

**MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS**



**VNIVERSITAT  
DE VALÈNCIA**

**TRABAJO DE FIN DE MÁSTER**

**TRANSFORMERS EN SEGMENTACIÓN DE  
IMÁGENES: RENDIMIENTO EN EL MUNDO REAL  
TRAS ENTRENAMIENTO EN IMÁGENES SINTÉTICAS**

**AUTOR:**

**LORENZO PARDO CHICO**

**TUTORES**

**VALERO LAPARRA PÉREZ-MUELAS**  
**PABLO HERNÁNDEZ CÁMARA**

**JULIO, 2023**

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Segmentación de imágenes . . . . .	1
1.2. Transformers en Computer Vision . . . . .	3
1.3. Objetivos . . . . .	4
<b>2. Marco teórico</b>	<b>7</b>
2.1. Multilayer Perceptron (MLP) . . . . .	7
2.2. Convolutional Neural Network (CNN) . . . . .	9
2.3. Transformers . . . . .	11
2.3.1. Encoder . . . . .	13
2.3.2. Decoder . . . . .	17
2.3.3. Transición a Computer Vision . . . . .	18
2.4. Métricas . . . . .	19
2.4.1. Intersection over Union . . . . .	19
2.4.2. Precision . . . . .	20
2.4.3. Recall . . . . .	21

## ÍNDICE GENERAL

---

2.4.4. F1-score . . . . .	21
2.5. Funciones de coste . . . . .	22
2.5.1. Cross Entropy . . . . .	22
2.5.2. <i>Mean Intersection over Union</i> como función de coste . . . . .	23
<b>3. Modelo Segformer</b>	<b>25</b>
3.1. Arquitectura . . . . .	25
3.1.1. Encoder . . . . .	26
3.1.2. Decoder . . . . .	30
3.2. Rendimiento . . . . .	31
<b>4. Datos</b>	<b>33</b>
4.1. Cityscapes . . . . .	33
4.2. GTA V . . . . .	37
4.3. Mapillary . . . . .	40
4.4. Desbalanceo de clases . . . . .	41
<b>5. Experimentos</b>	<b>45</b>
5.1. Modelo entrenado al completo en Cityscapes . . . . .	46
5.2. Modelo entrenado en Cityscapes y Fine Tuning del decoder en GTA V	49
5.3. Modelo entrenado en Cityscapes y Fine Tuning completo en GTA V .	52
5.4. Modelo entrenado al completo en GTA V . . . . .	55
5.5. Modelo entrenado al completo en GTA V y Fine Tuning del decoder en Cityscapes . . . . .	58

---

5.6. Modelo entrenado al completo en GTA V y Fine Tuning completo en Cityscapes . . . . .	61
5.7. Comparación de modelos . . . . .	64
<b>6. Conclusiones</b>	<b>69</b>

# Capítulo 1

## Introducción

La Visión por Computadora (*Computer Vision* en inglés) es una disciplina que se encarga de la extracción automática de información útil a partir de imágenes y vídeos. Esta disciplina se ha convertido en un área de investigación clave en los últimos años debido a su amplia variedad de aplicaciones, entre las que se incluyen la detección de objetos, la clasificación y la segmentación de imágenes. Estas aplicaciones se pueden usar en tareas que van desde el reconocimiento de patrones, la biometría, la vigilancia, la robótica, la medicina, la realidad aumentada, hasta la automatización de procesos industriales y de la vida diaria.

En este trabajo nos vamos a centrar en la segmentación de imágenes haciendo uso de una arquitectura de redes neuronales conocida como Transformers, que ha revolucionado el campo del procesamiento del lenguaje natural (NLP, de sus siglas en inglés) y ha demostrado ser altamente efectiva en otras tareas como la Visión por Computadora.

### 1.1. Segmentación de imágenes

La segmentación de imágenes es una tarea fundamental en Visión por Computadora que consiste en dividir una imagen en múltiples partes o regiones con el objetivo de separar los objetos de interés del fondo o de otros objetos en la escena. Se puede considerar como una clasificación de cada uno de los píxeles de una imagen en los diferentes elementos u objetos que la componen. Esta tarea es importante ya que permite la identificación y localización precisa de objetos en imágenes, lo que es útil en una amplia variedad de aplicaciones, como la conducción autónoma, la robótica, la medicina, el monitoreo de cultivos y la seguridad.



**Figura 1.1:** Ejemplo de segmentación semántica. A la izquierda se muestra la imagen y a la derecha la máscara de segmentación donde a cada elemento segmentado se le asigna un color diferente. [23]

Existen diferentes tipos de segmentación de imágenes:

1. **Segmentación semántica.** Esta técnica implica la asignación de etiquetas a cada píxel de la imagen de modo que esta se divide en las diferentes clases que la componen. Por ejemplo, se clasifica cada píxel en categorías como coche, árbol, persona, etc. pero sin diferenciar entre diferentes objetos de la misma clase. En la figura 1.1 podemos ver un ejemplo de este tipo de segmentación.
2. **Segmentación de instancia.** Esta técnica implica la asignación de una etiqueta única a cada instancia de un objeto en la imagen. Por ejemplo, si en una imagen hay cuatro personas, se asignará una etiqueta diferente a cada una de ellas. En este tipo de segmentación no se realiza la segmentación de todos los objetos de la imagen, si no que se centra en segmentar tipos de objeto determinados.
3. **Segmentación panóptica.** Esta técnica combina la segmentación semántica y la de instancia en una sola tarea, lo que permite una comprensión más completa de la escena en la imagen. Se realiza la segmentación de toda la imagen y, además, se diferencia entre los objetos que pertenecen al mismo tipo, asignando etiquetas diferentes.

En este trabajo nos centramos en el primer tipo de segmentación de imágenes: la segmentación semántica.

Existen diferentes enfoques para llevar a cabo la segmentación de imágenes, pero uno de los más populares es el uso de modelos previamente entrenados para segmentar objetos en la imagen. El modelo coge como input la imagen sin segmentar y devuelve como output la clasificación de cada píxel. Las redes neuronales convolucionales (CNNs) han demostrado ser altamente efectivas en esta tarea gracias a su capacidad para aprender representaciones de las características de la imagen, lo que les permite identificar de manera precisa los objetos y los bordes de las regiones. Sin embargo, varios trabajos

han demostrado que la arquitectura de Transformers alcanza, e incluso supera en algún caso, el rendimiento de las CNNs en esta tarea.

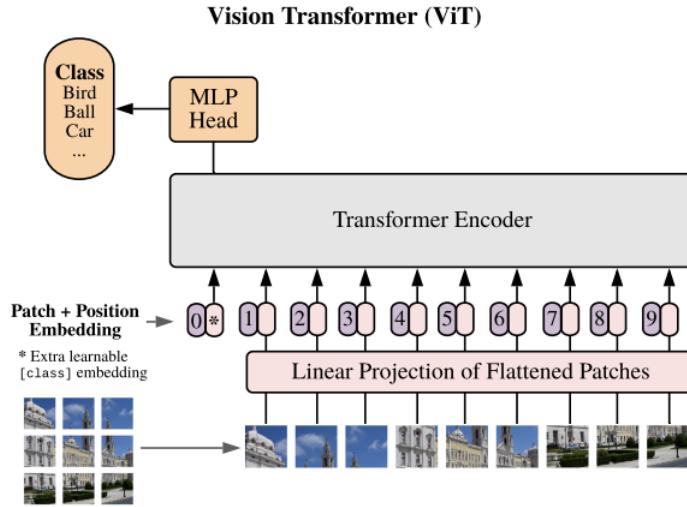
## 1.2. Transformers en Computer Vision

En los últimos años se ha producido un notable avance en la segmentación semántica de imágenes gracias a una variedad de modelos innovadores. Tanto los modelos basados en Transformers como otros enfoques han contribuido significativamente al estado del arte en esta área.

Uno de los modelos con mayor relevancia ha sido el modelo U-Net [12], que consiste en una arquitectura de codificador-decodificador que utiliza conexiones de salto para capturar características a diferentes niveles de escala. Este modelo ha sido ampliamente utilizado y ha sentado las bases para muchos modelos posteriores en este campo. Otro modelo importante es el FCN (*Fully Convolutional Network*) [9] que fue uno de los primeros en aplicar redes neuronales convolucionales completamente convolucionales a la segmentación semántica. En lugar de utilizar capas totalmente conectadas al final de la red, FCN utiliza convoluciones transpuestas para generar una salida de mapa de segmentación con la misma resolución que la imagen de entrada. Otra familia de modelos que ha contribuido de manera significativa es la *DeepLab* [10], que hace uso de una combinación de convoluciones dilatadas y *Atrous Spatial Pyramid Pooling* (ASPP) para capturar información contextual en diferentes escalas, mejorando así la precisión de la segmentación.

Los Transformers son una arquitectura de redes neuronales que fue introducida por primera vez en el paper *Attention Is All You Need* (Vaswani et al.) [14] en el año 2017. Esta arquitectura ha revolucionado el mundo del procesamiento del lenguaje natural y se ha convertido en la base de muchos sistemas de inteligencia artificial. Una de las grandes ventajas de esta arquitectura es que no depende de una arquitectura de red neuronal recurrente, si no que se basa en mecanismos de atención, lo que permite que a la hora de procesar grandes secuencias de texto no se pierda información de las primeras palabras por desvanecimiento del gradiente. Esta arquitectura divide una secuencia de texto en tokens, que es una secuencia de caracteres que se considera como una unidad básica para procesar el texto, y calcula la atención que se prestan unos a otros para la realización de una tarea.

A medida que los transformers se volvieron más populares en NLP, los investigadores comenzaron a explorar su potencial en otras áreas, como la visión por computadora.



**Figura 1.2:** *Vision Transformer (ViT). Este modelo divide la imagen de entrada en diferentes patches y los procesa a través del Transformer como si fuesen tokens de una secuencia de texto.* [6]

La idea principal es que, en lugar de procesar una secuencia de palabras, se procese cada imagen entendiéndola como una secuencia de píxeles o de grupos de píxeles. Esto significa que se puede aplicar el mismo enfoque utilizado en NLP, pero en imágenes.

Uno de los modelos basados en transformers más utilizados en la visión por computadora es el Vision Transformer (ViT) [6], que fue presentado por Google en 2020. Este modelo utiliza el enfoque de atención para procesar imágenes, dividiéndolas en parches y aplicando una capa de atención entre estos parches. Otros modelos que han utilizado transformers en la visión por computadora incluyen el Detection Transformer (DETR) [4] para detección de objetos o el Segformer [24] para segmentación de imágenes.

### 1.3. Objetivos

Uno de los principales inconvenientes a la hora de entrenar un modelo basado en transformers es la gran cantidad de datos necesaria para obtener buenos resultados. Para intentar minimizar este problema, el objetivo de este trabajo es estudiar si el uso de imágenes sintéticas en el entrenamiento de un modelo de segmentación de imágenes mejora los resultados a la hora de aplicar el modelo a imágenes reales.

Nos centraremos en un modelo concreto de segmentación de imágenes basado en transformers, conocido con el nombre de Segformer. Este modelo fue presentado por

Xie et al. en 2021 en el paper titulado *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers* [24]. Realizaremos experimentos en dos direcciones: en primer lugar cargaremos el modelo preentrenado con el conocido dataset Cityscapes y realizaremos Fine Tuning para reentrenarlo para un dataset que consiste en imágenes del videojuego GTA V y, en segundo lugar, aplicaremos el razonamiento inverso entrenando el modelo completo con imágenes de GTA V y realizando Fine Tuning con imágenes reales de Cityscapes. Finalmente, estudiaremos si alguno de estos fine tunings mejora los resultados obtenidos sobre un dataset más complejo de imágenes reales, como es el dataset Mapillary Vistas.

## CAPÍTULO 1. INTRODUCCIÓN

---

# Capítulo 2

## Marco teórico

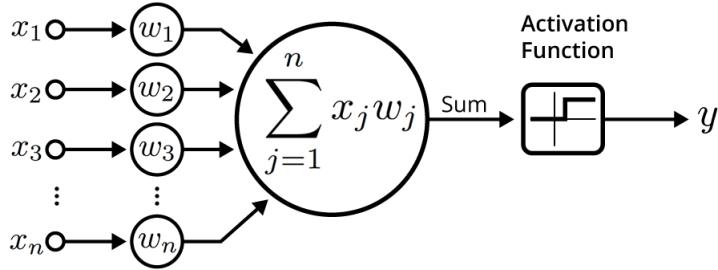
En este capítulo abordamos la teoría de los elementos necesarios para la comprensión del trabajo. La neurona, agrupada en capas formando un Multilayer Perceptron (MLP), es el elemento fundamental de un modelo de Deep Learning. Al tratar con imágenes un elemento clave es la arquitectura de red neuronal convolucional (CNN), ampliamente utilizada en este campo, por lo que un correcto entendimiento de este concepto es vital.

Obviamente, trataremos en profundidad la arquitectura Transformer tal y como surgió inicialmente para NLP en el artículo *Attention Is All You Need* [14]. También comentaremos su transición hacia tareas de Computer Vision.

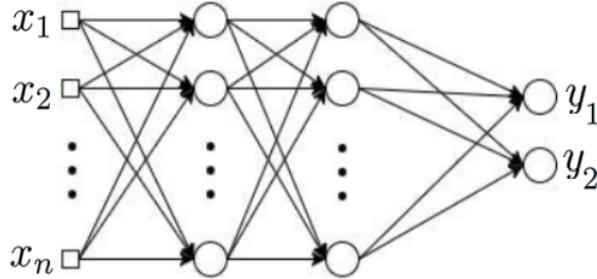
### 2.1. Multilayer Perceptron (MLP)

Un *Multilayer Perceptron* consiste en una serie de capas de neuronas interconectadas donde cada capa está conectada con la siguiente en una arquitectura en cascada. Cada una de las neuronas es un elemento que toma entradas, realiza una suma ponderada de estas entradas y aplica una función de activación no lineal para producir una salida. La arquitectura de una neurona la mostramos en la figura 2.1. En ella, observamos como un vector de entrada  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  es procesado por la neurona para obtener una salida  $\hat{y}$ . Para ello, se aplica una combinación lineal de los valores  $x_i$  ponderados por unos pesos  $w_i$  y se aplica una función de activación que permite una respuesta no lineal. De este modo:

$$\hat{y} = f \left( \sum_{j=1}^n x_j w_j + w_0 \right) \quad (2.1)$$



**Figura 2.1:** Arquitectura de una neurona. [21]



**Figura 2.2:** Multilayer Perceptron. [20]

donde  $f(\cdot)$  es la función de activación y  $w_0$  es un parámetro conocido como *bias*. Los parámetros  $w_i$  se ajustan durante el proceso de entrenamiento mediante un algoritmo conocido bajo el nombre de *backpropagation*.

El *Multilayer Perceptron* (MLP) surge de combinar varias de estas neuronas formando una estructura como la que mostramos en la figura 2.2, en la que cada uno de los nodos se corresponde con una neurona. Observamos que la salida de una neurona se emplea como entrada de las neuronas de la siguiente capa para generar el vector de salida final  $\hat{y}$ . En la figura se muestra un vector de salida de dimensión 2, aunque podría ser de cualquier dimensión. El algoritmo de *backpropagation* consta de los siguientes pasos:

1. Paso hacia adelante (*Forward pass*): consiste en una operación de inferencia en la que el MLP es alimentado con el vector de entrada  $\mathbf{x}$  que se propaga por todas las capas para generar el vector de salida  $\hat{y}$ , conocido como predicción.
2. Cálculo del error: el vector de salida  $\hat{y}$  se compara con el vector objetivo  $\mathbf{y}$  (conocido como *target*) a través de una función conocida como función de coste ( $L$ ). Este error representa cómo de bien está ajustando la red neuronal los datos de entrenamiento a las etiquetas (*target*) y deberá ser minimizado durante el proceso de entrenamiento de la red.
3. Retropropagación del error (*Backpropagation*): se propaga el gradiente de la fun-

ción de coste hacia atrás usando la regla de la cadena. De este modo se calcula la contribución de cada peso  $w_i$  en el error total. Tras esto, se ajustan los pesos en la dirección opuesta al gradiente de la función de coste, de modo que el error disminuye. De este modo, la actualización de pesos se puede expresar como:

$$w_i(t+1) = w_i(t) - \lambda \frac{\partial L}{\partial w_i} \quad (2.2)$$

donde  $t$  hace referencia a la época de entrenamiento y  $\lambda$  es una tasa de aprendizaje que se toma como parámetro de entrenamiento.

Estos tres pasos se repiten hasta la convergencia de la función de coste  $L$ , que indica que el proceso de entrenamiento ha finalizado.

## 2.2. Convolutional Neural Network (CNN)

Las redes neuronales convolucionales son una clase de redes neuronales comúnmente aplicada para tratar imágenes debido a su capacidad de descifrar los patrones más complejos en enormes conjuntos de imágenes. Están basadas en los experimentos que llevaron a cabo David Hubel y Torsten Wiesel en gatos y monos en las décadas de 1950 y 1960. A través de estos experimentos se descubrió que las células de la corteza visual de los animales tenían características selectivas de respuesta a estímulos visuales específicos, como líneas orientadas, bordes y gradientes de colores. Para intentar imitar el comportamiento de las células visuales, las CNNs están compuestas por filtros (conocidos como *kernels*) que realizan operaciones de convolución en las entradas visuales, de modo que cada uno de los filtros se especializa en detectar un tipo de características específico, como pueden ser bordes, texturas o patrones.

La convolución entre dos funciones  $f(t)$  y  $g(t)$ , representada como  $(f * g)(t)$ , indica cómo una función es afectada por otra y se define como:

$$(f * g)(t) \doteq \int_{-\infty}^{\infty} f(\eta)g(t - \eta)d\eta \quad (2.3)$$

En el caso que nos ocupa, al tratar con imágenes, que son elementos discretos, definimos la operación de convolución discreta entre una imagen  $f$  de tamaño  $M \times N$  y un kernel

$w$  de tamaño  $P \times Q$  como:

$$(f * w)[i, j] = \sum_{k=-\frac{P}{2}}^{\frac{P}{2}} \sum_{l=-\frac{Q}{2}}^{\frac{Q}{2}} f[i - k, j - l]w[k, l] \quad (2.4)$$

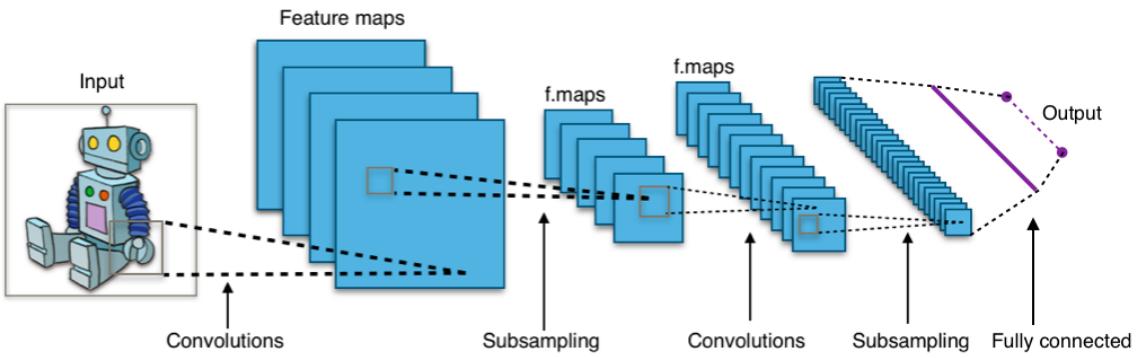
El resultado de esta convolución es otra imagen de tamaño  $M \times N$ . Cabe destacar que asumimos que el centro tanto de la imagen como del *kernel* está situado en su centro y, en el caso en que  $P$  o  $Q$  sean impares, se deben redondear.

Esta operación de convolución se realiza múltiples veces con diferentes kernels, cuyos valores (pesos) se aprenden en el proceso de entrenamiento de la red. El resultado de cada una de las convoluciones es lo que se conoce como mapa de características de modo que con cada *kernel* estamos extrayendo unas características concretas de la imagen.

En la figura [15] podemos observar la arquitectura de la aplicación de 4 capas convolucionales sobre una imagen. En la primera capa tenemos 4 *kernels* diferentes, por lo que, si la imagen input es de tamaño  $M \times N$ , la dimensión del resultado de aplicar esta capa será  $M \times N \times 4$  ya que en este caso el *kernel* recorre todos los píxeles de la imagen. Para extraer una mayor cantidad de mapas de características es conveniente aumentar el número de filtros de la capa convolucional y reducir el tamaño de cada uno de los mapas de características resultantes. Esto lo podemos conseguir si no recorremos todos los píxeles de la imagen a la hora de desplazar el filtro. El número de píxeles que se desplaza el *kernel* a medida que va recorriendo el input se conoce como *stride*. De este modo, si  $stride > 1$  se reducirá el tamaño de los mapas de características. Este fenómeno se ejemplifica en la figura [15] en la segunda y la cuarta capa que componen la CNN. Por ejemplo, si seleccionamos  $stride = 2$ , número de filtros  $k = 5$  y la dimensión del input es  $M \times N$ , la dimensión del output será  $\frac{M}{2} \times \frac{N}{2} \times 5$ .

Otro elemento importante en este tipo de redes es definir la forma en la que se tratan los bordes de la imagen a la hora de desplazar el *kernel*. Una estrategia ampliamente utilizada, y que se emplea en el modelo usado en este trabajo, es añadir ceros a los bordes del input, de modo que la dimensión del output sea la misma que la del input siempre que  $stride = 1$ . Finalmente, para poder aproximar funciones no lineales, se suele añadir una función de activación no lineal tras aplicar el filtro al input. Una de las funciones de activación más famosas y que empleamos en este trabajo es la función de activación conocida como *ReLU* (*Rectified Linear Unit*), que se define como:

$$ReLU(x) = \max(0, x) = \begin{cases} x & \text{si } x > 0, \\ 0 & \text{si } x \leq 0. \end{cases} \quad (2.5)$$



**Figura 2.3:** Arquitectura típica de red convolucional. [15]

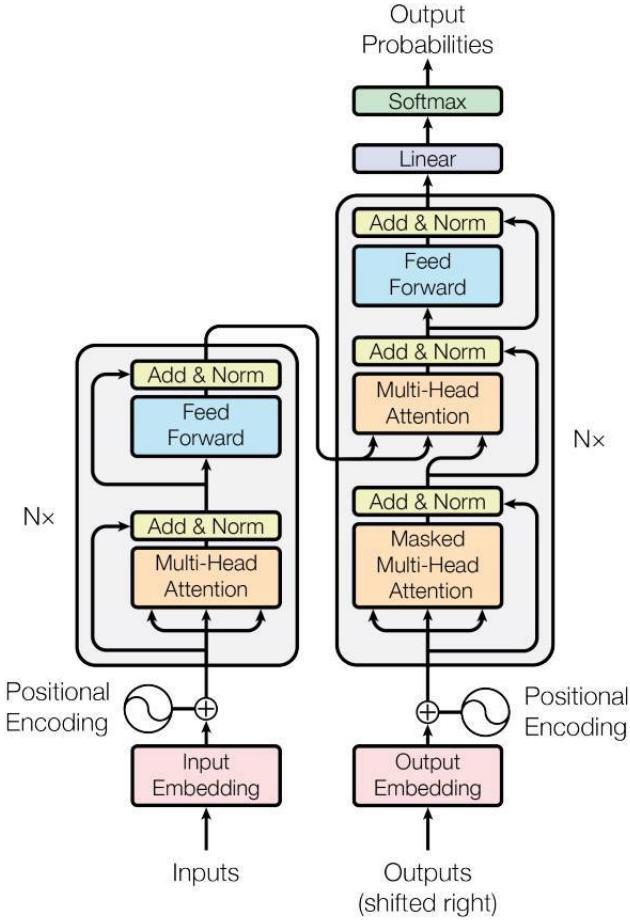
donde  $x$  es el resultado de la operación lineal realizada por el *kernel*.

## 2.3. Transformers

Como ya hemos avanzado en la sección de introducción, los Transformers son un tipo de arquitectura de redes neuronales que surgieron inicialmente para resolver problemas relacionados con el procesamiento del lenguaje natural (NLP) y que posteriormente han sido adaptados para la resolución de tareas muy diversas, alcanzando resultados situados en el estado del arte en gran parte de las tareas en las que han sido empleados. Fueron introducidos por primera vez en el famoso artículo *Attention Is All You Need*, publicado en el año 2017 por investigadores de Google. La arquitectura que proponían se muestra en la figura 2.4. En ella, podemos observar que el Transformer está compuesto por dos partes principales: el encoder (codificador) y el decoder (decodificador). Estas dos partes tienen su propia estructura, aunque muchos de sus elementos son compartidos.

Para el caso de procesamiento del lenguaje, el encoder toma como entrada una secuencia de palabras y la convierte en una serie de vectores de características, que luego se utilizan para realizar tareas como la traducción automática o la generación de texto. El decoder coge estos vectores de características y los utiliza para generar una salida, también en forma de secuencia de palabras.

Antes de entrar en el encoder, podemos ver que existe una capa de embedding y una de positional encoding. Estas capas son fundamentales, pues convierten las entradas de palabras en vectores de números reales que el modelo puede entender y procesar. La capa de embedding utiliza una matriz de pesos entrenables que asigna un vector de características único de dimensión  $d$  a cada palabra en el vocabulario. La dimensión



**Figura 2.4:** Arquitectura Transformer. A la izquierda se sitúa el encoder y a la derecha el decoder. [14]

$d$  del espacio de características es un parámetro del modelo, que aumentará su complejidad a medida que  $d$  aumente. La matriz de pesos se inicializa al azar y se ajusta durante el proceso de entrenamiento para que los vectores pertenecientes a palabras similares se ubiquen cerca en el espacio de características.

La capa de positional encoding agrega información posicional a las representaciones vectoriales de las palabras que pasan a través del modelo. Esto es necesario ya que en una oración el orden de las palabras es fundamental y es una información que la arquitectura Transformer no tiene en cuenta, a diferencia de las redes neuronales recurrentes. Al vector de cada palabra se le suma una codificación posicional que indica la posición que la palabra en cuestión ocupa dentro de la oración. Esta codificación debe cumplir una serie de características [8]:

- Debe ser única para cada posición de una palabra dentro de un texto.
- La distancia entre dos posiciones debe ser consistente entre oraciones con dife-

rentes longitudes.

- Debe poder generalizarse para oraciones de mayor longitud.
- Debe ser determinista.

Una codificación que cumple todas las condiciones mencionadas se muestra en la ecuación (2.6).

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{model}}) \\ PE(pos, 2i + 1) &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.6}$$

donde  $pos$  es la posición que la palabra ocupa en la oración,  $d_{model}$  es la dimensión del espacio de características e  $i$  es la posición del vector del espacio de características. Nótese que a cada posición del vector de características correspondiente a cada palabra le sumamos una cantidad diferente. Si esta posición es par sumamos con la función seno y si es impar sumamos con la función coseno.

### 2.3.1. Encoder

El encoder está compuesto de varias capas ( $N$ , si miramos la figura 2.4) apiladas en serie. Cada una de estas capas está compuesta a su vez de dos subcapas: una capa de atención múltiple y una capa feed-forward (alimentación hacia delante).

La capa de atención múltiple tiene como función principal calcular una representación contextualizada de cada palabra en una secuencia de entrada. Para ello, esta capa utiliza la *atención*, que permite detectar aquellas palabras más importantes de una oración para realizar determinada tarea. Esta técnica permite capturar dependencias de largo alcance en la secuencia de entrada, por lo que el Transformer tiene una mayor memoria que las redes neuronales recurrentes, cuyo principal inconveniente es el desvanecimiento del gradiente que hace que las últimas palabras de una oración pierdan la información de las primeras. Para conseguir este resultado, esta capa sigue los siguientes pasos:

1. Cálculo de los vectores query (**q**), key (**k**) y value (**v**) para cada palabra. Estos vectores se calculan a través de la matrices de pesos  $W^Q$ ,  $W^K$  y  $W^V$  del siguiente

modo:

$$\begin{aligned}\mathbf{q} &= \mathbf{x}W^Q \\ \mathbf{k} &= \mathbf{x}W^K \\ \mathbf{v} &= \mathbf{x}W^V\end{aligned}\tag{2.7}$$

donde  $\mathbf{x}$  es la representación de la palabra en el espacio de características de dimensión  $d$  y, por tanto, es de dimensión  $1 \times d$ . Las matrices  $W$  son matrices de pesos entrentables y se obtienen a partir de capas neuronales densas. La dimensión de estas matrices es la que define la complejidad del modelo en cuanto a número de parámetros. Se definen de forma que  $\dim(W^Q) = \dim(W^K) = d \times d_k$  y  $\dim(W^V) = d \times d_v$ . De este modo,  $\dim(\mathbf{q}) = \dim(\mathbf{k}) = 1 \times d_k$  y  $\dim(\mathbf{v}) = 1 \times d_v$ .

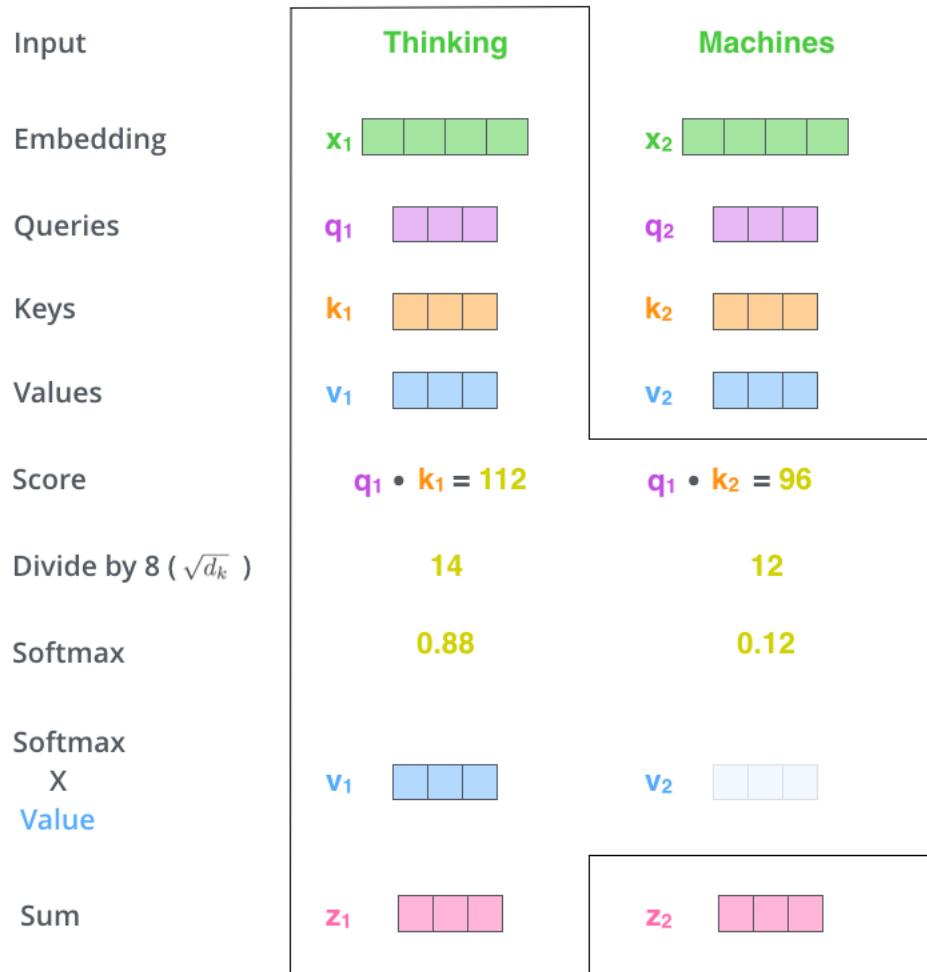
2. Cálculo de la atención que una palabra presta al resto de palabras de entrada. Para esto, se realiza el producto del vector  $\mathbf{q}$  de cada palabra con los vectores  $\mathbf{k}$  de todas las palabras de la entrada, de modo que si la oración de entrada tiene 10 palabras obtengamos 10 valores de atención diferentes para cada una de las palabras, indicando la atención que se prestan entre sí para realizar la tarea objetivo. Con el fin de obtener pesos entre 0 y 1, dividimos entre  $\sqrt{d_k}$  y aplicamos la función de activación *Softmax*.
3. Multiplicación de cada uno de los valores de atención obtenidos por el vector  $\mathbf{v}$  de cada palabra para obtener el vector de atención  $\mathbf{z}$ . Nótese que para calcular el vector  $\mathbf{z}_j$  de cada palabra debemos hacer la suma de los productos del vector  $\mathbf{q}_j$  de la palabra en cuestión con los vectores  $\mathbf{k}_i$  del resto de palabras.

$$\mathbf{z}_j = \sum_i \text{Softmax}\left(\frac{\mathbf{q}_j \mathbf{k}_i}{\sqrt{d_k}}\right) \mathbf{v}_i\tag{2.8}$$

De este modo, en el vector  $\mathbf{z}$  de cada palabra estamos sumando el vector valor  $\mathbf{v}$  del resto de palabras ponderado por la atención prestada, lo que se traduce en introducir el contexto de la oración. Por este motivo, al vector  $\mathbf{z}$  se le conoce como vector de contexto.

En la figura 2.5, extraída de [1] se muestra el proceso que hemos descrito tomando como ejemplo una frase de dos palabras.

Para tratar múltiples palabras a la vez debemos introducir un formalismo matricial. Esto se puede hacer de forma simple definiendo la matriz  $X$  de dimensión  $M \times d$  que contendrá cada una de las  $M$  palabras de la entrada en una fila. Esta matriz tendrá un número de columnas igual a la dimensión del espacio de características  $d$ . De este



**Figura 2.5:** Ejemplo de cálculo del vector de contexto en una frase de dos palabras. El proceso va desde la representación de cada palabra en su vector de características  $\mathbf{x}$  hasta el cálculo del vector de contexto  $\mathbf{z}$ . Nótese que únicamente se muestran las operaciones para el cálculo del vector de contexto de la primera palabra. [1]

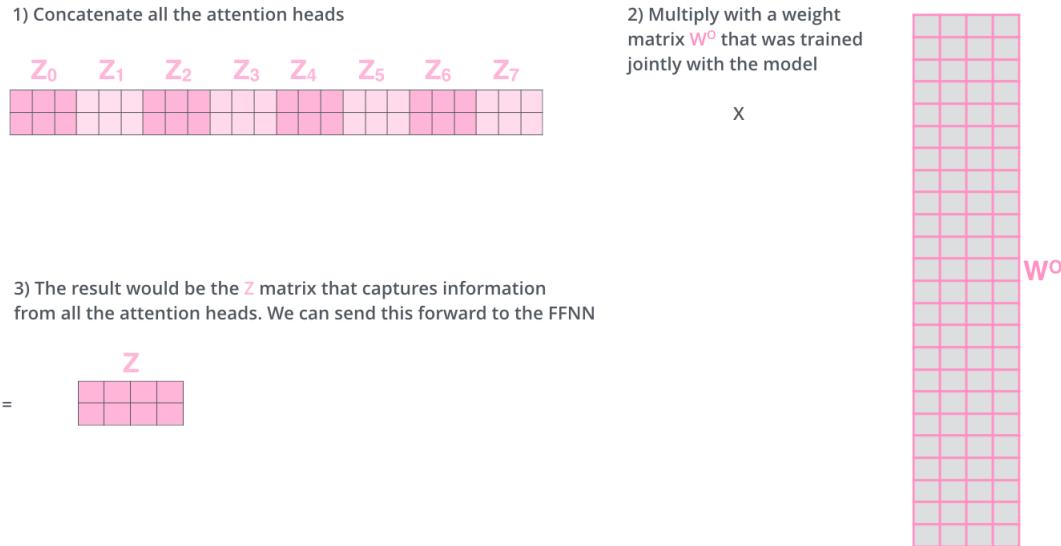
modo, los vectores  $\mathbf{q}$ ,  $\mathbf{k}$  y  $\mathbf{v}$  pasarán a ser también matrices  $Q$ ,  $K$  y  $V$  de dimensiones  $\dim(Q) = \dim(K) = M \times d_k$  y  $\dim(V) = M \times d_v$ .

$$\begin{aligned} Q &= XW^Q \\ K &= XW^K \\ V &= XW^V \end{aligned} \tag{2.9}$$

La matriz de contexto  $Z$ , de dimensión  $M \times d_v$ , se calcula del siguiente modo:

$$Z = \text{Softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \tag{2.10}$$

Esta capa se llama capa de atención múltiple porque porque el proceso que hemos descrito se realiza varias veces definiendo varias matrices de pesos  $W$ . De este modo, existen varias matrices  $Q$ ,  $K$  y  $V$  que durante el proceso de entrenamiento se encargan



**Figura 2.6:** Ejemplo de cálculo de la matriz de contexto usando atención múltiple. Se muestra el proceso para 8 cabezas de atención, de modo que se tienen 8 matrices de contexto  $Z^i$  diferentes que se combinan mediante otra matriz de pesos  $W^O$  para formar la matriz de contexto final,  $Z$ . [1]

de proyectar el input en diferentes subespacios de representación, capturando una mayor cantidad de características. En el modelo presentado en [14] se utilizan 8 cabezas de atención, por lo que se obtienen 8 matrices de contexto  $Z$  diferentes. Estas matrices se concatenan y se multiplican por otra matriz de pesos entrenables  $W^O$  para dar como resultado una única matriz de contexto  $Z$ . En la figura 2.6 se muestra este proceso.

Tras la capa de atención múltiple, el encoder cuenta con una capa feed forward, que no es más que una red neuronal donde las conexiones entre las neuronas no forman un ciclo. Esta capa aplica una transformación lineal a cada una de las posiciones de la secuencia de entrada por separado, seguida de una función de activación no lineal como la función *ReLU* (2.5). El hecho de que se aplique a cada una de las posiciones por separado permite al modelo procesar toda la secuencia de manera eficiente en paralelo.

Un importante detalle en la arquitectura del encoder de un Transformer es la presencia de conexiones residuales. Estas conexiones sirven para mantener la información original de la secuencia de entrada mientras se realiza el procesamiento en las capas del modelo. Estas conexiones permiten que la información original fluya a través de las capas, imposibilitando que se pierda. Cada capa del encoder tiene dos conexiones residuales, como vemos en la figura 2.4. La primera conexión residual conecta directamente la entrada de la capa con la salida de la capa de atención múltiple, mientras que la segunda conexión residual conecta la salida de la capa de atención múltiple con la entrada de la siguiente capa tras pasar por la capa de feed forward. Estas co-

nexiones ayudan a evitar el problema del desvanecimiento del gradiente y aceleran la convergencia del modelo durante el proceso de entrenamiento.

### 2.3.2. Decoder

El decoder del Transformer es la parte situada a la derecha en la figura 2.4. Su función es generar la salida final del modelo. Para ello, recibe como input el output del encoder y utiliza esta información para generar una secuencia de salida.

Podemos ver que, al igual que el encoder, el decoder está compuesto por  $N$  capas idénticas, aunque en este caso cada una de las capas cuenta con tres subcapas diferentes. Dos de estas tres subacapas son idénticas a las del encoder: la capa de atención múltiple y la capa de feed forward. La capa de atención múltiple enmascarada (*Masked Multi-Head Attention*) es muy similar a la capa de atención múltiple que hemos descrito en la sección anterior. La diferencia es que en este caso recibe como input la secuencia de salida que el transformer ha generado hasta el momento, de modo que sus outputs únicamente dependen de las palabras que el modelo ya ha generado previamente.

Con la salida del encoder, que será de dimensión  $M \times d_v$ , se construyen dos matrices  $K_{enc}$  y  $V_{enc}$  a través de matrices de pesos entrenables  $W$ . Las dimensiones de estas matrices serán  $\dim(K_{enc}) = M \times d_k$  y  $\dim(V_{enc}) = M \times d_v$ , donde  $M$  es el número de palabras que el transformer está procesando. Estas matrices serán las que pasan a través del decoder en la subcapa intermedia de atención múltiple.

Veamos cómo es el flujo de datos a través del decoder:

1. Masked Multi-Head Attention.

Esta subcapa toma como input la secuencia que el transformer ha generado hasta el momento para calcular la atención y el vector de contexto  $Z_{dec}$  de igual modo que sucedía en el encoder. Esta matriz  $Z_{dec}$  será de dimensión  $M' \times d_v$ , donde  $M'$  el número de palabras que el transformer ha generado hasta el momento. Con esta matriz y una matriz de pesos  $W$  se genera la matriz Query ( $Q_{dec}$ ) que pasa a la siguiente subcapa del decoder.

2. Multi-Head Attention.

Esta subcapa utiliza las matrices  $Q_{dec}$ ,  $K_{enc}$  y  $V_{enc}$  para calcular la matriz de contexto  $Z$ , que será de dimensión  $M' \times d_v$ . Esta matriz se pasa a la capa Feed Forward.

### 3. Feed Forward.

Por último se aplica una capa neuronal igual que la última capa del encoder, seguida de una transformación lineal y una función de activación *Softmax* que proporciona el output en forma de probabilidades. Este output, al igual que el input del encoder, es un embedding que podrá traducirse a palabras realizando la transformación inversa que se ha hecho antes de introducir las secuencia de palabras al modelo.

Nótese que este proceso se repite tantas veces como palabras haya en el output, ya que necesitamos las palabras generadas previamente para obtener la siguiente. En el encoder no teníamos este problema, pues todas las palabras de la secuencia input se procesan al mismo tiempo.

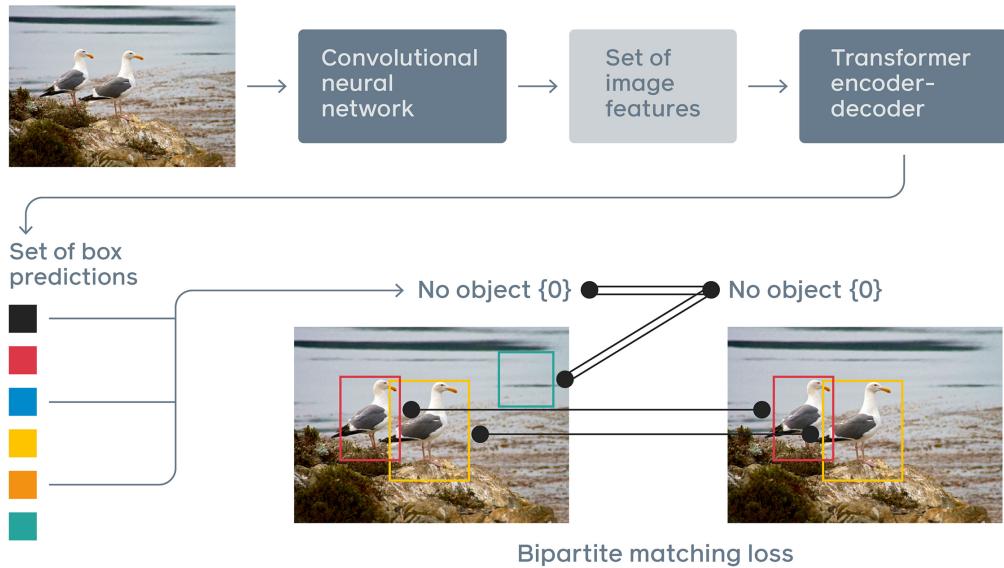
### 2.3.3. Transición a Computer Vision

En Computer Vision, los transformers se utilizan para procesar imágenes y capturar las relaciones a largo plazo entre los píxeles de la imagen. A diferencia de NLP, donde los datos se representan como secuencias de palabras, en computer vision, los datos se representan como matrices de píxeles. En consecuencia, los transformers necesitan adaptarse para procesar las entradas de imágenes de manera efectiva.

Cada uno de los modelos de visión que utilizan Transformers se ha adaptado de una forma diferente. Muchas veces existe una combinación entre redes neuronales convolucionales y capas de atención propias del Transformer.

Por ejemplo, el modelo Vision Transformer (ViT) que hemos mencionado en la introducción (figura 1.2) es la transferencia más directa de los Transformers desde NLP a computer vision. En la figura vemos que simplemente usamos los patches de la imagen como si fueran tokens de texto y los pasamos a través del encoder que hemos descrito anteriormente. Antes de introducirlos al encoder aplicamos un position embedding (igual que hacíamos en NLP) pues, de no hacer esto, no podríamos identificar la posición de cada patch de ninguna manera.

Otra forma de aplicar la arquitectura Transformer en Computer Vision es introducir una serie de características de la imagen extraídas con una CNN al Transformer. Esto es empleado por el modelo de detección de objetos DETR [4]. En la figura 2.7 podemos ver el flujo de datos desde que la imagen entra al modelo hasta que sale la predicción de las cajas que contienen los objetos.



**Figura 2.7:** Detection Transformer (DETR). [18]

El modelo de segmentación de imágenes Segformer, que explicaremos con detalle más adelante, también emplea una combinación de CNNs y capas de atención para llevar a cabo su tarea.

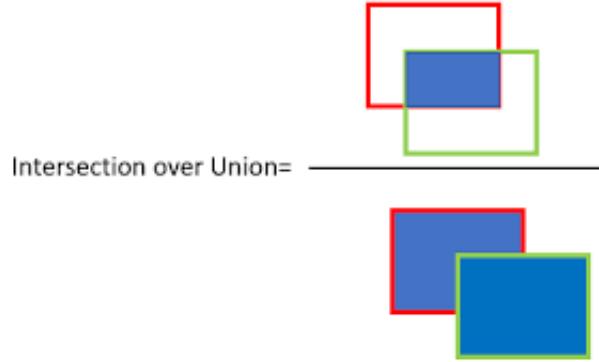
En resumen, existen múltiples formas de aplicar Transformers en computer vision y, como es un campo muy reciente, se espera que puedan surgir nuevas. La característica común de todos estos modelos es la aplicación de capas de atención.

## 2.4. Métricas

Para evaluar nuestros modelos de segmentación de imágenes vamos a tener en cuenta cuatro métricas diferentes: *Intersection over Union (IoU)*, *Precision*, *Recall* y *F1-score*. La elección de usar una métrica u otra al evaluar un modelo de segmentación depende del contexto y las necesidades específicas del problema. A continuación pasamos a describir cada una de las métricas y los casos en los que son de utilidad.

### 2.4.1. Intersection over Union

La métrica *Intersection over Union (IoU)*, también conocida como Jaccard Index, es una medida comúnmente utilizada para evaluar la precisión de la segmentación de imágenes. Esta métrica se calcula dividiendo el área de intersección entre la máscara



**Figura 2.8:** *Intersection Over Union.* [19]

de predicción y la máscara de verdad en la clase entre el área de unión entre ambas (figura 2.8). Matemáticamente, la podemos expresar como:

$$IoU = \frac{|T \cap P|}{|T \cup P|} \quad (2.11)$$

donde  $T$  es la máscara real y  $P$  es la máscara predicha por el modelo. Estos dos elementos serán matrices donde cada elemento se corresponderá con un píxel, por lo que serán de la misma dimensión que las imágenes. Los valores de  $T$  y  $P$  únicamente podrán ser 1 o 0, que indicarán la pertenencia, o no, de un píxel a una clase. Los símbolos  $| |$  es una medida de área y se pueden interpretar como la suma de todos los elementos de la matriz, pues todos los valores de la misma estarán comprendidos entre 0 y 1.

Un  $IoU$  de 1 significa que la predicción y la verdad se superponen completamente, mientras que un  $IoU$  de 0 indica que no hay ninguna superposición, por lo que el modelo no habría acertado la categoría de ningún píxel.

Para el caso de segmentación de imágenes con más de una categoría diferente, simplemente calculamos la media del  $IoU$  de cada clase por separado, obteniendo la métrica conocida como *Mean Intersection over Union (meanIoU)*.

#### 2.4.2. Precision

La precisión de un modelo de clasificación es una métrica que evalúa la capacidad del modelo para clasificar correctamente los casos positivos, es decir, la proporción de casos positivos clasificados correctamente en relación con el número total de casos clasificados como positivos. Se calcula dividiendo el número de verdaderos positivos (casos positivos

que el modelo clasifica correctamente,  $TP$ ) entre la suma de los verdaderos positivos y los falsos positivos (casos negativos que el modelo clasifica incorrectamente como positivos,  $FP$ ):

$$Precision = \frac{TP}{TP + FP} \quad (2.12)$$

La precisión mide la exactitud de las predicciones positivas de un modelo. Una precisión alta indica una baja tasa de falsos positivos y es una métrica relevante cuando lo que se quiere minimizar los falsos positivos.

Sin embargo, es importante tener en cuenta que la precisión por sí sola puede ser engañosa si no se considera en conjunto con otras métricas. Por ejemplo, un modelo que clasifica todos los casos como negativos obtendría una precisión del 100% pero tendría una capacidad de detección de casos positivos nula. Debido a esto, incluimos métricas como el *Recall* y el *F1-score*.

### 2.4.3. Recall

El *recall*, también conocido como sensibilidad, mide la proporción de casos positivos que el modelo clasifica correctamente en relación con el número total de casos positivos presentes en los datos de prueba. Se calcula dividiendo el número de verdaderos positivos ( $TP$ ) entre la suma de los verdaderos positivos y los falsos negativos ( $FN$ ):

$$Recall = \frac{TP}{TP + FN} \quad (2.13)$$

El recall proporciona información sobre la capacidad del modelo para encontrar todos los casos positivos, sin importar si clasifica algunos casos negativos incorrectamente como positivos. Un recall alto indica una baja tasa de falsos negativos.

Debido a que esta métrica no tiene en cuenta los falsos positivos, se suele usar en conjunto con otras métricas a la hora de evaluar un modelo de clasificación.

### 2.4.4. F1-score

El *F1-score* es una métrica común en problemas de clasificación que combina la precisión y el recall en un solo valor. Es una medida de equilibrio entre ambas métricas

y proporciona una evaluación más completa. Se calcula como:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.14)$$

Esta métrica tiene en cuenta tanto los falsos positivos como los falsos negativos y es de utilidad cuando hay que considerar tanto la precisión como el recall con la misma importancia. En el caso en el que se requiera más énfasis en la precisión o el recall se pueden usar variantes del *F1-score* como el *F-beta* que ajusta el peso relativo de ambas métricas.

## 2.5. Funciones de coste

La función de coste (o función de pérdidas) se utiliza en el entrenamiento de modelos para medir cómo de bien el modelo está haciendo las predicciones en comparación con los valores reales del conjunto de datos de entrenamiento. Esta función calcula la diferencia entre la salida predicha por el modelo y el valor real, por lo que el objetivo del entrenamiento es minimizar esta función de modo que el modelo cada vez se comporte de mejor manera. Esto se consigue ajustando los parámetros del modelo para minimizar esta función.

La elección de la función de coste es muy importante a la hora de entrenar un modelo, ya que tiene un impacto significativo en la capacidad del modelo para aprender patrones y hacer predicciones precisas. Esta función debe ser seleccionada cuidadosamente para que sea adecuada para el tipo de problema que se está abordando y para los datos que se están utilizando.

La función de coste más utilizada para resolver la tarea de segmentación de imágenes es la conocida como *Cross Entropy*. Sin embargo, para realizar el entrenamiento de los modelos de este trabajo se ha empleado una función de coste derivada de la métrica *meanIoU*, pues nos parecía interesante su uso para detectar aquellas clases menos representadas en los datasets.

### 2.5.1. Cross Entropy

La Cross Entropy es una función de coste comúnmente utilizada en el aprendizaje automático y el aprendizaje profundo para resolver problemas de clasificación. Se utiliza

cuando se trata de minimizar la diferencia entre dos distribuciones de probabilidad, una distribución de probabilidad de salida predicha por el modelo y otra distribución de probabilidad de los valores reales de las etiquetas. Por lo tanto, es particularmente útil en problemas de clasificación donde el objetivo es predecir la probabilidad de pertenencia de una observación a diferentes categorías.

La función de coste Cross Entropy ( $H$ ) se define como:

$$H(p, q) = - \sum_{i=1}^n p_i \log(q_i) \quad (2.15)$$

Donde:

- $n$  es el número de clases en el problema de clasificación.
- $p$  es el vector de distribución de probabilidad de las etiquetas reales (también conocido como *etiquetas one-hot*).
- $q$  es el vector de distribución de probabilidad predicha por el modelo.

### 2.5.2. *Mean Intersection over Union* como función de coste

La función *meanIoU* no es una función diferenciable y, por tanto, no es posible utilizarla como función de coste a la hora de entrenar un modelo, ya que no se puede propagar su gradiente. Esto es debido a que en la definición 2.11 los valores de  $T$  y  $P$  deben ser 0s o 1s, es decir, un pixel debe pertenecer a una categoría, o no, absolutamente. Sin embargo, las salidas de un modelo de Deep Learning suelen marcar probabilidades de pertenencia a la clase, que vienen dadas por valores entre 0 y 1. Para solucionar este problema, se puede hacer una aproximación de la *IoU* a través del concepto de probabilidad. En el artículo titulado “Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation” [2] proponen la siguiente aproximación para la segmentación de imágenes binaria:

$$IoU' = \frac{|T * P|}{|T + P - T * P|} = \frac{I}{U} \quad (2.16)$$

donde  $T$  y  $P$  toman el mismo significado que antes, salvo que en este caso los valores de  $P$  están comprendidos entre 0 y 1 indicando la probabilidad de pertenencia a la clase.  $T * P$  simboliza la multiplicación elemento a elemento de las matrices  $T$  y

$P$ . En el numerador,  $T * P$  es una aproximación de la intersección ( $I$ ), mientras que el denominador se una aproximación de la unión ( $U$ ).

Como  $T$  es la máscara de verdad, tendrá como valores 0 o 1 según el píxel pertenezca o no a la clase en cuestión. Los valores de  $P$  son probabilidades de pertenencia a la clase, por lo que idealmente valdrían 1 en las mismas posiciones en las que  $T = 1$ . En este caso ideal,  $|T * P| = |P| = |T|$ , por lo que  $IoU' = 1$ , lo que significa que la unión es igual a la intersección, tal y como es de esperar. En el caso real en el que los valores de  $P$  se sitúan entre 0 y 1, tendremos que  $|T * P| < |P|$ , ya que el la multiplicación elemento a elemento de  $T$  con  $P$  lo único que hace es anular elementos de  $P$ , pues los elementos de  $T$  pueden valer 0 o 1. Además, como los valores de  $P$  van entre 0 y 1,  $|T * P| < |T|$ . Esto significa que  $I < U$ , por lo que  $IoU' < 1$ .

De este modo tenemos una función que se comporta como la  $IoU$  y que además es diferenciable, por lo que puede ser usada para entrenar un modelo. Como queremos que la función sea menor a medida que el modelo obtiene mejores resultados de clasificación, debemos expresar la función de coste de la siguiente manera:

$$L_{IoU} = 1 - IoU' \quad (2.17)$$

La derivada parcial con respecto a una de las variables  $P_x$  se puede calcular fácilmente:

$$\frac{\partial L_{IoU}}{\partial P_x} = \frac{I * \frac{\partial U}{\partial P_x} - U * \frac{\partial I}{\partial P_x}}{U^2} = \frac{I * (1 - T_x) - U * T_x}{U^2} \quad (2.18)$$

Por lo tanto, se podrá propagar el gradiente y actualizar los pesos del modelo a través de la ecuación 2.2.

El razonamiento aplicado es para un problema de segmentación de imágenes binario, es decir, un píxel únicamente puede pertenecer a dos categorías diferentes. Sin embargo, es fácilmente extendible para el caso de clasificación en  $n$  categorías. Simplemente debemos realizar una media para cada categoría tratándolas de forma independiente.

$$L_{meanIoU} = 1 - meanIoU' = 1 - \frac{1}{n} \sum_{i=1}^n \frac{|T_i * P_i|}{|T_i + P_i - T_i * P_i|} \quad (2.19)$$

donde  $n$  es el número total de categorías. Nótese que la matriz  $T_i$  cambiará para cada una de las categorías, tomando siempre dos únicos valores en todas sus posiciones: 0 o 1.

# Capítulo 3

## Modelo Segformer

En este trabajo hemos estudiado el modelo de segmentación de imágenes basado en Transformers conocido como Segformer. Este modelo fue presentado en el año 2021 en un paper titulado *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers* [24] y ha demostrado ser muy efectivo en una variedad de tareas de segmentación de imágenes, incluyendo la segmentación de objetos, la segmentación semántica y la segmentación de instancias.

Este modelo hace uso de una combinación de CNNs, Transformers y perceptrón multicapa (MLP) y ha conseguido situarse en el estado del arte en tres datasets públicos de segmentación de imágenes ampliamente utilizados, como son Cityscapes, ADE20K y COCO-Stuff.

En esta sección tratamos en profundidad su arquitectura, así como el rendimiento de diferentes modelos preentrenados.

### 3.1. Arquitectura

El modelo consta de dos partes: un encoder y un decoder. El encoder es la parte más compleja del modelo que extrae las características de la imagen, mientras que el decoder es simplemente un Multilayer Perceptron (MLP) que aprovecha las características extraídas por el encoder para realizar la tarea objetivo. La arquitectura del modelo la mostramos en la figura 3.1.

A continuación pasamos a describir los diferentes elementos que conforman el mo-

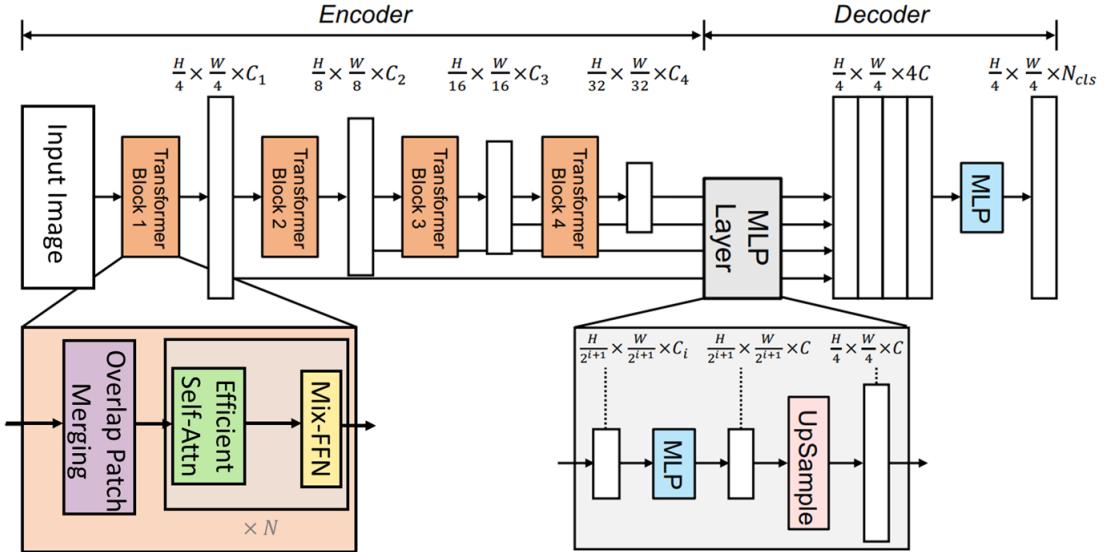


Figura 3.1: Arquitectura de modelo Segformer. [24]

delo Segformer.

### 3.1.1. Encoder

Un hecho destacable de este modelo es que, a diferencia de la arquitectura original del Transformer o el modelo de visión ViT, no cuenta con una capa de positional embedding. Para captar las relaciones espaciales entre los píxeles de una imagen, el modelo Segformer hace uso de una capa CNN como primera capa de entrada del modelo.

El encoder cuenta con una secuencia de bloques Transformer. A su vez, cada uno de estos bloques está formado por un capa CNN (Overlap Patch Merging) y  $N$  bloques formados por una capa de atención (Efficient Self-Attention) seguida de una combinación de capas densas y convolucionales (Mix-FFN). Cada vez que se pasa por un bloque Transformer, se genera una matriz de atención de resolución diferente. Esto es una estructura jerárquica que nos permite capturar características a alta y baja resolución.

Concretamente, si el modelo recibe una imagen de resolución  $H \times W \times 3$ , genera cuatro mapas de características de dimensión  $\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i$  con  $i \in \{1, 2, 3, 4\}$  y  $C_{i+1} > C_i$  a la salida de cada uno de los bloques Transformer. Esto mismo lo vemos indicado en la figura 3.1.

Describamos cada una de las capas presentes en el encoder:

## Overlap Patch Merging

Esta capa simplemente es una capa convolucional seguida de una capa de reshape. La capa convolucional es el sustituto del position embedding, pues es capaz de capturar la relación espacial entre los diferentes píxeles. El tamaño del kernel utilizado (patch) es clave para captar esta relación y, obviamente, no puede ser 1. En el modelo propuesto, la primera capa de este tipo tiene un kernel de dimensión  $7 \times 7$ , mientras que en el resto de capas es de dimensión  $3 \times 3$ . Además, para generar mapas de características a diferentes resoluciones, estas capas cuentan con un valor de stride ( $s$ ) diferente de 1. Concretamente, la capa del primer bloque cuenta con  $s = 4$ , mientras que en las siguientes capas  $s = 2$ .

Al aplicar la capa convolucional, las dimensiones de entrada cambian:

$$(\text{batch}, H, W, C) \rightarrow \left( \text{batch}, \frac{H}{s}, \frac{W}{s}, \text{Hidden\_size} \right)$$

donde  $H$ ,  $W$  y  $C$  es la dimensión de la imagen de entrada. *Hidden\_size* vendrá definido por el número de filtros que usemos a la hora de hacer la convolución. Esto es un parámetro del modelo y puede ser modificado en cada una de las capas. Por defecto este parámetro aumenta a medida que la capa es más profunda dentro de la arquitectura del modelo.

Tras esta capa convolucional existe una capa de reshape que aplica una transformación para pasar la información a la capa de atención. Esta transformación elimina una de las dimensiones del mapa de características:

$$\left( \text{batch}, \frac{H}{s}, \frac{W}{s}, \text{Hidden\_size} \right) \rightarrow \left( \text{batch}, \frac{H}{s} \times \frac{W}{s}, \text{Hidden\_size} \right)$$

## Efficient Self-Attention.

Esta capa recibe como input un tensor de dimensión  $(\text{batch}, \frac{H}{s} \times \frac{W}{s}, \text{Hidden\_size})$  procedente de la capa anterior y calcula las matrices Query ( $Q$ ), Key ( $K$ ) y Value ( $V$ ) a través de capas densas con un número de unidades igual a *Hidden\_size*. Si definimos el tensor input como  $X$  nos quedan las misma ecuaciones que hemos presentado en la

sección 2, donde las matrices  $W$  vienen dadas por las capas densas:

$$\begin{aligned} Q &= XW^Q \\ K &= XW^K \\ V &= XW^V \end{aligned} \tag{3.1}$$

De este modo, como las tres capas densas contienen el mismo número de unidades, tendremos que  $\dim(Q) = \dim(K) = \dim(V) = \text{batch} \times \frac{HW}{s^2} \times \text{Hidden\_size}$ . A la hora de realizar la implementación y tener en cuenta múltiples cabezas en la capa de atención, se aplica un reshape de todas las matrices, de modo que:

$$\left( \text{batch}, \frac{HW}{s^2}, \text{Hidden\_size} \right) \rightarrow \left( \text{batch}, \text{num\_attention\_heads}, \frac{HW}{s^2}, \text{attention\_head\_size} \right)$$

donde

$$\text{Hidden\_size} = \text{num\_attention\_heads} \times \text{attention\_head\_size} \tag{3.2}$$

A partir de aquí ya podemos calcular la matriz de contexto  $Z$ :

$$Z = \text{Softmax} \left( \frac{QK^T}{\sqrt{\text{attention\_head\_size}}} \right) V \tag{3.3}$$

Esta matriz de contexto tendrá la misma dimensión que las matrices  $Q$ ,  $K$  y  $V$ . A continuación, se realiza un reshape, de modo que obtengamos de nuevo la misma dimensión que la capa de atención ha recibido como input. Por lo tanto, al final de la capa de atención  $\dim(Z) = \text{batch} \times \frac{HW}{s^2} \times \text{Hidden\_size}$ .

Uno de los principales inconvenientes a la hora de emplear Transformers para resolver tareas de computer vision es el límite computacional, pues se debe trabajar con tensores de un tamaño enorme para calcular la atención que se prestan todos los píxeles con todos los píxeles. Para una imagen de resolución  $H \times W$ , la dimensión de la matriz de atención sería  $HW \times HW$  considerando que empleamos una única cabeza, lo que es un tamaño extremadamente costoso de manejar. Para resolver esto lo que se suele hacer es aplicar capas convolucionales con *stride* mayor que 1 antes de calcular la matriz de atención, tal y como se hace en las capas de Overlap Patch Merging de este modelo.

Además, para reducir todavía más el coste computacional, el modelo Segformer aplica un parámetro de reducción ( $s_r$ ) en las capas de atención, de ahí que sean llamadas *Efficient Self-Attention*. Este método de reducción consiste en aplicar una capa convolucional con *stride* =  $s_r$  sobre el tensor input  $X$  de las ecuaciones 3.1 a la hora de

formar las matrices  $K$  y  $V$ . Para ello, se aplica un reshape al tensor  $X$  con el objetivo de poder convolucionarlo correctamente y, tras la convolución, se aplica otro reshape para devolver a  $X$  a las dimensiones apropiadas.

De este modo, tendremos que:

$$\dim(K) = \dim(V) = \left( \text{batch}, \text{num\_attention\_heads}, \frac{HW}{s^2 s_r^2}, \text{attention\_head\_size} \right) \quad (3.4)$$

A la hora de calcular la matriz de contexto  $Z$  con la ecuación 3.3 la tercera dimensión de  $K$  y  $V$  se cancela, por lo que la dimensión de  $Z$  no se ve afectada por realizar esta reducción.

En el modelo propuesto, por defecto se presentan cuatro parámetros de reducción diferentes para las cuatro capas de atención, tomando los valores  $s_r = [8, 4, 2, 1]$

### Mix-FFN

Esta parte del bloque Transformer aplica sucesivamente una serie de capas diferentes que pasamos a enumerar según su orden de ejecución:

1. Capa densa con  $n = \text{Hidden\_size}$  unidades sobre la matriz  $Z$  de la capa de atención anterior. Como la última dimensión de la matriz  $Z$  es  $\text{Hidden\_size}$ , esta capa no afecta a la dimensión del flujo de datos.
2. Capa convolucional con número de filtros  $k = \text{Hidden\_size}$ ,  $stride = 1$  y  $kernel = 3$ . Nótese que para aplicar esta capa se debe realizar un reshape del tensor antes. Tras la convolución se vuelve a realizar el reshape para devolver el tensor a las mismas dimensiones.

Esta capa, junto con la de *Overlap Patch Merging* es clave a la hora de captar la relación espacial entre los diferentes píxeles.

3. Función de activación GELU (*Gaussian Error Linear Units*)

$$GELU(x) = x \frac{1}{2} \left[ 1 + \operatorname{erf}(x/\sqrt{2}) \right] \quad \text{con } \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \quad (3.5)$$

4. Capa densa con  $n = \text{Hidden\_size}$  unidades. Esta capa no modifica la dimensión del tensor, por tanto la dimensión de  $Z$  no cambia al pasar por la capa Mix-FFN.

A la salida de la última de las  $N$  capas Mix-FFN se aplica un reshape para obtener el

mapa de atención que se produce a la salida de cada uno de los bloques Transformer. Este mapa de atención es de dimensión:

$$\dim(\text{attention map}) = \text{batch} \times \frac{H}{s} \times \frac{W}{s} \times \text{Hidden\_size} \quad (3.6)$$

### 3.1.2. Decoder

El decoder está formado únicamente de capas MLP, lo que hace que sea muy sencillo y computacionalmente ligero. Este decoder emplea todos los mapas de atención que se han generado a la salida de cada uno de los bloques Transformer del encoder. A cada uno de estos mapas los llamamos  $F_i$  y tienen una resolución diferente:

$$\dim(F_i) = \frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i \text{ con } i \in \{1, 2, 3, 4\} \quad (3.7)$$

donde  $C_i$  es mayor a medida que  $i$  aumenta. El decoder del modelo Segformer cuenta con cuatro pasos que pasamos a enumerar.

1. Unificación de la dimensión de los canales.

Cada mapa de atención cuenta con un valor de  $C_i$  diferente, por lo que un primer paso es unificar esta dimensión de modo que todos tengan un valor  $C$ . Esto se puede conseguir pasando cada mapa de atención por un MLP de  $C$  unidades.

2. Upsampling.

A continuación se unifica el resto de dimensiones de los cuatro mapas de atención.

Para ello se realiza un Upsampling de modo que  $\dim(F_i) = \frac{H}{4} \times \frac{W}{4} \times C \forall i$ .

3. Concatenación y fusión de los mapas de atención.

Los mapas de atención se concatenan de modo que obtenemos un único tenso de dimensión  $\frac{H}{4} \times \frac{W}{4} \times 4C$ . Tras esto, se aplica un MLP con  $C$  unidades que fusiona la última dimensión, de modo que obtenemos un mapa de atención  $F$  de dimensión  $\frac{H}{4} \times \frac{W}{4} \times C$ .

4. Predicción de la máscara de segmentación.

El objetivo de esta capa es generar una dimensión que sea igual al número de categorías que presenta el problema de segmentación de imágenes, de modo que se pueda asignar un valor de probabilidad de pertenencia a una categoría para cada píxel. Para ello se emplea una capa densa de  $N_{cls}$  unidades, donde  $N_{cls}$  es el

Versión	Profundidades	Hidden Sizes	Dec. Hidden Size	P (M)
MiT-B0	[2, 2, 2, 2]	[32, 64, 160, 256]	256	3.7
MiT-B1	[2, 2, 2, 2]	[64, 128, 320, 512]	256	14.0
MiT-B2	[3, 4, 6, 3]	[64, 128, 320, 512]	768	25.4
MiT-B3	[3, 4, 18, 3]	[64, 128, 320, 512]	768	45.2
MiT-B4	[3, 8, 27, 3]	[64, 128, 320, 512]	768	62.6
MiT-B5	[3, 6, 40, 3]	[64, 128, 320, 512]	768	82.0

Cuadro 3.1: Diferentes versiones del modelo Segformer.

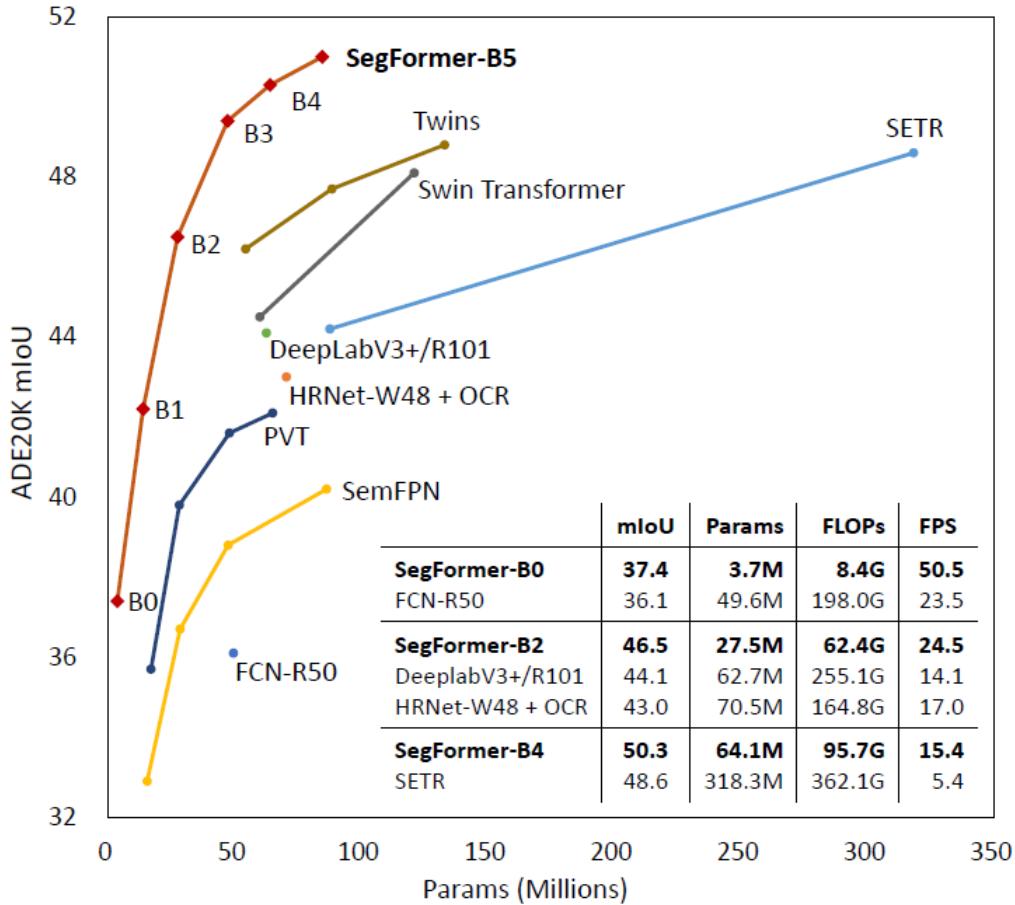
número de categorías del problema de segmentación. De este modo, la dimensión de salida del modelo es  $\frac{H}{4} \times \frac{W}{4} \times N_{cls}$ .

Nótese que para obtener la máscara de segmentación se debe realizar a continuación un Upsampling del output del modelo para obtener un tensor de dimensión  $H \times W \times N_{cls}$ .

## 3.2. Rendimiento

Según la complejidad del modelo en función del número de parámetros, se proponen seis arquitecturas, desde la MiT-B0 hasta la MiT-B5, donde MiT hace referencia a Mix Transformer encoder, que es como los autores se refieren a la arquitectura del encoder del modelo Segformer. En la tabla 3.1 se muestra la configuración de las diferentes versiones. La columna *Profundidades* hace referencia a al número de bloques de *Efficient Self-Attention* y *Mix-FFN* presentes en cada uno de los bloques Transformer del encoder. La columna *Hidden Sizes* indica las dimensiones  $C_i$  de los mapas de atención, mientras que la columna *Dec. Hidden Size* se refiere al número de unidades  $C$  de las capas densas empleadas en el decoder. Por último, en el campo *P (M)* indicamos el número de parámetros en millones de cada una de las versiones del modelo.

Este modelo ha sido probado en tres conocidos conjuntos de datos: Cityscapes [5], ADE20K [25] y COCO-Stuff [3]. En todos ellos ha conseguido resultados situados en el estado del arte de la segmentación de imágenes. Además, mejora el rendimiento del resto de modelos utilizando una menor cantidad de parámetros, lo que reduce significativamente el coste computacional de entrenamiento. En la figura 3.2 mostramos el rendimiento de las diferentes versiones del modelo Segformer en función del número de parámetros, así como una comparación de otros modelos de segmentación de imágenes. Podemos ver que este modelo supera el rendimiento (usando la *mIoU* como métrica) de su competencia con muchos menos parámetros. Los FLOPs (*Floating Point Operations*) es una medida de la cantidad de cálculos aritméticos de punto flotante que



**Figura 3.2:** Rendimiento vs. Eficiencia en el dataset ADE20K. Observamos que el modelo Segformer supera el rendimiento en términos de meanIoU de su competencia haciendo uso de un menor número de parámetros. [24]

realiza un modelo de Deep Learning en el paso de un batch a través de él, por lo que es una medida común de la complejidad computacional de un modelo de red neuronal. En la tabla de la figura vemos que esta medida es mucho menor en las variantes de los modelos Segformer.

Además, otro aspecto a destacar es que la mayor parte de los parámetros se encuentran en el encoder, por lo que es sencillo usar los modelos preentrenados con alguno de los tres conjuntos de datos mencionados y realizar un reentrenamiento únicamente del decoder para adaptar el modelo a unos datos nuevos.

# Capítulo 4

## Datos

En este trabajo se han utilizado tres conjuntos de datos diferentes: Cityscapes, GTA V y Mapillary Vistas. El modelo Segformer inicial está preentrenado en Cityscapes, y faremos un fine tuning con las imágenes sintéticas procedentes del dataset GTA V. Tras esto, realizaremos el experimento en la otra dirección entrenando el modelo en GTA V y haciendo fine tuning con imágenes reales de Cityscapes. Comprobaremos si alguno de estos fine tunings mejora los resultados en la segmentación de imágenes de un dataset más complejo, como es el dataset Mapillary Vistas.

### 4.1. Cityscapes

El conjunto de datos Cityscapes es una amplia colección de imágenes etiquetadas para resolver la tarea de segmentación de imágenes en entornos urbanos. Estas imágenes son fotografías de calles y áreas urbanas capturadas de 50 ciudades de Alemania (4.1) diferentes en las distintas estaciones del año y siempre de día.

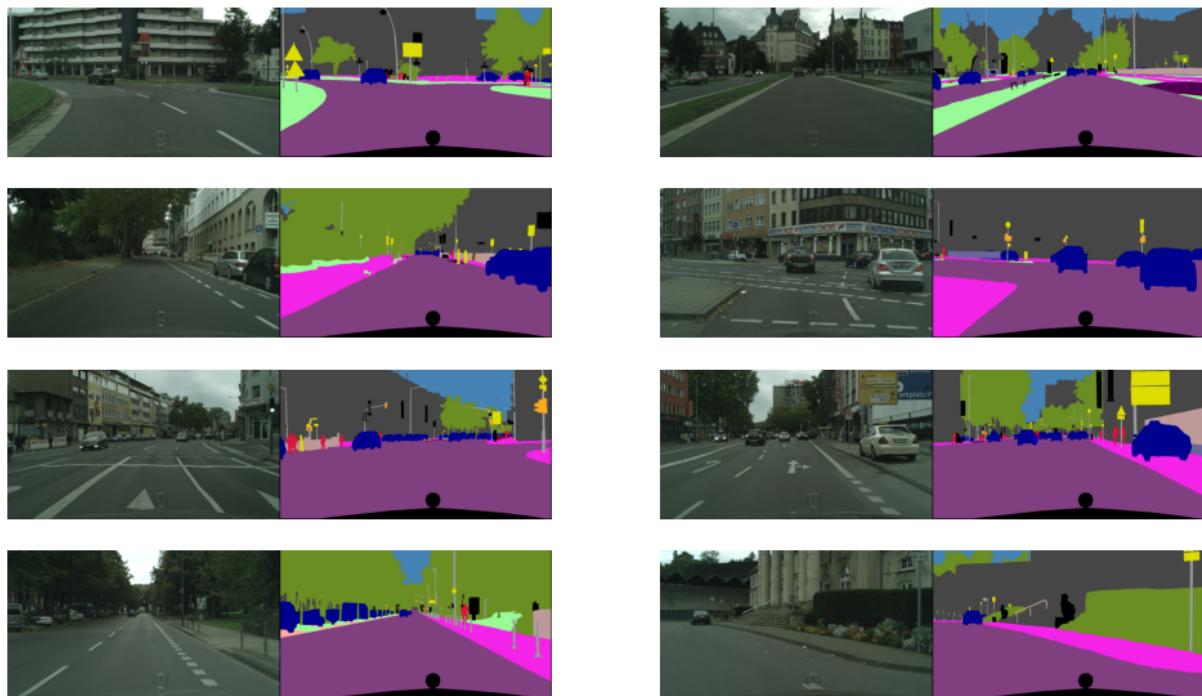
Cada imagen del conjunto de datos presenta una resolución de  $1024 \times 2048$  píxeles y están divididas en entrenamiento, test y validación, contando en total con 5000 imágenes etiquetadas píxel a píxel para la segmentación. Cada uno de los píxeles está clasificado en una de las 30 categorías presentes en Cityscapes, que pertenecen a 8 grupos diferentes. Estos grupos los mostramos en la tabla 4.1.

En la figura 4.4 mostramos ejemplos etiquetados tomados de diferentes ciudades, donde cada categoría se muestra etiquetada con un color diferente. La equivalencia entre las categorías y los colores la podemos ver en la tabla 4.2.



**Figura 4.1:** Ciudades alemanas presentes en el dataset Cityscapes. [16]

Imágenes junto con sus máscaras de segmentación



**Figura 4.2:** Ejemplo de varias imágenes del dataset Cityscapes, cada una de ellas junto con su máscara de segmentación. La equivalencia entre el color de la máscara de segmentación y la categoría se muestra en la tabla 4.2.

Sin embargo, este número de categorías es muy grande para entrenar nuestro modelo de segmentación de imágenes, por lo que han sido reducidas a 20, que mostramos en la tabla 4.2.

Grupo	Clases
Flat	Road, Sidewalk, Parking, Rail track
Human	Person, Rider
Vehicle	Car, Truck, Bus, On rails, Motorcycle, Bicycle, Caravan, Trailer
Construction	Building, Wall, Fence, Guard rail, Bridge, Tunnel
Object	Pole, Pole group, Traffic Sign, Traffic light
Nature	Vegetation, Terrain
Sky	Sky
Void	Ground, Dynamic, Static

Cuadro 4.1: Categorías en las que se dividen los píxeles de la imágenes del dataset Cityscapes

Categoría	Categoría Reducida	Color
Unlabeled		
Ground		
Dynamic		
Static		
Parking		
Rail track		
Guard rail	Void	
Bridge		
Tunnel		
Pole group		
Caravan		
Trailer		
Road	Road	
Sidewalk	Sidewalk	
Building	Building	
Wall	Wall	
Fence	Fence	
Pole	Pole	
Traffic light	Traffic light	
Traffic sign	Traffic sign	
Vegetation	Vegetation	
Terrain	Terrain	
Sky	Sky	
Person	Person	
Rider	Rider	
Car	Car	
Truck	Truck	
Bus	Bus	
Train	Train	
Motorcycle	Motorcycle	
Bicycle	Bicycle	

Cuadro 4.2: Categorías reducidas del dataset Cityscapes junto con su color de etiqueta correspondiente.

## CAPÍTULO 4. DATOS

---

Con el objetivo de minimizar el coste computacional, el tamaño de todas las imágenes ha sido reducido a  $526 \times 957$  píxeles.

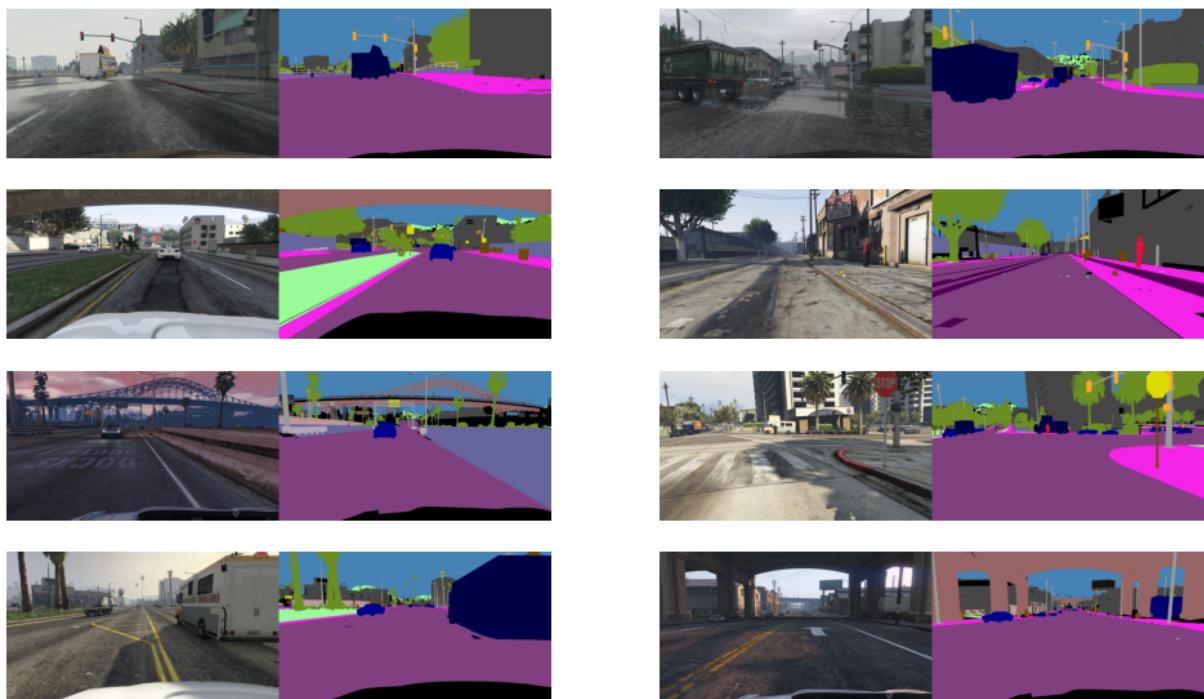
## 4.2. GTA V

El dataset GTA V ([11]) consiste en imágenes sintéticas extraídas del conocido videojuego de mundo abierto Grand Theft Auto V. Gran parte de este videojuego transcurre en una ciudad, por lo que tenemos imágenes de características similares a las imágenes del dataset Cityscapes en cuanto a distribución de los elementos. La gran diferencia es que en este caso se trata de imágenes sintéticas, que son mucho más fáciles de conseguir que las imágenes reales. La inclusión de este tipo de imágenes para entrenar un modelo de Deep Learning, concretamente un modelo basado en Transformers, solventaría uno de los principales problemas que es la gran cantidad de datos necesaria para alcanzar buenos resultados.

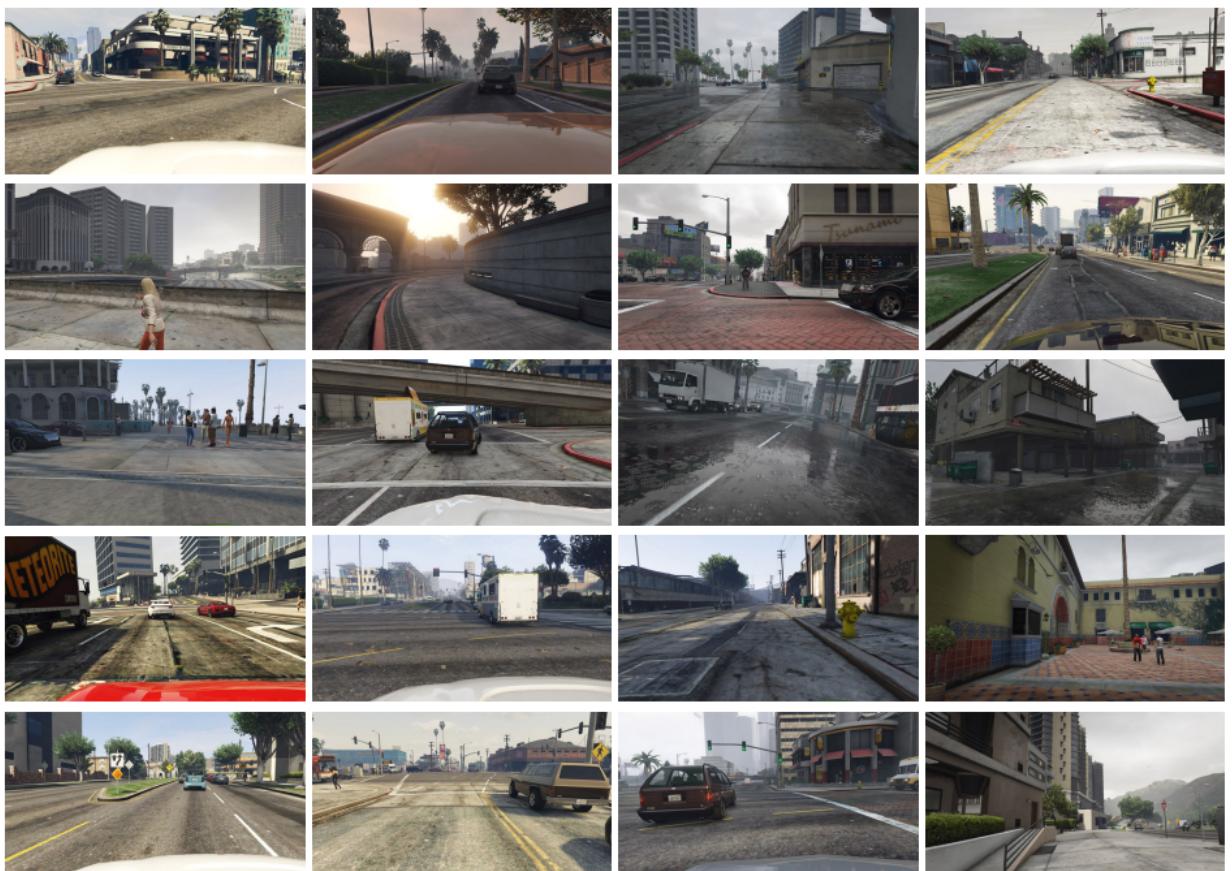
Este conjunto de datos viene etiquetado para resolver la tarea de segmentación de imágenes con las mismas categorías que el dataset Cityscapes (4.2). Esto hace que sea sencillo entrenar el modelo en GTA V y realizar finetuning en Cityscapes, o viceversa, de modo que podamos emplear la facilidad de conseguir imágenes sintéticas para mejorar el rendimiento del modelo en imágenes reales. En total tenemos 25000 imágenes anotadas píxel a píxel en una de las 20 categorías de Cityscapes, aunque para este trabajo han sido utilizadas únicamente 6500 debido a la falta de poder computacional. En la figura 4.3 mostramos varios ejemplos de estas imágenes y sus correspondientes máscaras de segmentación.

Las imágenes de GTA V son muy diversas, pues en el videojuego existe la posibilidad de ir andando o en coche, a diferencia de las imágenes del dataset Cityscapes que siempre son desde la perspectiva de un coche. Esto lo podemos observar en las imágenes de la figura 4.4, donde distinguimos múltiples escenas diferentes. Por este motivo vamos a ver si podemos usar las imágenes de GTA V para mejorar los resultados del modelo Segformer preentrenado en Cityscapes para un dataset de mayor complejidad como es el dataset Mapillary.

Imágenes junto con sus máscaras de segmentación



**Figura 4.3:** Ejemplo de varias imágenes del dataset *GTA V*, cada una de ellas junto con su máscara de segmentación.



**Figura 4.4:** Ejemplo de imágenes extraídas aleatoriamente del dataset GTA V. En ellas se puede apreciar la gran diversidad de imágenes que contiene este conjunto de datos, representando múltiples escenas desde diferentes perspectivas. [17]

### 4.3. Mapillary

El conjunto de datos Mapillary Vistas es uno de los datasets de segmentación de imágenes más grandes que contiene imágenes de alta resolución etiquetadas de forma manual y que cubren una gran variedad de escenas urbanas y rurales. Contiene más de 25000 imágenes etiquetadas píxel a píxel en más de 60 categorías diferentes para poder entrenar modelos de segmentación. En la figura 4.5 mostramos algún ejemplo de estas imágenes y sus máscaras.

Imágenes junto con sus máscaras de segmentación



**Figura 4.5:** Ejemplo de varias imágenes del dataset Mapillary Vistas, cada una de ellas junto con su máscara de segmentación.

Como nuestros modelos están entrenados únicamente para segmentar las imágenes en las 20 categorías de Cityscapes y GTA V, debemos reducir el número de categorías de Mapillary. El mapeo de categorías que hemos empleado ha sido extraído de [7] y lo mostramos en la figura 4.6. Las categorías de Mapillary no presentes en la figura 4.6 han sido etiquetadas como *Void*, de modo que no las tendremos en cuenta para analizar el rendimiento de los modelos.

<b>ID<sub>City</sub></b>	<b>Class</b>	<b>ID<sub>MV</sub></b>	<b>Class</b>
0	Road	13, 24, 41	Road, Lane Marking - General, Manhole
1	Sidewalk	2, 15	Curb, Sidewalk
2	Building	17	Building
3	Wall	6	Wall
4	Fence	3	Fence
5	Pole	45, 47	Pole, Utility Pole
6	Traffic Light	48	Traffic Light
7	Traffic Sign	50	Traffic Sign (Front)
8	Vegetation	30	Vegetation
9	Terrain	29	Terrain
10	Sky	27	Sky
11	Person	19	Person
12	Rider	20, 21, 22	Bicyclist, Motorcyclist, Other Rider
13	Car	55	Car
14	Truck	61	Truck
15	Bus	54	Bus
16	Train	58	On Rails
17	Motorcycle	57	Motorcycle
18	Bicycle	52	Bicycle

**Figura 4.6:** Mapeo de categorías realizado para reducir las 66 categorías de Mapillary a las 19 de Cityscapes y GTA V. A la izquierda las categorías de Cityscapes y a la derecha las de Mapillary. Las categorías de Mapillary no presentes en esta tabla han sido puestas como Void. [7]

#### 4.4. Desbalanceo de clases

En todos estos conjuntos de datos las clases están desbalanceadas. Hablamos de desbalance de clases cuando cuando la distribución de las clases objetivo dentro de un conjunto de datos de entrenamiento está significativamente desequilibrada. Esto significa que hay una gran disparidad en el número de ejemplos disponibles para cada clase.

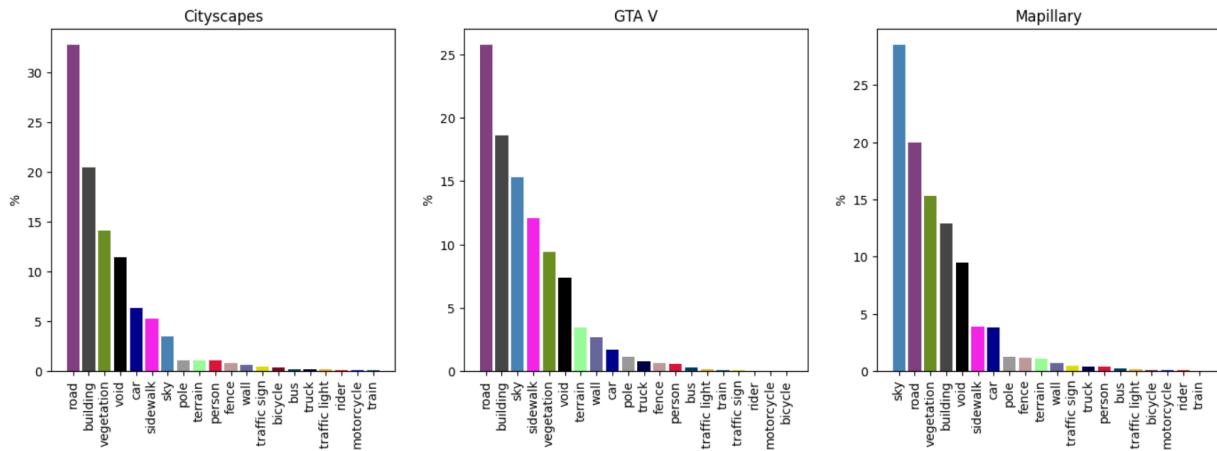
En los tres datasets usados en este trabajo existen clases que están mucho menos representadas que otras. Por ejemplo, la clase *carretera* aparece en prácticamente todas las imágenes representando un alto porcentaje de las mismas, mientras que clases como *tren* aparecen en un bajo porcentaje de las imágenes. En la tabla 4.3 y en la figura 4.7 mostramos cómo se distribuyen en porcentaje las clases en cada uno de los datasets, utilizando 1000 imágenes de cada uno de ellos como muestra.

El desbalanceo de clases puede tener varios efectos negativos a la hora de entrenar un modelo de segmentación de imágenes:

1. El modelo no aprende a segmentar correctamente las clases minoritarias debido a que no cuenta con suficientes ejemplos de estas clases.

Clases	Cityscapes (%)	GTA V (%)	Mapillary (%)
road	32.76	25.77	20.02
building	20.42	18.65	12.94
vegetation	14.12	9.38	15.35
void	11.43	7.36	9.46
car	6.37	1.65	3.77
sidewalk	5.27	12.11	3.87
sky	3.45	15.29	28.52
pole	1.06	1.12	1.24
terrain	1.04	3.41	1.10
person	1.03	0.55	0.35
fence	0.75	0.63	1.18
wall	0.65	2.65	0.69
traffic sign	0.48	0.08	0.48
bicycle	0.36	0.01	0.09
bus	0.18	0.29	0.23
truck	0.18	0.80	0.40
traffic light	0.17	0.13	0.18
rider	0.11	0.02	0.06
motorcycle	0.09	0.01	0.08
train	0.08	0.10	0.01

**Cuadro 4.3:** Distribución de clases en cada uno de los datasets. Se muestra el porcentaje de píxeles sobre el total de cada una de las clases, tomando como muestra 1000 imágenes de cada conjunto de datos.



**Figura 4.7:** Representación visual de la tabla 4.3

2. El modelo predice siempre las clases mayoritarias con alta probabilidad sin prestar atención a las clases minoritarias, ya que esto hace disminuir la función de coste empleada.

A la vista de la figura 4.7 existen clases que están muy poco representadas en los tres conjuntos de datos. Estas clases serán mucho más difíciles de detectar por nuestros modelos de segmentación de imágenes.

Para reducir este problema existen diferentes técnicas, como aumentar artificialmente el número de muestras minoritarias (difícil de conseguir en segmentación de imágenes) o escoger una función de coste que preste más atención a las clases menos representadas.



# Capítulo 5

## Experimentos

Para los experimentos de este trabajo hemos usado la arquitectura Segformer con el menor número de parámetros (MiT-B0) con el objetivo de reducir el coste computacional. Estudiaremos el resultado obtenido por varias versiones de este modelo a la hora de realizar inferencia sobre el dataset Mapillary Vistas.

Los modelos que trataremos son:

1. Modelo entrenado al completo en Cityscapes.
2. Modelo entrenado en Cityscapes y Fine Tuning del decoder en GTA V.
3. Modelo entrenado en Cityscapes y Fine Tuning completo en GTA V.
4. Modelo entrenado al completo en GTA V.
5. Modelo entrenado en GTA V y Fine Tuning del decoder en Cityscapes.
6. Modelo entrenado en GTA V y Fine Tuning completo en Cityscapes.

Para entrenar estos modelos contamos con 6500 imágenes de GTA V y con 2975 imágenes de Cityscapes, dejando en ambos datasets un 95 % de las imágenes para entrenamiento y un 5 % para validación. Antes de pasar por los modelos, las imágenes son normalizadas y reescaladas, de modo que todas ellas pasan a una resolución de  $512 \times 512$ .

La inferencia se ha realizado para 500 imágenes aleatorias del dataset Mapillary. Analizaremos las cuatro métricas descritas en la sección 2.4 para evaluar la segmentación de cada clase: *meanIoU*, *precision*, *recall* y *F1-score*.

El entrenamiento de todos los modelos, así como la inferencia sobre Mapillary, se ha realizado aprovechando el entorno de GPU que ofrece Google Colab en su versión Pro. A continuación pasamos a mostrar los resultados de cada uno de los modelos.

## 5.1. Modelo entrenado al completo en Cityscapes

Este modelo preentrenado está proporcionado por NVIDIA y ha sido extraído de la página web Hugging Face ([22]). El encoder de este modelo está inicializado con un entrenamiento sobre el dataset Imagenet-1K [13]. Tras esta inicialización, se realiza el entrenamiento del modelo completo con el dataset Cityscapes a una resolución de  $1024 \times 1024$ .

Antes de que las imágenes pasen por el modelo existe un extractor de características que las normaliza y las reduce a una resolución de  $512 \times 512$ . Por lo tanto, antes de realizar la inferencia sobre las imágenes de Mapillary debemos repetir estas operaciones de normalización y reescalado.

Este modelo está preentrenado para predecir todas las clases de Cityscapes excepto la clase *Void*, por lo que ningún píxel será clasificado como tal. En la figura 5.1 mostramos el resultado de la segmentación obtenido por este modelo para alguna imagen de Mapillary. Podemos ver que el resultado es bastante bueno, detectando con precisión prácticamente todas las clases.

En cuanto a las métricas analizadas, mostramos los resultados en la tabla 5.1. Tal y como es de esperar, se obtienen mejores resultados para aquellas categorías más presentes en todos los datasets. Con una amplia diferencia, la categoría mejor detectada es *sky*, que es la categoría con mayor presencia en Mapillary y, además, entra dentro de las categorías más representadas en Cityscapes. Por tanto, se trata de una categoría bien detectada y con mucha presencia, por lo que el valor de las distintas métricas es muy elevado.



**Figura 5.1:** Máscaras de segmentación resultado de realizar inferencia del modelo preentrenado en Cityscapes sobre el dataset Mapillary.

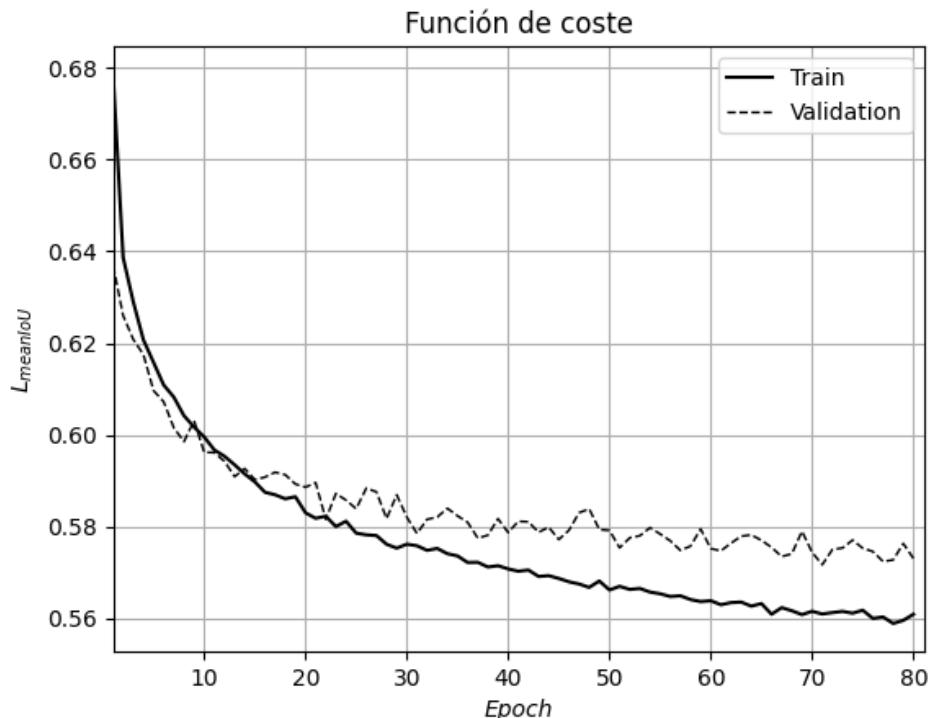
	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.94	0.98	0.96	0.97
vegetation	0.81	0.87	0.91	0.89
car	0.77	0.85	0.89	0.87
road	0.74	0.76	0.96	0.85
building	0.70	0.75	0.92	0.82
train	0.46	0.49	0.89	0.63
bus	0.45	0.79	0.51	0.62
person	0.42	0.51	0.72	0.60
terrain	0.42	0.46	0.80	0.59
traffic sign	0.36	0.55	0.52	0.53
sidewalk	0.35	0.60	0.46	0.52
rider	0.35	0.52	0.51	0.52
pole	0.32	0.59	0.41	0.48
motorcycle	0.32	0.53	0.44	0.48
truck	0.30	0.61	0.38	0.47
traffic light	0.30	0.67	0.36	0.47
bicycle	0.29	0.34	0.67	0.45
fence	0.28	0.41	0.46	0.44
wall	0.20	0.27	0.46	0.34
<b>Media</b>	<b>0.75</b>	<b>0.82</b>	<b>0.89</b>	<b>0.85</b>

**Cuadro 5.1:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 preentrenado en Cityscapes sobre 500 imágenes del dataset Mapillary Vistas.*

## 5.2. Modelo entrenado en Cityscapes y Fine Tuning del decoder en GTA V

Este modelo consiste en un Fine Tuning del modelo presentado en 5.1 con 6500 imágenes del dataset GTA V, usando 6175 para entrenamiento y 325 para validación. Únicamente se entranan los parámetros pertenecientes al decoder del modelo Segformer (ver figura 3.1) que suman un total de 400K, de modo que se pretende que el modelo conserve lo aprendido del dataset Cityscapes dejando congelado el encoder y se adapte a escenas más diversas a través de las imágenes de GTA V.

El modelo ha sido entrenado durante 80 épocas con un *batch* de 5 imágenes, la función de coste derivada de la métrica *meanIoU* (sección 2.5.2) y el optimizador *Adams*. La reducción en la función de coste, tanto para las imágenes de entrenamiento como para las imágenes de validación, la mostramos en la figura 5.2.



**Figura 5.2:** Función de coste durante las 80 épocas del entrenamiento del decoder del modelo MiT-B0 preentrenado en Cityscapes con 6500 imágenes del dataset GTA V. Con línea continua se representa la función de coste en imágenes de entrenamiento y con línea discontinua en imágenes de validación.

En la figura 5.2 podemos observar que la función de coste se estabiliza alrededor de  $L_{meanIoU} = 0.56$  para las imágenes de entrenamiento. El resultado de la inferencia para unas cuantas imágenes de Mapillary lo mostramos en la figura 5.3



**Figura 5.3:** Máscaras de segmentación resultado de realizar inferencia del modelo Fine Tuning decoder GTA sobre el dataset Mapillary.

A simple vista de la figura 5.3, vemos que las máscaras predichas por este modelo se ajustan peor que las máscaras predichas por el modelo únicamente entrenado en Cityscapes. Sobre todo se aprecia en las categorías *sidewalk*, *vegetation* y *traffic sign*.

En lo que respecta a las métricas empleadas para evaluar el rendimiento del modelo, en la tabla 5.2 exponemos los valores de *IoU*, *Precision*, *Recall* y *F1-score* para las 500 imágenes de Mapillary y las diferentes categorías.

Los resultados son generalmente peores a los que arroja el modelo únicamente entrenado en Cityscapes, tal y como anticipábamos al ver las máscaras de segmentación. Obtenemos valores inferiores para los valores medios de las cuatro métricas analizadas.

	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.93	0.96	0.96	0.96
car	0.77	0.86	0.87	0.87
vegetation	0.76	0.89	0.84	0.86
building	0.72	0.83	0.84	0.84
road	0.71	0.82	0.84	0.83
person	0.45	0.60	0.65	0.63
bus	0.40	0.77	0.45	0.57
terrain	0.32	0.34	0.87	0.49
pole	0.31	0.62	0.38	0.47
sidewalk	0.31	0.39	0.60	0.47
traffic light	0.28	0.71	0.31	0.43
fence	0.27	0.42	0.42	0.42
truck	0.26	0.40	0.41	0.41
rider	0.25	0.44	0.36	0.40
motorcycle	0.23	0.56	0.28	0.37
wall	0.19	0.27	0.42	0.33
bicycle	0.16	0.49	0.19	0.28
traffic sign	0.15	0.66	0.16	0.25
train	0.02	0.02	0.81	0.05
<b>Mean</b>	0.70	0.80	0.85	0.82

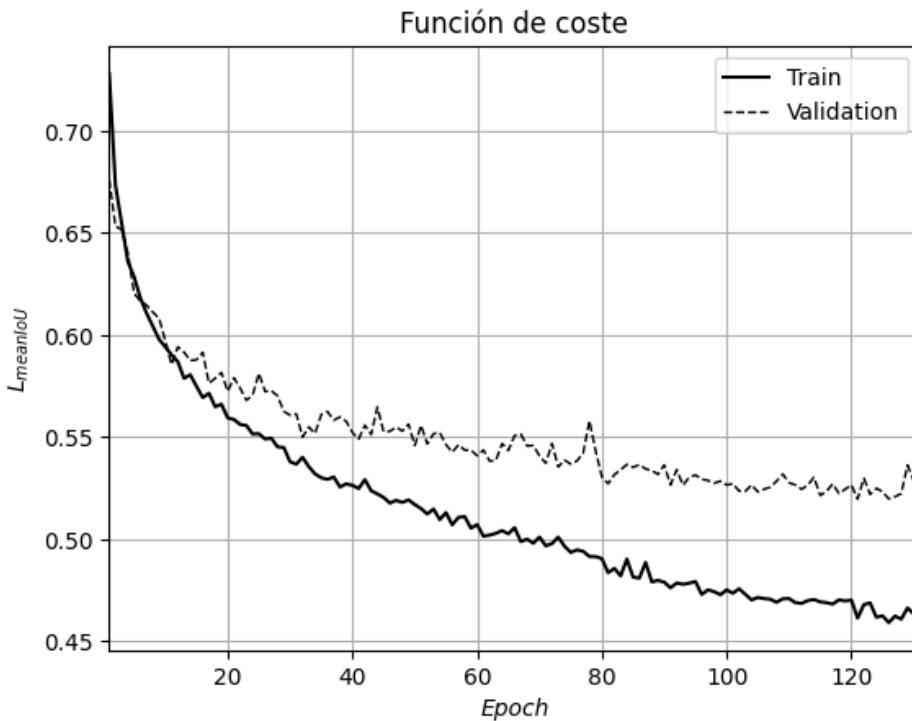
**Cuadro 5.2:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 Fine Tuning decoder GTA sobre 500 imágenes del dataset Mapillary Vistas.*

Sin embargo, este modelo mejora al anterior en alguna categoría como *building* o *person* si nos fijamos en la métrica *IoU* o *F1-score*.

### 5.3. Modelo entrenado en Cityscapes y Fine Tuning completo en GTA V

En este caso entrenamos el modelo MiT-B0 preentrenado en Cityscapes (5.1) de igual forma a la que presentamos en la sección 5.2, salvo que en este caso no congelamos el encoder del modelo durante el proceso de entrenamiento, de modo que reentrenamos la totalidad de los parámetros, sumando un total de 3.7 M. El objetivo de este entrenamiento es entrenar los bloques Transformer que únicamente están situados en el encoder del modelo Segformer (ver figura 3.1) de modo que aprendan la distribución espacial de los elementos de las imágenes del dataset GTA V.

Al igual que antes empleamos como función de coste la derivada de la métrica *meanIoU* (sección 2.5.2), el optimizador *Adams* y un *batch* de 5 imágenes durante un total de 130 épocas. La función de coste durante el proceso de entrenamiento la mostramos en la figura 5.4.



**Figura 5.4:** Función de coste durante las 130 épocas de entrenamiento del modelo MiT-B0 preentrenado en Cityscapes con 6500 imágenes del dataset GTA V. Con línea continua se representa la función de coste en imágenes de entrenamiento y con línea discontinua en imágenes de validación.

La máscara de segmentación resultante de la inferencia sobre Mapillary la exponemos en la figura 5.5. En ella podemos ver que los resultados son bastante peores que

los obtenidos por el modelo inicial, por lo que este entrenamiento no aportaría ningún valor para conseguir el objetivo. Lo más destacable es la dificultad que presenta el modelo a la hora de detectar la categoría *sky* en alguna de las imágenes. Esto indica que el aspecto del cielo del videojuego se aleja bastante de la realidad.



**Figura 5.5:** Máscaras de segmentación resultado de realizar inferencia del modelo Fine Tuning completo en GTA sobre el dataset Mapillary.

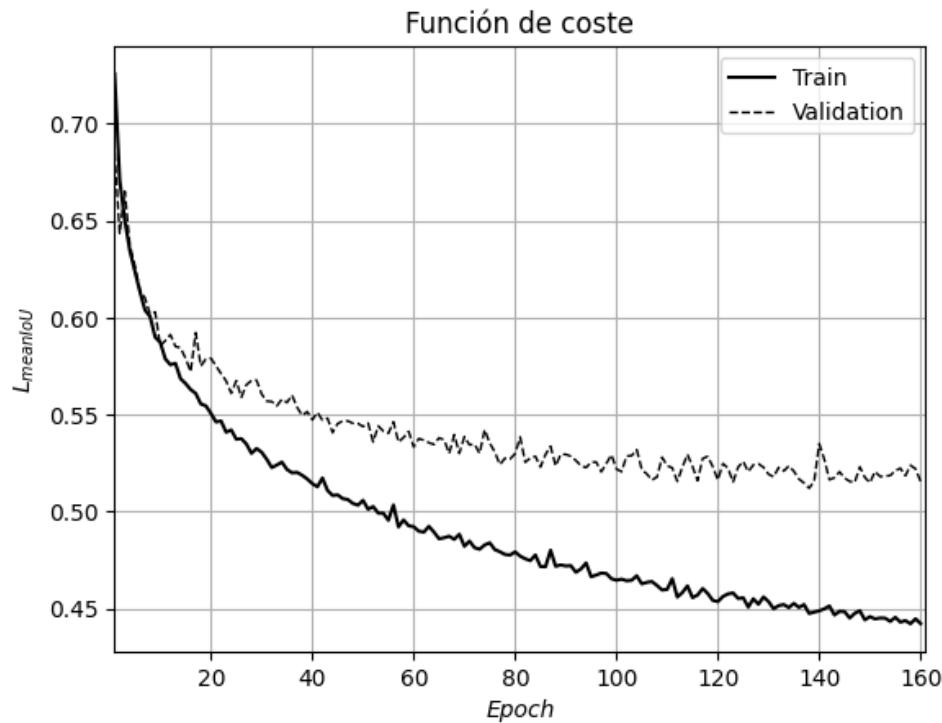
	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.75	0.94	0.79	0.86
building	0.64	0.77	0.79	0.78
road	0.63	0.67	0.92	0.77
vegetation	0.63	0.87	0.69	0.77
car	0.61	0.89	0.66	0.76
person	0.40	0.62	0.53	0.58
terrain	0.27	0.28	0.82	0.42
sidewalk	0.25	0.46	0.35	0.40
pole	0.23	0.68	0.26	0.38
traffic light	0.14	0.79	0.15	0.25
wall	0.14	0.19	0.35	0.25
fence	0.12	0.24	0.19	0.21
rider	0.10	0.43	0.12	0.18
truck	0.09	0.22	0.12	0.16
traffic sign	0.08	0.88	0.08	0.15
motorcycle	0.07	0.64	0.07	0.12
bicycle	0.05	0.38	0.06	0.10
bus	0.01	0.36	0.01	0.01
train	0.00	0.00	0.13	0.01
<b>Mean</b>	<b>0.58</b>	<b>0.73</b>	<b>0.75</b>	<b>0.74</b>

**Cuadro 5.3:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 Fine Tuning completo en GTA sobre 500 imágenes del dataset Mapillary Vistas.*

## 5.4. Modelo entrenado al completo en GTA V

Este modelo consiste en un entrenamiento completo del modelo Segformer MiT-B0 con las 6500 imágenes del dataset GTA V, de nuevo dejando un 5 % de las imágenes para validación. Al igual que sucedía con el modelo únicamente entrenado en Cityscapes, hemos partido de un modelo cuyo encoder está inicializado con el entrenamiento sobre el dataset Imagenet-1K [13], lo que acelera drásticamente la convergencia de la función de pérdidas. Durante el proceso de entrenamiento el modelo extraerá y aprenderá las características de las imágenes sintéticas de GTA V.

El entrenamiento se ha realizado durante 160 épocas con el optimizador *Adams* y un *batch* de 5 imágenes. La función de coste la mostramos en la figura 5.6, donde podemos observar que la convergencia se produce con un número de épocas mayor que en el resto de entrenamientos. Esto es debido a que el modelo debe aprender todas las características del dataset GTA V sin partir de una buena inicialización como es el dataset Cityscapes.



**Figura 5.6:** Función de coste durante las 160 épocas de entrenamiento del modelo Segformer MiT-B0 con 6500 imágenes de GTA V. Con línea continua se representa la función de coste en imágenes de entrenamiento y con línea discontinua en imágenes de validación.

La función de coste llega a ser de  $L_{meanIoU} = 0.4423$  para imágenes de entrenamiento y de  $L_{meanIoU} = 0.5156$  para imágenes de validación, que es un mejor resultado que para el caso del modelo preentrenado en Cityscapes y fine tuning completo en GTA V

## CAPÍTULO 5. EXPERIMENTOS

(5.3), lo que quiere decir que este modelo segmenta mejor las imágenes de GTA V, tal y como es de esperar, pues no está influenciado por las imágenes reales de Cityscapes.

En la figura 5.7 vemos las máscaras de segmentación predichas por este modelo sobre imágenes del dataset Mapillary. A simple vista observamos que el resultado es notablemente inferior que el conseguido por el modelo entrenado en Cityscapes, por lo que este modelo no aporta valor a la hora de resolver la tarea de segmentación de imágenes.



**Figura 5.7:** Máscaras de segmentación resultado de realizar inferencia del modelo Segformer MiT-B0 entrenado en GTA V sobre el dataset Mapillary.

Vemos que este modelo presenta especiales dificultades a la hora de reconocer ciertas categorías, como puede ser el cielo con presencia de nubes o los árboles. Esto es debido a que son elementos con falta de realismo en el videojuego, por lo que el modelo no es

capaz de asociar esos elementos del videojuego con los de la vida real.

En la tabla 5.4 exponemos los valores que toman las métricas a la hora de realizar inferencia sobre 500 imágenes de Mapillary. Como avanzábamos anteriormente, los resultados son peores que los del resto de modelos, de modo que no batimos los resultados del modelo entrenado en Cityscapes para ninguna de las categorías.

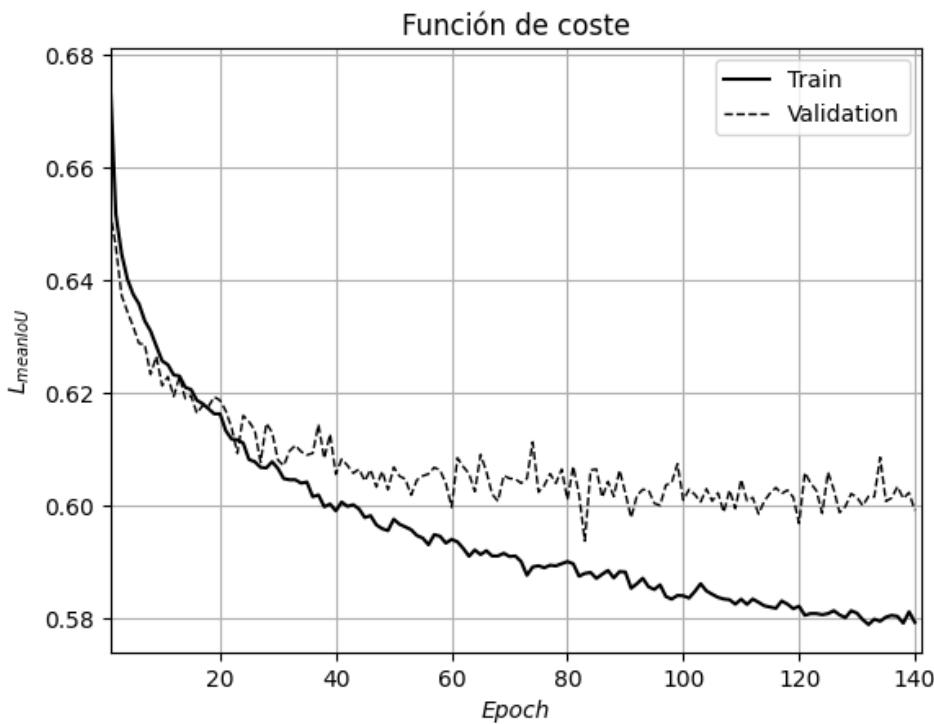
	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.74	0.94	0.78	0.85
vegetation	0.62	0.86	0.69	0.76
car	0.58	0.86	0.64	0.74
building	0.58	0.69	0.78	0.73
road	0.56	0.67	0.77	0.72
person	0.36	0.60	0.47	0.53
terrain	0.22	0.24	0.78	0.36
traffic light	0.21	0.76	0.23	0.35
pole	0.20	0.69	0.22	0.33
sidewalk	0.18	0.24	0.41	0.30
fence	0.17	0.34	0.25	0.29
truck	0.16	0.27	0.30	0.28
traffic sign	0.11	0.86	0.11	0.20
wall	0.07	0.13	0.12	0.13
rider	0.02	0.52	0.02	0.05
motorcycle	0.02	0.35	0.02	0.04
bicycle	0.01	0.36	0.01	0.03
train	0.00	0.00	0.24	0.01
bus	0.00	0.04	0.00	0.00
<b>Mean</b>	<b>0.53</b>	<b>0.67</b>	<b>0.71</b>	<b>0.69</b>

**Cuadro 5.4:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 entrenado al completo en GTA V sobre 500 imágenes del dataset Mapillary Vistas.*

## 5.5. Modelo entrenado al completo en GTA V y Fine Tuning del decoder en Cityscapes

Este modelo consiste en un reentrenado con imágenes de Cityscapes del modelo entrenado en GTA V (5.4) manteniendo el encoder congelado. La idea es que una vez el modelo haya extraído las características espaciales de GTA V pueda adaptarse a segmentar imágenes reales.

En este caso han sido utilizadas un total de 2975 imágenes de Cityscapes, empleando un 95 % de ellas para entrenamiento y un 5 % para validación. De nuevo, se ha entrenado con un *batch* de 5 imágenes y la función de coste derivada de la métrica *meanIoU* (sección 2.5.2). Como optimizador se ha vuelto a escoger el optimizador *Adams*. La función de pérdidas durante las 140 épocas de entrenamiento la mostramos en la figura 5.8.



**Figura 5.8:** Función de coste durante las 140 épocas de entrenamiento del decoder del modelo Segformer MiT-B0 preentrenado en GTA V con 2975 imágenes del dataset Cityscapes. Con línea continua se representa la función de coste en imágenes de entrenamiento y con línea discontinua en imágenes de validación.

En la figura 5.9 mostramos el resultado de las máscaras de segmentación predichas por este modelo. En este resultado se puede apreciar que el modelo ha vuelto a mejorar considerablemente en imágenes reales, a diferencia de lo que sucedía con el modelo

entrenado al completo sobre imágenes sintéticas. Esto es esperable, pues las imágenes reales poseen características mucho más complejas a nivel de detalle de los elementos que las imágenes sintéticas que el modelo debe aprender.



**Figura 5.9:** Máscaras de segmentación resultado de realizar inferencia del modelo sobre el dataset Mapillary.

Las diferentes métricas empleadas para evaluar el modelo sobre 500 imágenes del Mapillary las mostramos en la tabla 5.5.

Vemos que la *IoU* ha mejorado en todas las categorías con respecto al modelo entrenado únicamente en GTA V, salvo para la categoría *truck* donde ha empeorado. En términos medios se ha mejorado en las cuatro métricas estudiadas.

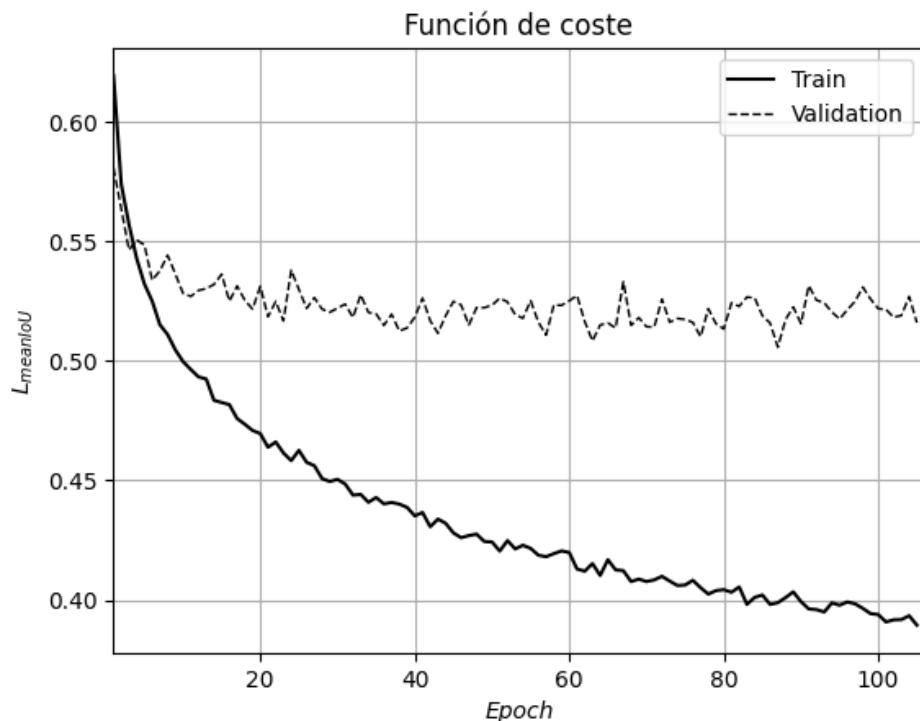
	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.85	0.98	0.87	0.92
vegetation	0.69	0.84	0.79	0.82
road	0.65	0.71	0.89	0.79
car	0.63	0.70	0.85	0.77
building	0.61	0.78	0.74	0.76
terrain	0.41	0.61	0.56	0.58
person	0.35	0.55	0.49	0.52
pole	0.28	0.53	0.38	0.44
traffic light	0.26	0.54	0.34	0.42
traffic sign	0.20	0.37	0.30	0.33
fence	0.19	0.27	0.39	0.32
sidewalk	0.19	0.36	0.28	0.31
rider	0.14	0.27	0.23	0.25
truck	0.11	0.23	0.19	0.21
motorcycle	0.11	0.25	0.17	0.20
bicycle	0.10	0.12	0.34	0.17
wall	0.09	0.20	0.15	0.17
bus	0.05	0.11	0.10	0.10
train	0.00	0.00	0.18	0.01
<b>Mean</b>	<b>0.62</b>	<b>0.75</b>	<b>0.78</b>	<b>0.77</b>

**Cuadro 5.5:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 sobre 500 imágenes del dataset Mapillary Vistas.*

## 5.6. Modelo entrenado al completo en GTA V y Fine Tuning completo en Cityscapes

Por último, vamos a entrenar de forma completa el modelo preentrenado en GTA V (5.4) con imágenes del Cityscapes. Si nos fijamos en la figura 3.1 donde mostramos la arquitectura del modelo, vemos que entrenando el encoder estamos actualizando los pesos de las capas de atención, por lo que esperamos una mejora en los resultados sobre las imágenes de Mapillary.

El entrenamiento se ha realizado siguiendo la misma técnica que en la sección 5.6 y la función de pérdidas la mostramos en la figura 5.10. En ella podemos ver que la función de coste en imágenes de validación difiere en mayor medida que en casos anteriores de la función de coste en imágenes de entrenamiento. Esto podría indicar la falta de regularización a la hora de entrenar el modelo.



**Figura 5.10:** Función de coste durante las 105 épocas de entrenamiento del modelo Segformer MiT-B0 preentrenado en GTA V con 2975 imágenes del dataset Cityscapes. Con línea continua se representa la función de coste en imágenes de entrenamiento y con línea discontinua en imágenes de validación.

En las máscaras de segmentación predichas por el modelo (figura 5.11) vemos que el modelo presenta una notable mejoría con respecto al modelo anterior, aunque sigue sin superar al modelo entrenado completamente con imágenes reales. La diferencia más significativa con respecto al modelo de la sección 5.6 la podemos ver en la última

imagen, donde ahora no se confunde la categoría *sky* con la categoría *road* (aunque la segmentación sigue sin ser perfecta).



**Figura 5.11:** Máscaras de segmentación resultado de realizar inferencia del modelo sobre el dataset Mapillary.

Para un análisis más detallado debemos fijarnos en la tabla de las métricas extraídas de la inferencia sobre las 500 imágenes de Mapillary (tabla 5.6). En ella podemos observar de nuevo una mejoría general con respecto al modelo anterior, aunque no llega a batir los resultados del modelo entrenado únicamente en Cityscapes.

	<b>IoU</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
sky	0.82	0.98	0.83	0.90
vegetation	0.74	0.83	0.87	0.85
car	0.68	0.81	0.81	0.81
road	0.67	0.83	0.77	0.80
building	0.54	0.63	0.79	0.70
terrain	0.39	0.60	0.53	0.57
person	0.38	0.56	0.53	0.55
traffic sign	0.29	0.51	0.41	0.45
traffic light	0.26	0.71	0.29	0.41
pole	0.26	0.61	0.31	0.41
fence	0.25	0.47	0.34	0.40
sidewalk	0.24	0.49	0.32	0.39
truck	0.19	0.38	0.27	0.31
bus	0.18	0.48	0.22	0.30
motorcycle	0.16	0.37	0.22	0.28
wall	0.16	0.36	0.22	0.27
rider	0.16	0.32	0.24	0.27
bicycle	0.14	0.18	0.40	0.25
train	0.01	0.01	0.07	0.02
<b>Mean</b>	0.65	0.81	0.76	0.79

**Cuadro 5.6:** *IoU, Precision y Recall de cada una de las categorías al realizar inferencia del modelo MiT-B0 sobre 500 imágenes del dataset Mapillary Vistas.*

## 5.7. Comparación de modelos

En esta sección comparamos los resultados de los distintos modelos analizados en este trabajo. En la tabla 5.7 mostramos en términos medios las métricas de cada uno de ellos.

Modelo	meanIoU	Precision	Recall	F1-score
City	0.75	0.82	0.89	0.85
City + Dec GTA	0.70	0.80	0.85	0.82
GTA + All City	0.65	0.81	0.76	0.79
GTA + Dec City	0.62	0.75	0.78	0.77
City + All GTA	0.58	0.73	0.75	0.74
GTA	0.53	0.67	0.71	0.69

**Cuadro 5.7:** *IoU, precisión, recall y F1-score promedios para cada uno de los modelos al realizar inferencia sobre 500 imágenes del dataset Mapillary.*

Vemos que el modelo entrenado únicamente con el dataset Cityscapes obtiene unos mejores resultados para el valor medio de todas las métricas analizadas. En las máscaras de segmentación (figura 5.13) también podemos reconocer a simple vista que el modelo entrenado solo en Cityscapes es capaz de segmentar mejor los diferentes elementos de las imágenes.

El modelo que arroja peores resultados en términos medios es el entrenado de forma completa con imágenes de GTA V. Las imágenes sintéticas de GTA V son mucho más sencillas en cuanto a nivel de detalle de los elementos que las imágenes reales, por lo que al evaluar el modelo sobre Mapillary no es capaz de alcanzar el rendimiento del entrenamiento con imágenes reales. Esto, tal y como es de esperar, indica que la inclusión de imágenes reales es vital. De hecho, la idea original de este trabajo es compensar la menor complejidad en cuanto a detalle de las imágenes sintéticas con su mayor diversidad de distribución de elementos (figura 4.4).

Observamos que los resultados de los modelos preentrenados en GTA V mejoran cuando se reentrena con imágenes reales y, además, es mejor el modelo donde el reentrenado es completo.

A nivel categoría, mostramos un gráfico comparativo de las cuatro métricas en la figura 5.12. Si nos fijamos en el *IoU* vemos que el modelo entrenado al completo en Cityscapes es batido *person* y *building* por el modelo en el que se ha realizado un fine tuning del decoder en GTA V. Estas categorías son muy importantes en tareas como la conducción autónoma, por lo que sí que estaríamos encontrando utilidad al uso de imágenes sintéticas. Además, en otras categorías como *sky*, *car*, *terrain*, *pole* y *wall* el

rendimiento de los modelos entrenados a través de imágenes sintéticas es muy similar al rendimiento del modelo entrenado al completo en imágenes reales.

En cuanto a la precisión, los modelos entrenados en GTA V toman la delantera en varias de las categorías. Las más destacadas son *building*, *person*, *traffic sign*, *traffic light* y *bicycle*. Esto quiere decir que estos modelos tienen una baja tasa de errores y realizan las predicciones correctamente en mayor medida para estas categorías.

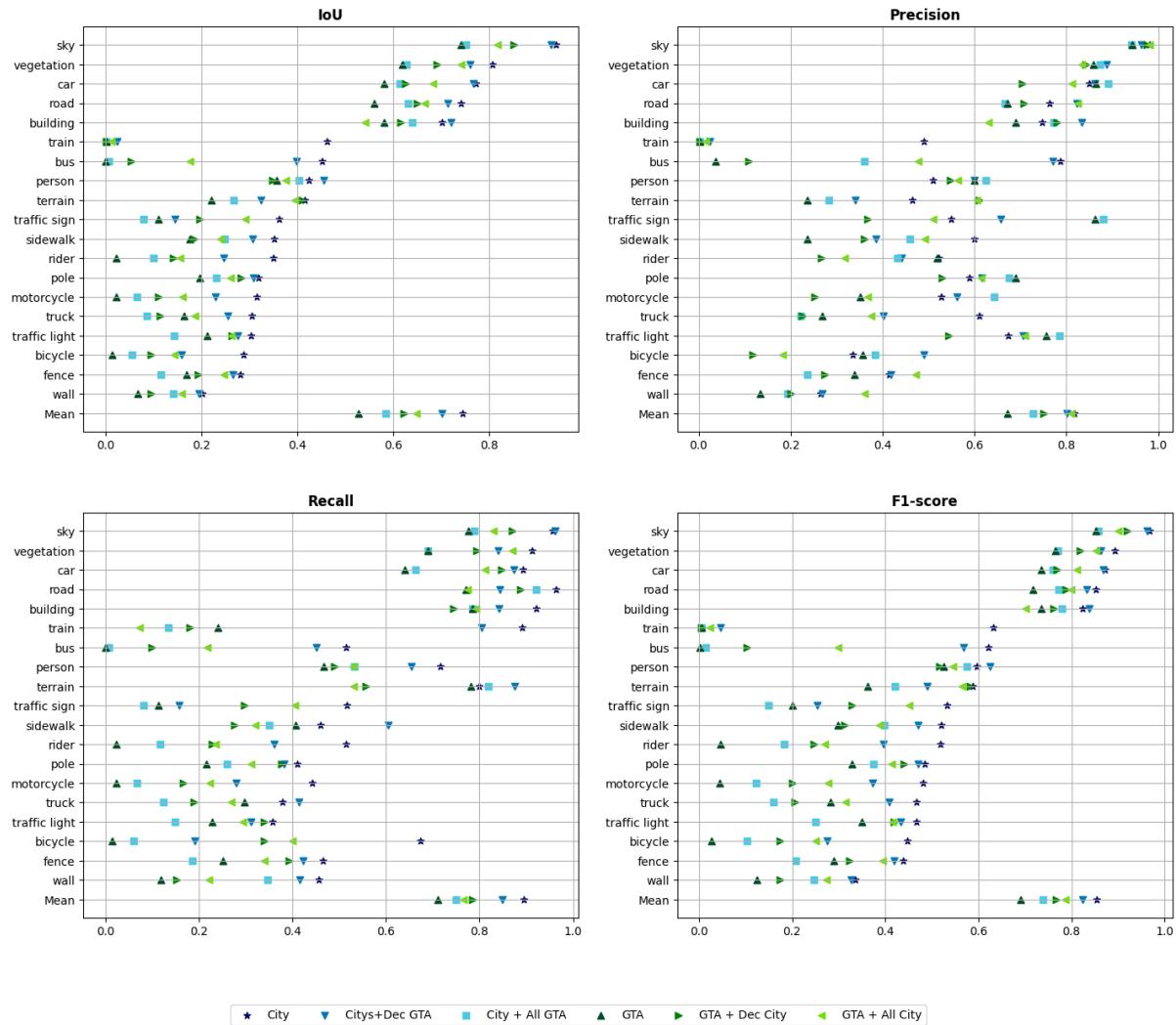
Sin embargo, tal y como explicamos en la sección 2.4 es importante considerar más métricas a la hora de evaluar el rendimiento de un modelo de clasificación, pues el análisis de una única métrica puede dar lugar a confusiones. Por ejemplo, el modelo preentrenado en Cityscapes y reentrenado al completo en GTA V tiene el valor de precisión más elevado para la categoría *traffic sign*, pero también el valor más bajo de *recall* para esta misma categoría, lo que indica que este modelo es muy conservador a la hora de hacer predicciones, evitando clasificar como positivos casos que son negativos.

En lo que respecta al *recall*, el modelo entrenado únicamente en Cityscapes es de nuevo batido por el resto en varias de las categorías como *terrain*, *sidewalk* y *truck*.

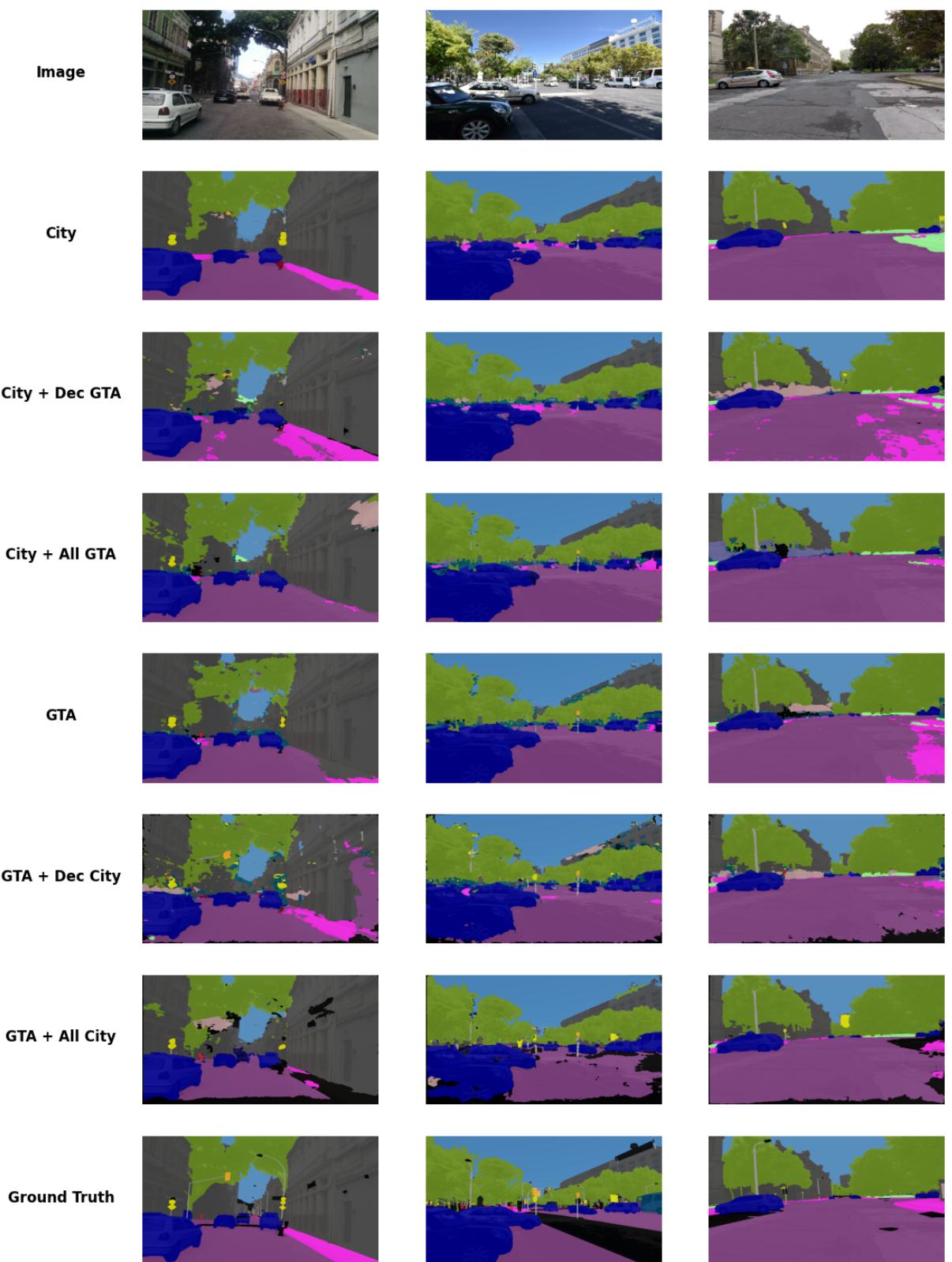
El *F1-score* combina la precisión y el recall dando la misma importancia a los falsos negativos que a los falsos positivos y los resultados para esta métrica son muy similares a los de la *IoU*, obteniendo el modelo entrenado únicamente en Cityscapes el mejor resultado en todas las categorías excepto en *building* y *person* donde el modelo con fine tuning del decodeer en GTA V es mejor.

Otro aspecto importante a comentar es que en la categoría *train* (y *bus* en menor medida) se obtiene un valor muy bajo para las métricas de los modelos preentrenados en GTA V. Esto es debido a que en las imágenes sintéticas de entrenamiento esta categoría está muy poco representada. Además, el variedad de trenes en el videojuego es muy escasa, lo que complica todavía más su identificación en la vida real.

En la imagen 5.13 mostramos una comparativa de las máscaras de segmentación de los distintos modelos para una misma imagen de Mapillary. En esta figura únicamente se presenta el resultado para tres imágenes. Una batería de resultados para más imágenes se puede encontrar en el apéndice.



**Figura 5.12:** Resultados obtenidos las diferentes versiones del modelo Segformer MiT-B0. Se muestra de forma visual los resultados de las tablas de los apartados anteriores.



**Figura 5.13:** Comparación de las máscaras de segmentación predichas por cada uno de los modelos estudiados en el trabajo.



# Capítulo 6

## Conclusiones

El objetivo de este trabajo de fin de máster era probar si se podía conseguir mejorar los resultados de un modelo de segmentación de imágenes mediante el uso de imágenes sintéticas en el proceso de entrenamiento. De esta forma, se ayudaría a solventar uno de los principales problemas a la hora de entrenar este tipo de modelos, que es la dificultad de conseguir y etiquetar imágenes.

Para llevar a cabo este objetivo se ha propuesto el entrenamiento del modelo basado en arquitectura transformer conocido bajo el nombre de Segformer en su versión más reducida en cuanto a número de parámetros. Este modelo se ha entrenado de diferentes formas haciendo uso tanto de imágenes reales como de imágenes sintéticas. La evaluación del rendimiento de cada modelo en cada una de las categorías del dataset Cityscapes se ha realizado a partir de las métricas *IoU*, *precision*, *recall* y *F1-score*. Además, como función de coste se ha empleado una función poco común derivada de la métrica *meanIoU*, que nos permite mejorar el rendimiento del modelo en categorías poco representadas de los datasets.

En términos medios, los modelos que hacen uso de imágenes sintéticas en el entrenamiento no consiguen batir al modelo entrenado al completo sobre imágenes reales. Sin embargo, para alguna de las categorías sí que se observa una mejoría al introducir las imágenes artificiales. Por ejemplo, el modelo preentrenado en Cityscapes y reentrenado su decoder en las imágenes de GTA V consigue mejorar el *IoU* en las categorías *person* y *building*, que son de importancia vital en aplicaciones como la conducción autónoma. Si nos fijamos en las otras métricas, también observamos una mejoría en el rendimiento del modelo para ciertas categorías.

Debido a nuestros límites computacionales, estos resultados han sido obtenidos utili-

## CAPÍTULO 6. CONCLUSIONES

---

zando alrededor de un 30 % de las imágenes sintéticas disponibles, por lo que pensamos que el modelo presenta bastante margen de mejora si se realiza el entrenamiento con el dataset GTA V al completo. Además, como posible extensión del conjunto de datos, se pueden emplear otras imágenes sintéticas extraídas de otras fuentes, lo que hace que las posibilidades de obtener imágenes de entrenamiento sean prácticamente ilimitadas.

De este modo, se ha probado que la inclusión de imágenes sintéticas en el entrenamiento de un modelo de segmentación puede ser beneficioso a la hora de detectar ciertas categorías, lo que abre una ventana a su inclusión para la resolución de diversas tareas, que van desde el control de cultivos hasta el coche autónomo.

# Bibliografía

- [1] Jay Alammar. “The Illustrated Transformer”. En: (2018). URL: <https://jalammar.github.io/illustrated-transformer/>.
- [2] F. van Beers et al. “Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation”. En: feb. de 2019. DOI: 10.5220/0007347504380445.
- [3] Holger Caesar, Jasper Uijlings y Vittorio Ferrari. *COCO-Stuff: Thing and Stuff Classes in Context*. 2018. arXiv: 1612.03716 [cs.CV].
- [4] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV].
- [5] Marius Cordts et al. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV].
- [6] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].
- [7] Yang He et al. “Segmentations-Leak: Membership Inference Attacks and Defenses in Semantic Image Segmentation”. En: nov. de 2020, págs. 519-535. ISBN: 978-3-030-58591-4. DOI: 10.1007/978-3-030-58592-1\_31.
- [8] Amirkhossein Kazemnejad. “Transformer Architecture: The Positional Encoding”. En: *kazemnejad.com* (2019). URL: [https://kazemnejad.com/blog/transformer\\_architecture\\_positional\\_encoding/](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/).
- [9] Jonathan Long, Evan Shelhamer y Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].
- [10] Liang-Chieh Chen George Papandreou Iasonas Kokkinos Kevin Murphy y Alan L. Yuille. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets Atrous Convolution and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV].
- [11] Stephan R. Richter et al. *Playing for Data: Ground Truth from Computer Games*. 2016. arXiv: 1608.02192 [cs.CV].
- [12] Olaf Ronneberger, Philipp Fischer y Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

## BIBLIOGRAFÍA

---

- [13] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. En: *International Journal of Computer Vision (IJCV)* 115.3 (2015), págs. 211-252. DOI: 10.1007/s11263-015-0816-y.
- [14] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].
- [15] Web. Arquitectura típica de cnn. [https://upload.wikimedia.org/wikipedia/commons/6/63/Typical\\_cnn.png](https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png).
- [16] Web. *Cityscapes Cities*. <https://www.cityscapes-dataset.com/wordpress/wp-content/uploads/2018/05/cityscapesCities-768x998.jpg>.
- [17] Web. *Cityscapes Examples*. <https://www.cityscapes-dataset.com/examples/fine-annotations>.
- [18] Web. *DETR pipeline*. [https://alcinos.github.io/detr\\_page/assets/overview.jpg](https://alcinos.github.io/detr_page/assets/overview.jpg).
- [19] Web. *Intersection Over Union*. <https://pylessons.com/media/Tutorials/YOLO-tutorials/YOLOv3-TF2-mAP/IoU.png>.
- [20] Web. *Multilayer Perceptron*. [https://miro.medium.com/proxy/1\\*eloYEyFrblGHVZhU345PJw.jpeg](https://miro.medium.com/proxy/1*eloYEyFrblGHVZhU345PJw.jpeg).
- [21] Web. *Neurona*. [https://insights.sei.cmu.edu/media/images/sestilli-deeplearning\\_artificialneuron3.original.png](https://insights.sei.cmu.edu/media/images/sestilli-deeplearning_artificialneuron3.original.png).
- [22] Web. *Segformer MiT-B0 preentrenado en Cityscapes*. <https://huggingface.co/nvidia/segformer-b0-finetuned-cityscapes-1024-1024>.
- [23] Web. *Segmentación de imagen*. <http://www.zemris.fer.hr/~sseguic/multiclod/images/causevic16semseg2.png>.
- [24] Enze Xie et al. *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*. 2021. arXiv: 2105.15203 [cs.CV].
- [25] Bolei Zhou et al. *Semantic Understanding of Scenes through the ADE20K Dataset*. 2018. arXiv: 1608.05442 [cs.CV].

# Apéndice

Máscaras de segmentación para imágenes diferentes a las de la figura 5.13.

