Universidad Politécnica de Madrid

Máster Universitario en Inteligencia Artificial

Reporte de resultados de Clasificición de Imágenes y Detección de Objetos

Asignatura: Visión por Computador

Autores:
Joel Pardo Ferrera
Daniel Carmona Pedrajas

Correos:
joel.pardof@alumnos.upm.es
daniel.carmonap@alumnos.upm.es

23 Enero 2023

Índice

1. Introducción

En este reporte analizamos los resultados obtenidos en las prácticas de clasificación de imágenes y detección de objetos.
En la Sección 2 trataremos la clasificación de imágenes y en la Sección 3 trataremos la detección de objetos.

2. Reporte de Clasificación de Imágenes

En esta sección abordaremos el problema de clasificación de imágenes con redes neuronales usando el conjunto *benchmark* de imágenes llamado "xview_recognition", en primer lugar usando *Feed Forward Neural Networks* en la sección 2.1 y a continuación con *Convolutional Neural Networks*. Por último, extraeremos conclusiones del proceso en la Sección 2.3.

Para este problema trabajaremos con el conjunto de datos xView, en nuestro caso un subconjunto de las siguientes clases: Las clases están desbalanceadas lo que afectará a nuestros modelos.

Categoría	**Count**	**Frecuencia**
Cargo plane	628	0.12 %
Helicopter	49	0.01 %
Small Car	195133	38.76 %
Bus	6549	1.30 %
Truck	10640	2.11 %
Motorboat	1231	0.24 %
Fishing vessel	736	0.15 %
Dump truck	1238	0.25 %
Excavator	706	0.14 %
Building	283491	56.32 %
Storage tank	1462	0.29 %
Shipping container	1522	0.30 %

Tabla 1: Distribución de categorías

Las imágenes son de tamaño 224×224×3 lo que determinará el tamaño de entrada de las redes que entrenemos. Para el problema de clasificación, entrenaremos nuestras redes de neuronas con 5000 imágenes de cada clase como máximo y utilizaremos todo el conjunto de test. Hemos entrenado las redes de neuronas localmente, lo que ha limitado el número de experimentos que hemos podido realizar.

Data Augmentation consiste en generar nuevos datos a partir de los datos existentes, con el objetivo de aumentar la cantidad de datos para el entrenamiento de un modelo. Esto ayuda a mejorar el rendimiento del modelo al reducir el sobreajuste y aumentar la capacidad del modelo para generalizar a nuevos datos. Las técnicas empleadas son:

- Rotación: Rotar las imágenes a la izquierda, derecha y 180 grados.
- Reflejo: Volteando las imágenes horizontal o verticalmente.
- Cambio de iluminación: Modificando la iluminación de las imágenes.
- Desplazamiento de canal: Cambiar el valor de los canales de color de una imagen.

2.1. Reporte FFNNs

En primer lugar, buscaremos una arquitectura *Feed Forward* con la que obtengamos un buen ratio de clasificación. En la tabla 3 se puede ver una tabla con los 17 experimentos realizados, en negrita están resaltados los cambios con respecto al experimento anterior, hemos intentado hacer un sólo cambio por experimento para poder analizar mejor su resultado. En la tabla utilizamos las siguientes abreviaturas por razones de espacio:

- **Nº**: Número del experimento.

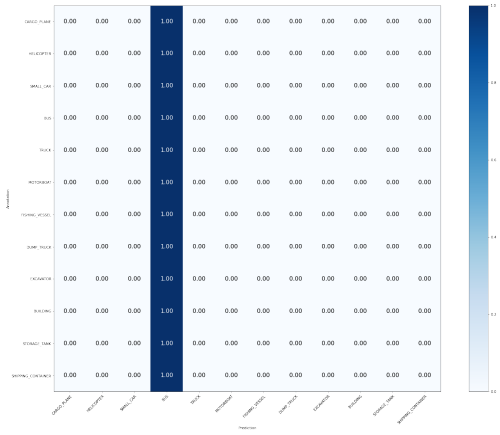
- **M**: Modelo o arquitectura (hay 4 modelos distintos que especificaremos a continuación).
- **LR**: Learning Rate usado en el optimizador.
- **Opt**: Optimizador utilizado.
- **Act**: Función de activación de las neuronas.
- **BS**: Batch Size utilizado.
- **Cbs**: Callbacks donde reduce learning rate se abrevia como *rlr* cada x episodios y con ratio de x y *es* es early stopping para la métrica de *val_accuracy* en x episodios.
- **Reg**: Regularización de los pesos.
- **Init**: Inicializador de los pesos de las neuronas.
- **Dr**: Ratio de dropout.
- **BN**: Indica si hemos usado batch normalization o no.
- **Acc**: Mean Accuracy del modelo en el conjunto de test.
- **Rec**: Mean Recall del modelo en el conjunto de test.
- **Prec**: Mean Precision del modelo en el conjunto de test.
- **Time**: Tiempo de entrenamiento en minutos.

A continuación enumeramos los modelos usados:

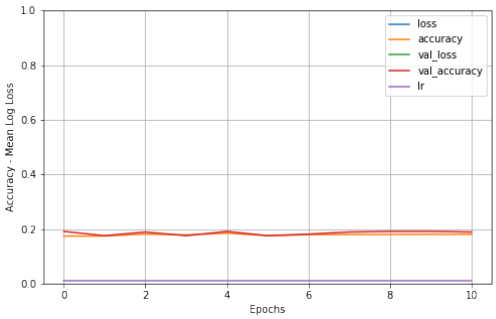
1. **Modelo base**: Sólo una capa de entrada y otra de salida.
2. **Modelo con cuatro capas profundas**: 256, 128, 64, 32 neuronas respectivamente además de las capas de entrada y salida
3. **Modelo con tres capas profundas**: 512, 128, 32 neuronas respectivamente, además de las capas de entrada y salida.
4. **Modelo con dos capas profundas**: 64, 32 neuronas respectivamente, además de las capas de entrada y salida.

Pasamos a explicar los cambios que hemos ido realizando en los experimentos y en qué afectan:

- En los experimentos 1 y 2 utilizamos los dos primeros modelos, para hacer unas pruebas iniciales y vemos que el modelo 2 consigue mejores resultados en cuanto a accuracy y recall. Hemos usado dropout y para comprobar si ha afectado al rendimiento de nuestros modelos procedemos a eliminarlo de nuestros modelos.
- Del experimento 3 al 8, comprobamos que no utilizar ningún tipo de regularización, ya sea dropout u otros métodos hacen que la red no aprenda aunque se cambie el learning rate, esto es porque tenemos un gran número de parámetros y lo que ocurre es que el modelo tiende a quedarse en un mínimo local, clasificando todas las imágenes en la misma categoría como podemos ver en Fig.1.
- En el experimento 9, introducimos Batch Normalization y vemos que el impacto es enorme, la red comienza a aprender y no clasifica todas las imágenes en la misma categoría como se puede ver en Fig.2
- A partir del experimento 10 cambiamos los callbacks, aumentamos la paciencia del early stopping y la reducimos para el decrecimiento del learning rate y comprobamos que obtenemos un entrenamiento que permite un aprendizaje más progresivo. Vamos incorporando inicializadores, para los que no encontramos diferencias significativas, y regularizadores de pesos hasta el episodio 13, en el que obtenemos un accuracy de 60 % en el conjunto de test pero es engañoso ya que en entrenemiento no ha pasado del 30 % de accuracy. Además, la matriz de confusión nos indica que el modelo se ha quedado en un mínimo local como se puede ver en la figura 3.



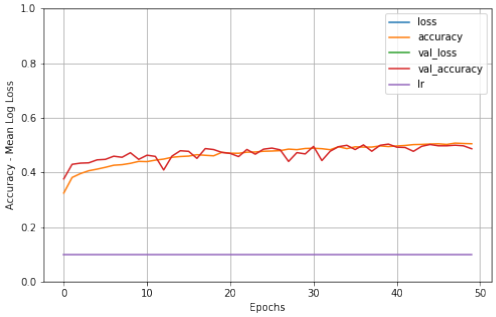
(a) Matriz de confusión del experimento 7



(b) Entrenamiento del experimento 7

Figura 1: Resultados del experimento 7

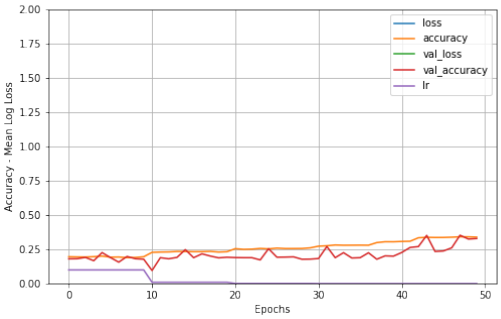
(a) Matriz de confusión del experimento 9



(b) Entrenamiento del experimento 9

Figura 2: Resultados del experimento 9

(a) Matriz de confusión del experimento 13



(b) Entrenamiento del experimento 13

Figura 3: Resultados del experimento 13

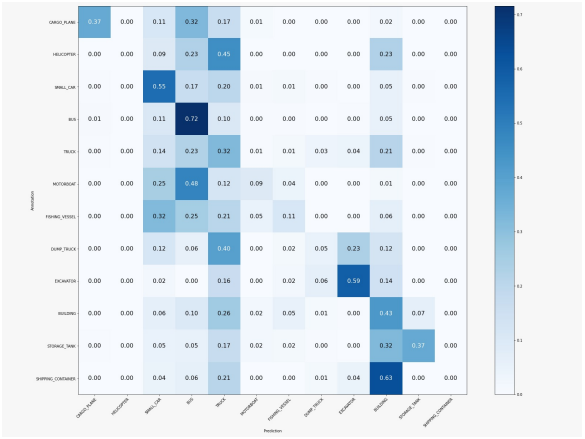
- Cambiando la regularización solucionamos el problema anterior y a partir del episodio 14 probamos la mejor configuración encontrada con todos los modelos para determinar finalmente que el modelo 3 es el

	Accuracy	Recall	Precision
Sin data augmentation	45.89	29.86	19.61
Con data augmentation	27.34	40.19	16.95

Tabla 2: Comparación de métricas para el modelo 3 con data augmentation y sin data augmentation

mejor de ellos.

Por último, realizamos data augmentation para el modelo 3 y comprobar si tenemos un mejor resultado. La forma de aumentar nuestros datos ha sido añadir imágenes a nuestra base de datos para las categorías con menos de 5000 imágenes haciendo un flip de la imagen original y rotando ambas versiones 90, 180 y 270 grados obteniendo así 7 imágenes por categoría. El efecto ha sido que el accuracy se ha reducido drásticamente a cambio de un aumento en el recall como vemos en la tabla 2 obteniendo así una matriz de confusión más centrada como vemos en Fig.4



(a) Matriz de confusión del modelo 3 sin data augmentation

(b) Matriz de confusión del modelo 3 con data augmentation

Figura 4: Comparativa de matrices de confusión del mejor modelo FFNN

Nº	M	Eps	LR	Opt	Act	BS	Cbs	Reg	Init	Dr	BN	Acc	Rec	Prec	Time
1	1	20	0.001	Adam 0.9 0.999	relu	32	rlr 10 0.1 - es 40	No	No	0.2	No	12.65	15.34	13.96	1197
2	**2**	20	0.001	Adam 0.9 0.999	relu	32	rlr 10 0.1 - es 40	No	No	0.2	No	33.54	16.75	6.27	1206
3	2	20	**0.1**	Adam 0.9 0.999	relu	32	rlr 10 0.1 - es 40	No	No	**No**	No	0.22	8.33	0.01	1286
4	**1**	50	**0.001**	Adam 0.9 0.999	relu	32	rlr 3 0.1 - es 5	No	No	No	No	1.95	13.44	11.74	724
5	**2**	50	**0.01**	Adam 0.9 0.999	relu	**64**	rlr 10 0.1 - es 10	No	No	No	No	0.22	8.33	0.01	1358
6	**3**	50	0.01	Adam 0.9 0.999	relu	64	rlr 10 0.1 - es 10	No	No	No	No	0.80	8.33	0.06	3209
7	**4**	50	0.01	Adam 0.9 0.999	relu	64	rlr 10 0.1 - es 10	No	No	No	No	0.80	8.33	0.06	2171
8	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 10 0.1 - es 10	No	No	No	No	0.80	8.33	0.06	2742
9	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 10 0.1 - es 20	No	No	No	**Yes**	36.08	25.88	20.46	9634
10	4	50	0.1	Adam 0.9 0.999	relu	64	**rlr 5 0.1 - es 20**	No	**HeUniform**	0.2	Yes	41.99	28.93	18.82	9670
11	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	No	**HeNormal**	0.2	Yes	40.64	29.65	18.97	9505
12	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	**L1 0.01**	**HeUniform**	0.2	Yes	49.34	16.14	15.03	9457
13	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	**L1 0.001**	HeUniform	0.2	Yes	60.14	14.56	11.10	9915
14	4	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	**L2 0.00001**	HeUniform	0.2	Yes	40.55	22.47	26.05	9957
15	**2**	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	L2 0.00001	HeUniform	0.2	Yes	43.56	24.93	16.00	10247
16	**3**	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	L2 0.00001	HeUniform	0.2	Yes	0.80	8.33	0.06	4515
17	3	50	0.1	Adam 0.9 0.999	relu	64	rlr 5 0.1 - es 20	L2 0.00001	HeUniform	0.2	Yes	45.89	29.86	19.61	10025

Tabla 3: Experimentos realizados para la clasificación de imágenes con FFNNs

2.2. Reporte CNNs

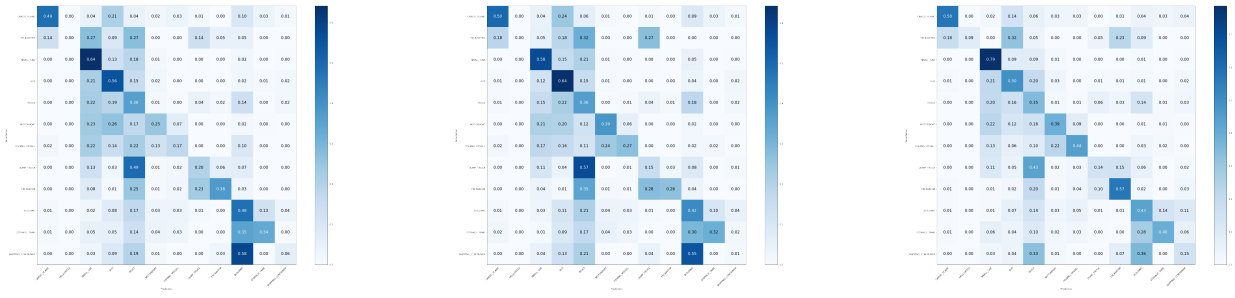
En esta sección buscaremos la mejor arquitectura con capas convolucionales. Variaremos las capas iniciales de nuestras redes de neuronas jugando con distintos números de filtros y concatenaciones de convoluciones, la parte final de nuestra red será un clasificador con la misma estructura para todas nuestras arquitecturas convolucionales: el modelo 4 obtenido en el experimento 14, aunque también reentrenaremos sus pesos. Para esta batería de experimentos hemos usado las siguientes arquitecturas:

1. **Modelo de una capa convolucional pequeño**: Una capa convolucional de 64 filtros y una de max pooling, seguida por la arquitectura del clasificador.
2. **Modelo de una capa convolucional grande**: Una capa convolucional de 128 filtros y una de max pooling, seguida por la arquitectura del clasificador.
3. **Modelo de dos capas con número de filtros decreciente**: Dos capas convolucionales de 128 y 64 filtros con capas de max pooling después de cada una de ellas, seguida por la arquitectura del clasificador.
4. **Modelo de dos capas con número de filtros creciente**: Dos capas convolucionales de 64 y 128 filtros con capas de max pooling después de cada una de ellas, seguida por la arquitectura del clasificador.
5. **Modelo de tres capas con número de filtros creciente**: Dos capas convolucionales de 64, 128 y 256 filtros con capas de max pooling después de cada una de ellas, seguida por la arquitectura del clasificador.
6. **Modelo de tres capas convolucionales con *stackeo* con número de filtros creciente**: Tres capas convolucionales de 128-128, 256-256 y 512-512 filtros (estilo VGG) con capas de max pooling después de cada una de ellas, seguida por la arquitectura del clasificador.

Los resultados de los experimentos se pueden consultar en la tabla 5 (estos cambios solo afectan a la parte convolucional de la arquitectura, no al clasificador densamente conectado) y su análisis es el siguiente:

- En los experimentos 18 y 19 podemos observar que simplemente con usar una capa convolucional los resultados mejoran mucho con respecto a una red neuronal tradicional sin usar ningún tipo de regularización, dropout o batch normalization, también vemos que con una capa convolucional con más filtros, mejor accuracy obtenemos, esto es porque las convoluciones son capaces de mantener la información espacial de las imágenes, permitiendo así a la capa de pooling presentar una features más ricas en información al clasificador densamente conectado.
- Del experimento 20 al 25 vamos introduciendo paulatinamente inicializadores, regularización y batch normalization viendo como mejoran las métricas de accuracy y recall aunque no hay cambios sustanciales en la precisión. También podemos remarcar que cuando todos los elementos actúan en conjunto, el modelo 2 ha salido perjudicado con respecto al 1.
- Es interesante el experimento 26, donde añadimos dropout a las capas convolucionales, esto tiene un efecto desastroso, se reduce el accuracy un 12 % con respecto al mejor resultado encontrado hasta el momento con arquitecturas convolucionales, creemos que es porque el dropout no permite a la convolución detectar bien en qué partes de la imagen hay ciertas estructuras o elementos y por lo tanto el pooling tampoco es preciso. Es por esto que prescindiremos del dropout en la parte convolucional de nuestras arquitecturas a partir de ahora.
- Los experimentos 27 y 28 comparan el uso de dos capas convolucionales que crecen y decrecen en cuanto a filtros, en este caso, el modelo 3 ha obtenido mejores resultados, no obstante nos quedaremos con el modelo que va aumentando los filtros a medida que avanzamos en profundidad en la red porque es lo que se ha visto que funciona en arquitecturas como VGG.
- En el experimento 29 entrenamos el modelo 5 y vemos que el modelo 3 todavía sigue siendo el que mejor resultados ha obtenido. En el experimento 30 introducimos como función de activación la función elu, que impide que las neuronas mueran (su peso sea menor que 0) y vemos que tiene un impacto fuerte en el *recall*, ha mejorado un 4 % con respecto al mejor recall encontrado hasta el momento.
- Por último en el experimento 31 hemos usado una arquitectura VGG reducida y se puede comprobar que es el modelo con mejores métricas de todos los experimentos.

En general, hemos observado que el recall es mucho mejor en las redes convolucionales como se puede comprobar en Fig. 5 donde las matrices de confusión se encuentran centradas, cosa que solo ha ocurrido en las FFNNs cuando hemos usado data augmentation. Además la precisión obtenida también ha sido en general mejor que con las FFNNs.



(a) Matriz de confusión del experimento 18

(b) Matriz de confusión del experimento 28

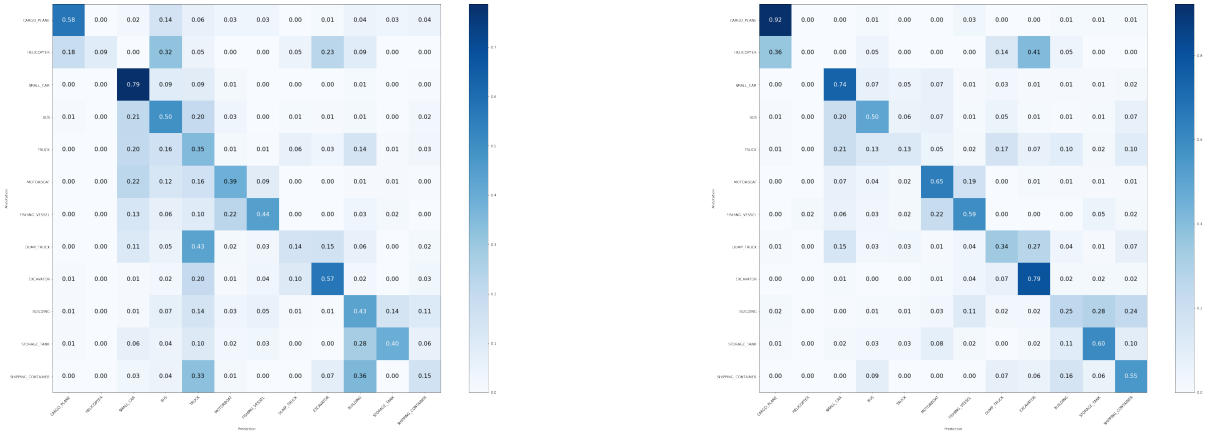
(c) Matriz de confusión del experimento 31

Figura 5: Matrices de confusión de experimentos con redes convolucionales

	Accuracy	Recall	Precision
Sin data augmentation	54.20	40.22	29.62
Con data augmentation	40.68	50.45	20.78

Tabla 4: Comparación de métricas para el modelo 6 con data augmentation y sin data augmentation

De nuevo, entrenamos nuestro mejor modelo (el modelo 6) desde cero con la base de datos de imágenes aumentada y obtenemos el mismo efecto que con las FFNNs, el accuracy se reduce drásticamente pero el recall aumenta un 10 %. En la tabla 4 se muestran los resultados obtenidos y en Fig. 6 se aprecia que la matriz de confusión está ás centrada.



(a) Matriz de confusión del modelo 6 sin data augmentation

(b) Matriz de confusión del modelo 6 con data augmentation

Figura 6: Comparativa de matrices de confusión del mejor modelo CNN

Nº	**M**	**Eps**	**LR**	**Opt**	**Act**	**BS**	**Cbs**	**Reg**	**Init**	**Dr**	**BN**	**Acc**	**Rec**	**Prec**	**Time**
18	1	50	0.1	Adam 0.9 0.999	relu	32	rlr 5 0.1 - se 20	No	No	No	No	35.08	28.05	18.22	20303
19	**2**	50	0.1	Adam 0.9 0.999	relu	**16**	rlr 5 0.1 - se 20	No	No	No	No	40.90	22.14	18.25	20103
20	**1**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	No	**HeNormal**	No	No	37.52	25.64	20.92	19657
22	1	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	**L2 0.00001**	HeNormal	No	No	38.16	30.24	18.21	18377
24	1	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	**Yes**	48.20	27.34	18.03	18326
25	**2**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	40.42	28.82	16.85	18724
26	**1**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	**0.2**	Yes	34.07	20.34	25.94	18458
27	**3**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	52.67	32.75	20.49	19356
28	**4**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	46.48	32.56	19.44	19556
29	**5**	50	0.1	Adam 0.9 0.999	relu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	50.79	35.58	20.79	19881
30	5	50	0.1	Adam 0.9 0.999	**elu**	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	50.12	39.02	21.61	18519
31	**6**	28	0.1	Adam 0.9 0.999	elu	16	rlr 5 0.1 - se 20	L2 0.00001	HeNormal	No	Yes	54.20	40.22	29.62	-

Tabla 5: Experimentos realizados para la clasificación de imágenes con CNNs

2.3. Conclusiones Clasificación de Imágenes

Podemos concluir que las arquitecturas con mayor profundidad tienen mejores resultados, lo mismo ocurre con el número de filtros. Además el data augmentation que hemos utilizado en la tarea de clasificación ha aumentado significativamente el recall, lo que nos indica que la red neuronal es capaz de clasificar una cantidad mayor de imágenes en la categoría correcta, que es algo que nos interesa con una tarea como la que estamos abordando.

3. Reporte de Detección de objetos

La detección de objetos en imágenes y videos mediante visión por computador es una tarea fundamental en el campo de la inteligencia artificial y la robótica. A través de técnicas de procesamiento de imágenes y aprendizaje automático, es posible desarrollar sistemas capaces de detectar y reconocer objetos en una escena.

En esta tarea, llevamos la clasificación de imágenes al siguiente nivel, reconociendo múltiples objetos de diferentes clases dentro de una misma imagen. En este caso el cojunto de imágenes es 'xview_detection'.

3.1. Convolutional Sliding Window

La ventana deslizante convolucional (Convolutional Sliding Window) es una técnica de detección de objetos que utiliza una red convolucional (CNN) para clasificar varias regiones, o ventanas, de una imagen. La CNN se entrena para reconocer el objeto de interés y se aplica a cada ventana a medida que se desplaza por la imagen siguiendo un patrón cuadriculado.

Este enfoque se utiliza normalmente en combinación con un detector de ventana deslizante que se utiliza para generar varias ventanas sobre la imagen donde puede estar situado el objeto de interés.

Una de las ventajas de esta aproximación es que puede manejar variaciones de escala del objeto de interés en la imagen, y no se ve afectado por la ubicación del objeto de interés en la imagen.

3.1.1. Implementación:

Hemos utilizado uno de los modelos implementados en la primera práctica para clasificar cada una de las ventanas en las que se separa una imágen.

3.1.2. Resultado:

El modelo debe de clasificar por cada uno de las cuadrículas extraidas de la imagen. Sin embargo, los clasificadores implementados en la anterior parte tienen un tamaño de entrada especifico de 224 x 224 x 3. Las ventanas varian ese tamaño, por lo tanto, no es capaz de clasificar. La ventana deslizante, en sí misma, funciona.

3.2. YOLO

YOLO (You Only Look Once) es una arquitectura de detección de objetos en imágenes y videos. Es conocida por su alta velocidad y precisión en comparación con otros algoritmos de detección de objetos, por ser fácil de implementar y escalar.

La arquitectura de YOLO es una CNN (Convolutional Neural Network) entrenada para realizar la tarea de detección de objetos en una sola pasada, 'Single-stage', a través de la imagen o el video. Divide la imagen en una cuadrícula y cada celda de esta cuadrícula se encarga de predecir si hay un objeto y en que posición está en la imagen.

YOLO tiene varias versiones. En nuestro caso empleamos la v7. Cada versión mejora en rendimiento y precisión. Algunas de las mejoras incluyen el uso de capas residuales, el uso de capas de atención para mejorar la precisión de la detección y el uso de una arquitectura más profunda para mejorar el rendimiento.

3.2.1. Preparación de datos:

Para utilizar YOLO v7, es necesario seguir un formato específico en el conjunto de datos. Este formato incluye la separación de los datos en tres carpetas distintas: entrenamiento, validación y pruebas. Es importante tener en cuenta que una parte pequeña de los datos de entrenamiento deben ser utilizados para la creación del conjunto de validación. El formato de anotaciones requerido por YOLO v7 es diferente al proporcionado por defecto.

Los datos por defecto según orden de aparición son *Nombre del imagen; Número de objetos en la imagen; Número; Caja donde se localiza un objeto (Minimo en el eje x, Minimo en el eje y, Maximo en el eje x, Maximo en el eje y); Clase*. Un ejemplo de los datos por defecto es:

$$train/2538.tif; 3; 386913; 652, 2, 673, 12; 41; 682970; 747, 15, 771, 40; 73; 682971; 618, 33, 630, 49; 41$$

El formato especifico que necesita YOLO consiste en un '.txt' por cada una de las imagenes que contenga una cadena separada por espacios donde cada una de las lineas representa a un objeto en dicha imagen. Las elementos contenidos en cada cadena son:

- Primer elemento es la clase del objeto a tratar, normalizada entre 0 y 1.
- Segundo elemento es el centro x de la caja del objeto, respecto del total de la imagen. Este valor viene dado por:

$$Centro_x = \frac{(max(x) - min(x))/2}{anchura}$$

- Tercero elemento es el centro y de la caja del objeto, respecto del total de la imagen. Este valor viene dado por:

$$Centro_y = \frac{(max(y) - min(y))/2}{altura}$$

- Cuarto elemento es el ancho máximo de la caja del objeto, respecto del total de la imagen. Este valor viene dado por:

$$Anchura_{caja} = \frac{(max(x) - min(x))}{anchura}$$

- Quinto elemento es la altura máxima de la caja del objeto, respecto del total de la imagen. Este valor viene dado por:

$$Altura_{caja} = \frac{(max(y) - min(y))}{altura}$$

3.2.2. Implementación:

El primer experimento consiste en una red formada por varias capas convolucionales con filtros desde 32 hasta 512, todos mutilplos de 2, capas de concatenación, y max pooling. Primero se definen los parámetros de la red, es decir, el número de clases, la profundidad del modelo y el ancho de las capas. Posteriormente, se entrena y evalua.

3.2.3. Resultado:

Después de adecuar el entorno, los datos y la arquitectura del modelo, debido a la falta de potencia en la unidad de procesamiento gráfica, al entrenar con una RTX3070 con 16GB, no llega a pasar de media época de entrenamiento. Se han probado otras gráficas anteriormente pero tenian caracteristicas por debajo con respecto a la mencionada. Todo el código está disponible en YOLOv7.

4. Conclusiones

En la realización de esta memoria se han planteado dos tareas; clasificación y reconocimiento de objetos.

Para la primera hemos implementado casi 50 experimentos incluyendo FFNNs y CNNs, de los cuales en este documento solo aparecen los más relevantes.

Para la tarea de reconocimiento de objetos hemos planteado dos aproximaciones; el uso de una ventana convolucional deslizante y la arquitectura YOLO. Debido a los requisitos de computación necesarios, al desbordamiento del CeSViMa por la compartición de memoria y gráficas, así como las barreras que supone hacerlo en una plataforma como Google Colab (donde se ha intentado varias veces), ya sea por la cantidad de datos como por la limitación temporal que te impone a la hora de entrenar, los experimentos con estas técnicas no han conseguido ser satisfactorios.

De este modo, podemos concluir que hemos cumplido con el cometido de la primera tarea, mientras que en la segunda los resultados no han sido los esperados.