Visión por Computadora

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Pŕactico Final

Estimación de homografía con Deep Learning

Integrante	LU	Correo electrónico
Silvani, Damián Emiliano	658/06	dsilvani@gmail.com
Palladino, Julián Alberto	336/13	julianpalladino8@gmail.com

Utilización de regresión con Deep Learning para estimar homografías 2D entre pares de imágenes. Análisis e implementación del algoritmo original propuesto por el paper "Deep Learning Homography Estimation". Implementación y discusión de ideas alternativas.

Homografía **Deep Learning** **Visión por Computadora**

Índice

1. Introducción

La estimación de homografías a partir de imágenes es una tarea fundamental en el área de visión por computadora. Sus usos y aplicaciones incluyen calibración de cámaras, reconstrucción 3D, visión estéreo, entre muchas otras.

Particularmente, en la materia de Visión por Computadora, analizamos la estrategia "manual" de utilizar el algoritmo DLT junto con SIFT y RANSAC para detectar la homografía entre un par de imágenes. Esta estrategia comienza con la extracción de features de ambas imágenes, luego el cálculo de las correspondencias, el decarte de correspondencias ruidosas y finalmente el cálculo de la homografía. Este es el enfoque clásico, al que se denomina "manual", ya que requiere un trabajo más cercano a la imagen en sí - hilando fino en cada paso del algoritmo. También, y por este motivo, se lo puede denominar de "caja blanca", ya que sus resultados se pueden explicar o justificar prestando atención a cómo se implementó cada parte del algoritmo.

En este trabajo práctico se aborda el mismo desafío de encontrar homografías, pero con una estrategia alternativa, tomando provecho de Deep Learning. La idea consiste en implementar una red neuronal profunda, y que dicha red tome abundantes pares de imágenes junto con sus respectivas homografías para que encuentre el patrón que define el cálculo de la homografía, es decir, que "aprenda". Esta estrategia se la puede calificar de "caja negra", ya que es mucho más difícil (aunque no imposible) justificar o explicar las decisiones tomadas por la red.

2. Red y metodología de trabajo propuesta

El objetivo del presente trabajo es implementar la red propuesta en el paper *Deep Image Homography Estimation*[2].

El paper presenta una red convolucional profunda para estimar homografías entre pares de imágenes. La red convolucional consiste de 10 capas, y toma como entrada dos imágenes en escala de grises. La salida es una homografía de 8 grados de libertad, que puede ser utilizada para crear correspondencias entre los píxeles de la primer imagen a la segunda.

En cuanto a la implementación, fue realizada enteramente en **Python 3**, utilizando la librería **TensorFlow** y su API de alto nivel **Keras**. También, se aprovechó del uso de GPU para procesar los datos de manera simultánea y así entrenar y evaluar el modelo más rápidamente.

2.1. Modelo de regresión vs. de clasificación

En el paper, los autores implementan esta red de dos maneras distintas: Haciendo clasificación y haciendo regresión. El modelo de regresión es el más intuitivo y simple: La red produce 8 valores que indican las diferencias entre las posiciones de las 4 esquinas de la primer imagen sobre la segunda - permitiendo mapear los píxeles de las dos imágenes, y de esa manera generar una homografía. Por ejemplo, para la homografía identidad, la salida será el vector nulo. Por otro lado, el modelo clasificador devuelve una matriz de 8x21, que puede ser interpretada como distribuciones probabilísticas de la posición de cada esquina de la primer imagen, sobre la segunda. O sea, si la segunda imagen se divide en forma de grilla de 21x21, el output de la red clasificadora indica sobre esa grilla, las probabilidades de cada esquina de la primer imagen de ubicarse en cada celda. Este enfoque clasificador tiene ciertos usos muy específicos, sin embargo, nos pareció muy rebuscado y complejo. Sumado a que en el paper su performance da mucho peor que el modelo de regresión, elegimos no implementarlo. En lugar de hacerlo, preferimos concentrarnos en el modelo de regresión, intentando darle algún giro y mejorar aún más la performance obtenida en la publicación original.

2.2. Generación del dataset

El dataset utilizado en el paper fue MS-COCO 2014 [3]. En nuestro trabajo se utilizó MS-COCO 2017, el cual tiene más imágenes de entrenamiento.

La generación se realizó de la misma manera propuesta por el paper. La idea principal es aprovechar al máximo las imágenes del dataset, ya que las redes neuronales profundas requieren una enorme cantidad de datos para obtener buenos resultados. Para lograrlo, aplicamos transformaciones proyectivas aleatorias sobre parches aleatorios de las imágenes del dataset original (en total 10 parches por imagen).

Estas transformaciones proyectivas se obtienen a partir de tomar un parche de la imagen original, modificar la posición de las esquinas según la variable de distorsión ρ, y luego tomar un parche más chico sobre esa imagen.

Ejemplo:

Figura 1: **Primer paso.** Se toma un parche aleatorio sobre una imagen del dataset

Figura 2: **Segundo paso.** Se distorsionan las esquinas de manera aleatoria, correspondiendo con la variable ρ

Figura 3: **Tercer paso.** Se aplica la transformación correspondiente a la distorsión aplicada

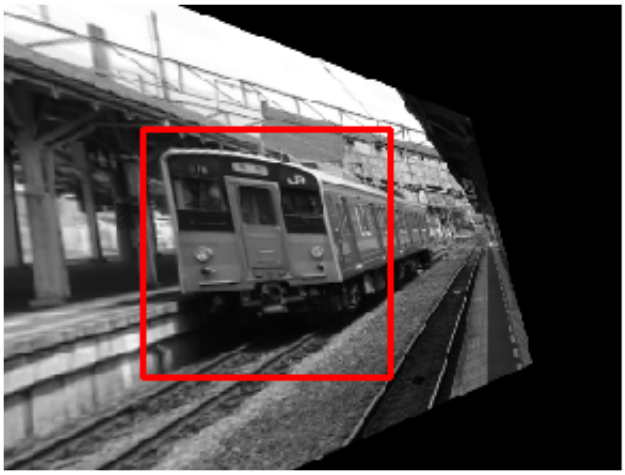

Figura 4: **Cuarto paso.** Se toma el parche original sobre la imagen distorsionada

Figura 5: **Listo.** Se obtuvieron los dos parches, original y distorsionado aleatoriamente. Además, tenemos la homografía exacta que transforma el uno en el otro.

Total de imágenes: De un dataset originalmente de 40670 imágenes de entrenamiento, realizando 10 parches por imagen, obtenemos un **dataset de 406.700 parches** de entrenamiento, lo cual alcanza para entrenar nuestro modelo.

2.3. Métricas utilizadas

La métrica utilizada en este trabajo es Mean Average Corner Error (**MACE**). Como la salida del modelo es la distancia entre las esquinas del parche distorsionado de las del original, MACE calcula la distancia L2 entre las posiciones de las esquinas según la verdad de campo y las estimadas. El error se promedia sobre las cuatro esquinas, y esto se promedia sobre todo el conjunto de imágenes. De esa forma podemos medir qué tan cercana a la realidad fue nuestra predicción.

2.4. Evaluación del modelo

Para poder detectar si nuestro modelo sobreajusta, hicimos las evaluaciones finales sobre un conjunto completamente distinto de datos al de entrenamiento: La sección de validación de MS-COCO 2017.

3. Modelo propuesto

El modelo elegido, entonces, fue el de regresión. Como entrada, dos imágenes en escala de grises, correspondientes a un parche de una imagen del dataset y su distorsión en base a ρ. Como salida, 8 valores reales, representando la diferencia en dos dimensiones de cada una de las 4 esquinas.

3.1. Arquitectura

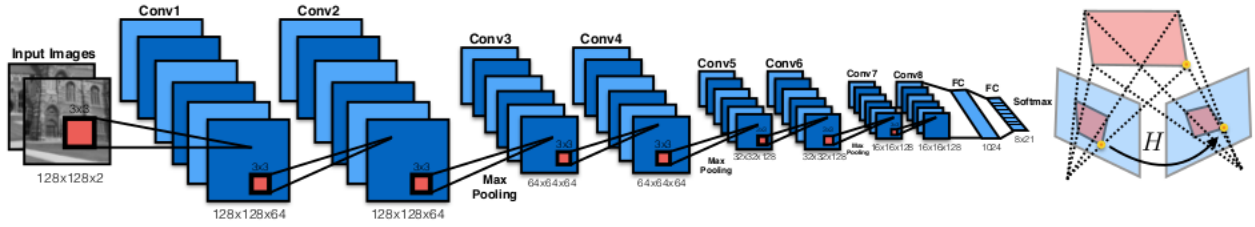

Figura 6: Arquitectura de regresión utilizada

La arquitectura de la red consta de 10 capas[1]: Ocho capas convolucionales y dos capas densas (`fully connected, FC`).

Las convolucionales operan con convoluciones de la misma manera que vimos en la materia. Su kernel era de 3x3, con `stride` de 2x2 y `pool size` de 2x2. La intención de las capas convolucionales es extraer atributos de la imagen de entrada.

Las capas densas son perceptrones multi-capa. Cada neurona de la capa anterior está conectada con una de la capa densa (por eso el nombre "fully connected"). La idea es que la salida de las capas de convolución representa atributos de alto nivel de la imagen de entrada. Luego, las capas densas toman estos atributos para "aprender".

La función de activación final es "softmax", la cual devuelve los 8 valores reales de salida que corresponden a las diferencias en píxeles de cada esquina del parche.

También se utilizó:

- Un Learning Rate (LR) de 0.005, con un Learning Scheduler que va reduciendo el LR de a poco para encontrar el mínimo global y no estancarse en un local (en caso de LR muy chico), y tampoco ir dando grandes saltos (en caso de LR muy grande).

[1]Sin contar entrada, `batch normalization`, `dropout`, etc.

- Además, se utilizaron capas de Batch Normalization entre cada par de capas convolucionales (para normalizar los valores entre capa y capa), y capas de Dropout antes de las densas (para evitar sobreajuste).
- Se utilizó el mismo batch size que en el paper (64).

3.2. Experimentación

Se realizaron ciertos experimentos con la intención de probar el modelo original del paper, y también distintas variaciones para estudiarlo desde otras perspectivas.

3.2.1. Evaluando el modelo original

Una cuestión importante al momento de crear este tipo de modelos es tener un **marco de referencia** o `baselines`, o sea, otras alternativas con la cual compararlo en cuanto a performance.

En el caso de este trabajo, para evaluar el modelo de Deep Learning, comparamos sus homografías resultantes contra:

- La homografía "identidad". O sea, la homografía que no aplica ninguna transformación sobre la imagen. No es la peor homografía posible, pero sí es buena referencia de cómo sería una homografía muy mal calculada.
- La homografía calculada con ORB + RANSAC + "findHomography()" de OpenCV. Es buena referencia de una homografía calculada con métodos clásicos manuales, utilizadas en la práctica.

Los resultados fueron:

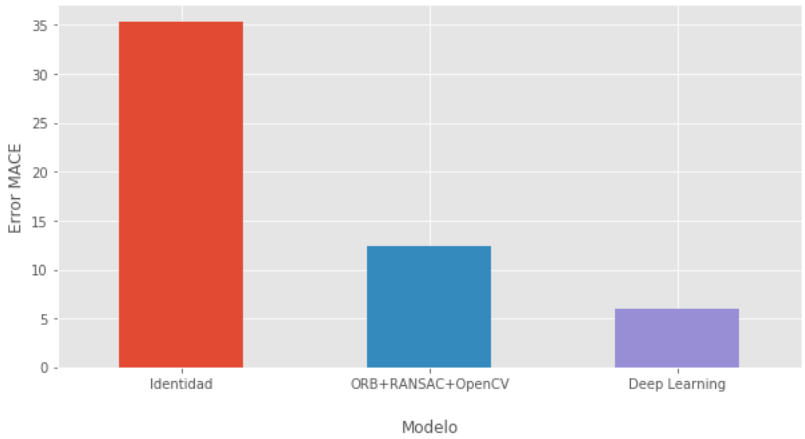

Figura 7: Evaluación de nuestro modelo vs. otras alternativas

Inicialmente se puede apreciar la gran diferencia entre la homografía "identidad" y la calculada con los métodos manuales. Esto muestra que los métodos manuales que utilizamos funcionan muy bien para encontrar correspondencias. Un ejemplo:

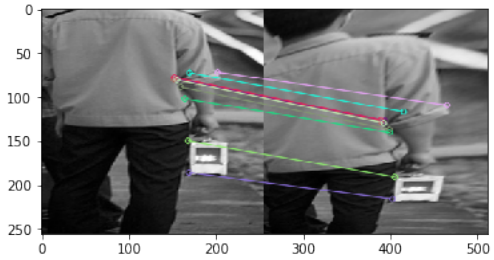

Figura 8: Ejemplo de cálculo de correspondencias con ORB.

Pero, siguiendo por el gráfico, vemos que este método manual es superado por nuestro modelo, que tiene la mitad del error del método manual.

De todas maneras, vale la pena mencionar la gran ventaja del método manual por sobre el de Deep Learning: Nuestro modelo tardó 6 horas en armar el dataset y entrenar el modelo. En cambio, el método manual lo hace de manera eficiente sin preparación ni uso de hardware especial, ni memoria para el modelo de la red, sólamente las dos imágenes para evaluar.

3.2.2. Variación de ρ

Como se mencionó anteriormente, la variable ρ (rho) es un valor fijado desde la construcción del dataset, pasando por el entrenamiento y finalmente sobre la evaluación. Determina la dificultad del problema a resolver: Es el valor máximo que puede tomar la distorsión de cada esquina del parche tomado. De esta manera, un valor de ρ chico implica pequeñas distorsiones sobre la imagen, ya que las esquinas del parche distorsionado estarán a menor distancia de las originales. En cambio, un valor grande, presentará imágenes más deformadas, lo cuál será más difícil de resolver para nuestro modelo.

Siendo que la creación del dataset, el entrenamiento y la evaluación toman mucho tiempo en ejecutarse cada una, no hicimos un test exhaustivo sobre distintos valores de ρ. Sin embargo, con sólamente dos valores distintos del original (uno mayor y otro menor), se puede ver con facilidad su impacto sobre el programa:

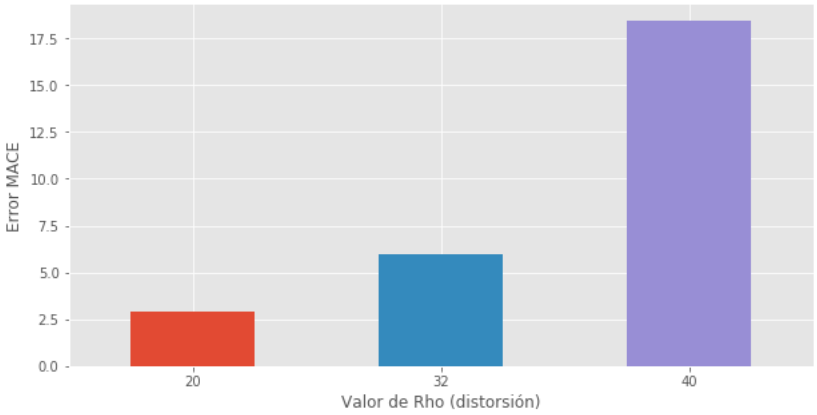

Figura 9: Experimento variando el valor de distorsión ρ, evaluando score de MACE. El valor original de ρ en el paper era 32.

Se ve gran diferencia entre el ρ original y las dos variaciones, aunque sorprende que parece crecer exponencialmente, ya que de 20 a 32 crece sólamente 3 puntos de MACE, mientras que de 32 a 40 aumenta aproximadamente 12.

3.2.3. Visualización de resultados reales

Presentamos algunos resultados reales obtenidos por el modelo, variando el valor de perturbación ρ. Vale la pena apreciar la graduación de la deformación de las imágenes, y la reducción de performance a medida que se aumentan dichas deformaciones.

Figura 10: Primer experimento: Imágenes original y perturbada, con factor $\rho = 20$.

Figura 11: Primer experimento: Imagen de predicción de la red. MACE = 1.449342

Figura 12: Segundo experimento: Imágenes original y perturbada, con factor $\rho = 32$.

Figura 13: Segundo experimento: Imagen de predicción de la red. MACE = 7.4091.

Figura 14: Tercer experimento: Imágenes original y perturbada, con factor $\rho = 40$.

Figura 15: Tercer experimento: Imagen de predicción de la red. MACE = 11.470876.

3.2.4. Sin dropout ni Batch Normalization

El paper propone utilizar tanto Dropout como Batch Normalization entre las capas de la red. Quisimos probar qué tan importantes son, desactivándolas y testeando su performance.

Dropout es una capa que especialmente combate el sobreajuste. La idea es evitar que recaiga mucho del comportamiento de la red en unas pocas neuronas. Dropout ayuda a evitar esto, mediante la desactivación de cierta cantidad de neuronas, elegidas aleatoriamente. De esta forma, el modelo evita que dependa todo el comportamiento sobre una sola neurona, ya que, de ser desactivada en la siguiente época, el modelo tendrá mala performance.

Batch normalization apunta a normalizar las capas y escalar las activaciones. Por ejemplo, si una feature tiene rango de 0 a 1, y otra de 0 a 10000, es importante que el modelo las varíe acorde a su respectivo rango. Entonces, para crear más estabilidad en la red, la capa de batch normalization normaliza el output de la activación previa, restándole el promedio del batch y dividiendo por el desvío estandard del batch.

Con esto en mente, evaluamos nuestro modelo con y sin estas capas:

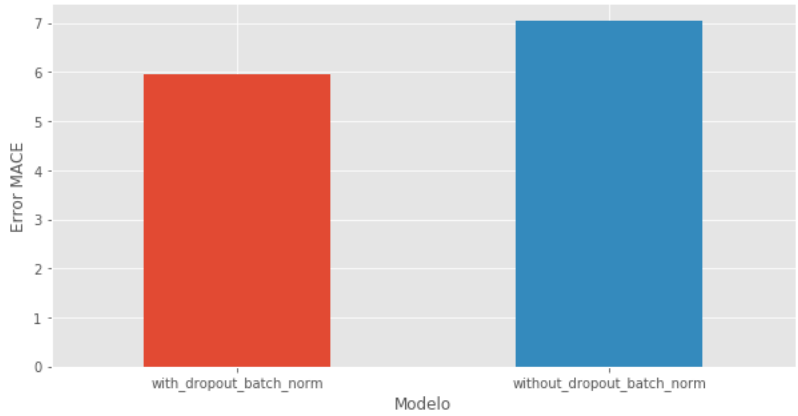

Figura 16: Experimento removiendo las capas de dropout y de batch normalization

Concluimos que los autores del paper hicieron bien en agregarlas, y que su funcionalidad es requerida para una mejor performance.

4. Alternativa: Transfer learning

Para darle una vuelta de tuerca extra al trabajo, nos embarcamos en experimentar con temas que los autores mismos del paper no propusieron.

4.1. Transfer learning

Transfer learning es una técnica utilizada cada vez más en el mundo de redes neuronales, especialmente cuando no se tienen suficiente cantidad de datos de entrenamiento como para hacer funcionar el aprendizaje de la red. La idea consiste en utilizar una red ya entrenada, con pesos en sus ejes, y utilizar esta información para prescindir de entrenar con grandes cantidades de datos.

Esta técnica es utilizada mucho, por ejemplo, en clasificación de objetos. Por más que los objetos en el primer entrenamiento no sean los mismos que los del segundo, las primeras capas de la red aprenden a distinguir bordes y esquinas (como si utilizara los métodos manuales de Canny y Harris), mientras que las capas superiores aprenden a distinguir rasgos más finos de los objetos del dataset.

De esta manera, la idea es aprovechar lo aprendido por la red en las primeras capas (detección de bordes, features y esquinas), descartando lo que aprendió en las últimas (detección más específica de los objetos del dataset). Para lograrlo, se entrena el dataset con sólo las primeras capas “congeladas” (o sea, no entrenables). Así, durante el training, la red modifica sus pesos sobre las últimas capas, más específicas del dataset en cuestión, mientras que aprovecha los pesos obtenidos anterioremente en las primeras.

En teoría, el efecto del transfer learning se puede visualizar de la siguiente manera:

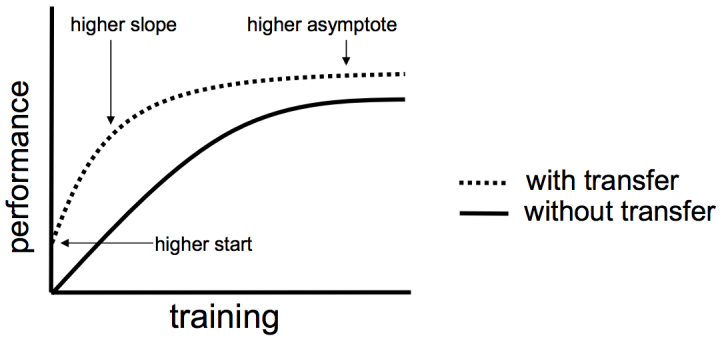

Figura 17: Ventaja en performance que otorga el transfer learning en teoría.

4.2. Intento de aplicarlo al trabajo

Al intentar aplicar esta idea al trabajo, nos vimos enfrentados con una barrera bastante previsible: No existen redes ya entrenadas para detectar homografías. La única alternativa entonces, fue hacer el experimento de transfer learning con una red entrenada para clasificar objetos, y utilizarla para detectar homografías.

Otro inconveniente es que las redes ya entrenadas tienen como input una imagen (para clasificar sobre ella), mientras que nosotros tenemos dos imágenes (la original y la distorsionada).

Nuestra arquitectura, entonces, consistió en una unión de dos redes pre-entrenadas, sumada a capas densas que reforzarían el aprendizaje de homografías. Las redes pre-entrenadas son de la arquitectura MobileNetV2 [4], que ofrece buenos resultados, mientras que tiene pocos requerimientos de cómputo para entrenar. Los elementos de pre-entrenamiento fueron las imágenes de Imagenet [1], usadas para clasificar objetos.

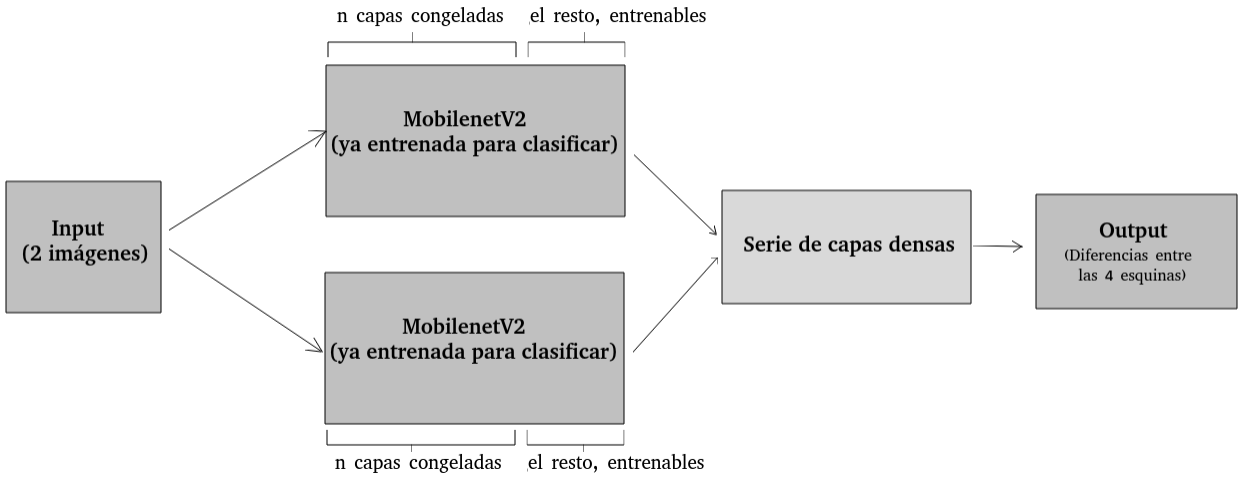

Figura 18: Modelo propuesto por nosotros de transfer learning.

La serie de capas densas la fuimos variando, desde una única capa de 8 con activación Softmax, hasta varias capas de 1024 neuronas cada una.

También fuimos variando el Learning Rate, el Learning Scheduler y la regularización.

Sin embargo, los resultados siempre fueron muy malos. Lo mejor que pudimos obtener fue con la configuración:

- Las primeras 130 capas de las redes MobileNetV2 congeladas (al entrenar).
- La serie de capas densas consistiendo en una de 1024, y luego una final de 8.

- Regularización L2 de 0.01.

Los resultados fueron:

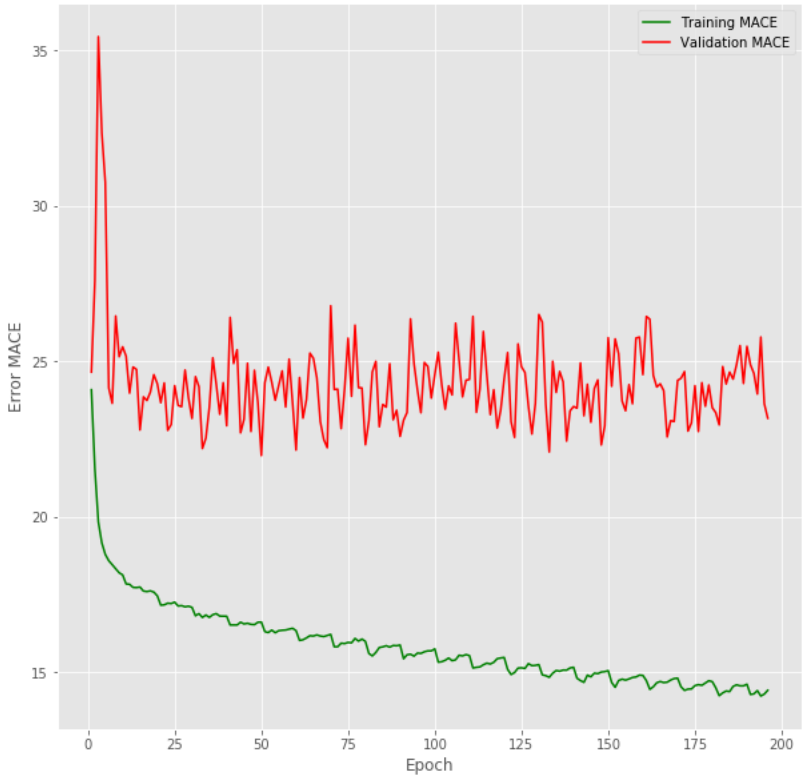

Figura 19: Entrenamiento del modelo de transfer learning.

Se puede apreciar la gran variabilidad del score de validación, pero más todavía se puede ver que el modelo está sobreajustando, y no está aprendiendo realmente. Esto se hace notar porque el score de entrenamiento baja constantemente y el de validación está estancado.

Charlando con los docentes, nos explicaron lo que sospechábamos: Si la red pre-entrenada fue diseñada y entrenada para algo completamente distinto, no hay manera de hacerlo andar.

5. Conclusiones

Este trabajo no sólo fue implementar un algoritmo propuesto por un paper, sino que también involucró explorar los límites teóricos y prácticos del modelo propuesto y los temas en cuestión; con la finalidad de aprender de manera teórica y práctica sobre Deep Learning y homografías.

Se describió el modelo original de manera teórica y detallada, y se presentaron distintos experimentos con el propósito de iluminar desde distintos ángulos su comportamiento interno de manera práctica (variación de ρ y arquitectura de modelo). También se evaluó su performance frente al caso manual de ORB+RANSAC, demostrando ser una alternativa eficiente. Por último, se presentó una idea nueva con su respectiva implementación e investigación: La utilización de transfer learning.

Por el lado personal, la implementación en sí consistió en un gran desafío en muchos aspectos:

- En el aspecto teórico, no estábamos seguros de cómo funciona Deep Learning aplicado al aprendizaje de homografías. Fue necesario confiar en el paper en varios aspectos, por ejemplo, que la arquitectura era la correcta.

- En el aspecto práctico, fue desafiante armar el modelo de red neuronal, aunque también lo fue armar la generación del dataset, en la cual tuvimos que recurrir a los conocimientos aprendidos en la materia de Visión por Computadora.
- En el aspecto de ingeniería de software, fue necesario armar determinadas estructuras de software para mantener un código organizado, claro y lo más simple posible. Ya de por sí las redes neuronales son bastante *cajas negras*, entonces tener un código complicado o difícil de leer exacerbaría esta situación al momento de *debuggear* o de entender ciertos resultados de experimentos.
- En el aspecto investigativo, tuvimos que mantener orden y claridad al momento de probar distintas alternativas, para no confundirse entre los distintos experimentos. Aquí también se pone en juego el gran costo temporal de cada experimento, ya que, al esperar 7 horas a que termine de correr una prueba, es fácil perder el hilo de lo que se estaba probando y con qué intenciones.

6. Trabajo futuro

Habían muchísimas maneras de abarcar la exploración que proponía este trabajo. Nosotros elegimos algunas y, por cuestiones de tiempo, no llegamos a ver todas. Algunas ideas que nos quedaron "en el tintero" fueron:

- Lograr mejorar, al menos un poco, la implementación de transfer learning. Esto podría haber sido de distintas maneras. Por ejemplo, reducir el ρ para facilitar el problema, o bien pre-entrenar nosotros mismos una red con un dataset de homografías y aplicar transfer learning sobre otro dataset. También evaluar el uso de capas convolucionales en vez de densas.
- Otro factor que nos hubiera gustado estudiar es el de operar con imágenes a color, ya que el paper propone tratar con sólo imágenes en escala de grises. Nosotros creemos que esta decisión fue para simplificar el problema y el modelo, pero quizás logrando que ande con color se pueden aprovechar más datos de la imagen, que la escala de grises no nos da.
- Evaluar performance en tiempo. Como no tenemos tecnología de punta (ni mucho menos) para realizar estos experimentos, no nos pareció interesante un análisis en cuestiones de tiempo utilizando nuestras computadoras. Quizás sería interesante analizar performance en tiempo, en caso de tener hardware más actualizado.
- Evaluar sobre un dataset de homografías real, con pares de imágenes diferentes. El dataset que utilizan en el paper y en nuestro trabajo es sintético, dado que los pares son fabricados aplicando una transformación artificial. Sería útil hacer una comparación de resultados, o reentrenar el modelo con este dataset, dado que se asemejaría más al uso en la práctica de la técnica analizada.

Referencias

[1] J. Deng y col. «ImageNet: A Large-Scale Hierarchical Image Database». En: *CVPR09*. 2009.

[2] Daniel DeTone, Tomasz Malisiewicz y Andrew Rabinovich. *Deep Image Homography Estimation*. DOI: `https://arxiv.org/abs/1606.03798`.

[3] Tsung-Yi Lin y col. «Microsoft COCO: Common Objects in Context». En: *ECCV*. 2014.

[4] Mark Sandler, Andrew Howard y Menglong Zhu. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. DOI: `https://arxiv.org/abs/1801.04381`.