se aumenta la complejidad de este.

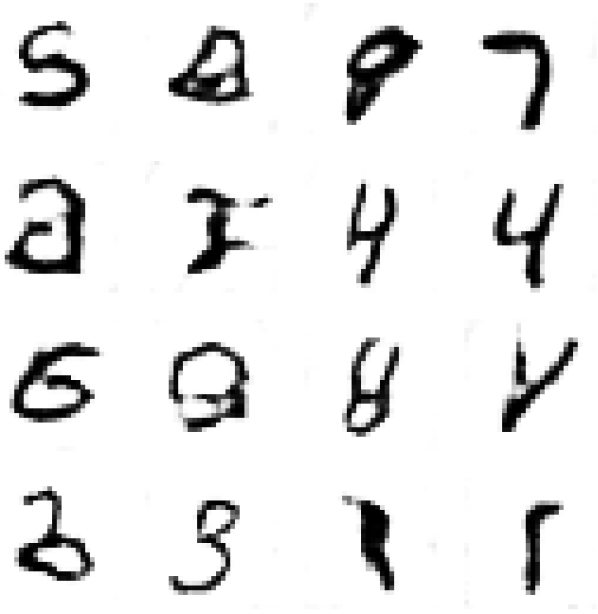

Figura 11: Resultados del modelo implementado en PyTorch para MNIST

Las imágenes generadas son similares a aquellas obtenidas por el modelo implementado en Tensorflow. Podemos observar un menor ruido en torno a los dígitos, y también se presentan más imágenes de aspecto realista.

Figura 12: Resultados del modelo implementado en PyTorch para Fashion-MNIST

De nuevo, las imágenes obtenidas son de una menor calidad que aquellas del conjunto de datos de MNIST. También tenemos un resultado mejor que en el modelo de Tensorflow. Probablemente esto se deba a la nueva capa convolucional, que dota de mayor complejidad a los ejemplos creados.