DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Final

Deep Image Homography Estimation

18/04/19 Visión por Computadora

Integrante	LU	Correo electrónico
Sturmer, Andreas	028/13	remruts@gmail.com
Lamela, Emanuel	021/13	emanuel93_13@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción

En el marco de la materia Visión por Computadora, nos propusimos resolver el problema de obtener una Homografía a partir de una imagen y una perturbación de la misma implementando la Red Neuronal *HomographyNet* descrita en el paper **Deep Image Homography Estimation**, de DeTone et al. [1] En dicho paper se describen 2 versiones de *HomographyNet*, una basada en Regresión y la restante basada en Clasificación. En este caso, implementamos la primera puesto que otorga mejores resultados.

El cálculo de homografías es esencial al campo de Visión por Computadora y se utiliza tanto en el algoritmo SLAM, como en sistemas de realidad aumentada y calibración de cámaras. Es por eso que encontrar una forma efectiva y rápida de calcular estas tranformaciones es tan importante y el uso de redes neuronales para solucionar este problema resulta de utilidad.

En este trabajo, comenzaremos por explicar la metodología del paper, haciendo hincapié en la arquitectura de la Red Neuronal implementada y la generación del dataset. Luego, detallaremos las leves diferencias entre la solución desarrollada por nosotros y la que describe el paper. A continuación, expondremos los resultados obtenidos. Y finalmente, discutiremos por qué creemos que los nuestros fueron mejores que lo reportados en DeTone et al.

2. Desarrollo

Para la implementación de este trabajo, se utilizó el lenguaje de programación *Python*, sumado a la conocidas librerías de Machine Learning *Tensorflow* y *Keras*.

2.1. Arquitectura de HomographyNet

La Red Neuronal *HomographyNet*, en su versión de Regresión, es una red neuronal convolucional similar a *VGG Net*. Toma como entrada una imagen en grayscale de dos canales de 128x128x2, es decir la imagen original y la perturbada combinadas.

Consta de 8 capas convolucionales. Una capa de max-pooling (de 2x2 con *stride* de 2) se encuentra intercalada luego de cada dos de las convolucionales previamente mencionadas.

Las primeras 4 capas convolucionales tienen 64 filtros, mientras que las últimas tienen 128. Luego de estas capas, se encuentran dos capas totalmente conectadas: la primera tiene 1024 unidades, mientras que la última tiene 8 unidades, correspondiendo a los valores reales de salida.

Se aplica *Dropout* con una probabilidad de 0.5 antes y después de la primer capa totalmente conectada. Además de esto, se utiliza *Batch Normalization* luego de cada capa convolucional.

La función de activación de las capas es *ReLu* y la función de pérdida es la distancia euclideana. Para el entrenamiento se utiliza *SGD* con momento de 0.9 con un learning rate base de 0.005, decrementando este último en un factor de 10 cada 30000 iteraciones.

Arquitectura de la red en Keras

```
Conv2D(input_shape=(128, 128, 2), data_format="channels_last", filters=64, kernel_size=3, padding
    ="same", activation="relu"),
BatchNormalization(),

Conv2D(filters=64, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

MaxPool2D(pool_size=2),

Conv2D(filters=64, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

Conv2D(filters=64, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

MaxPool2D(pool_size=2, strides=2, padding="same"),

Conv2D(filters=128, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

Conv2D(filters=128, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

MaxPool2D(pool_size=2, strides=2, padding="same"),

Conv2D(filters=128, kernel_size=3, padding="same", activation="relu"),
BatchNormalization(),

Conv2D(filters=128, kernel_size=3, padding="same", activation="relu"),
```

```
BatchNormalization(),

Flatten(),
Dropout(0.5),
Dense(1024, activation="relu"),

Dropout(0.5),
Dense(8)
```

2.2. La parametrización de 4 puntos de la Homografía

Tradicionalmente, las Homografías se parametrizan como una matriz de 3x3 que representa la rotación, traslación y escala de la transformación.

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \tag{1}$$

Figura 1: Matriz de Homografía

Sin embargo, si expandimos esto a un vector de 9 parámetros, se mezclan los términos de rotación y traslación. La submatriz $[H_{11}, H_{12}; H_{21}, H_{22}]$ representa la parte rotacional de la transformación y $[H_{13}, H_{23}]$ la parte translacional. Balancear esto en un problema de optimización, como es el entrenamiento de una Red Neuronal, es difícil.

Para ello, los autores del paper decidieron utilizar la Parametrización de 4 puntos, la cuál se basa en guardar los offsets entre las 4 esquinas del par de imágenes.

$$H_{4point} = \begin{bmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \\ \Delta u_3 & \Delta v_3 \\ \Delta u_4 & \Delta v_4 \end{bmatrix} \tag{2}$$

Figura 2: Parametrización de 4 Puntos de la Homografía

Donde $\Delta u_i = u'_i - u_i$ y $\Delta v_i = v'_i - v_i$, siendo (u_i, v_i) la esquina i de la imágen original y (u'_i, v'_i) la esquina i de la imágen perturbada, con $i \in \{1...4\}$. Por ende, la Red Neuronal, en la fase de entrenamiento, toma como label el arreglo de 8 posiciones correspondiente, con cada casillero representando una posición de esta parametrización. A su vez, la predicción de la red arroja un arreglo con las mismas características que luego es utilizado para calcular la Homografía. Dicho calculo se puede llevar a cabo utilizando un algoritmo de calculo de Homografías, por ejemplo DLT.

2.3. Armado del Dataset

Los autores del paper utilizan un dataset propio generado a partir del set de entrenamiento MS-COCO. Las imágenes son escaladas a 320x240 y transformadas a grayscale. Luego, se generan 500000 datos de entrenamiento a partir de estas con el método descrito a continuación.

Para generar un dato de entrenamiento, los autores eligen un recorte de tamaño 128x128 a partir de una imagen más grande en una posición p elegida al azar, evitando los bordes para prevenir defectos. Llaman a esta imagen I_p. Luego, las cuatro esquinas son perturbadas al azar en el rango [-32, 32]. Las cuatro correspondencias definen una homografía H_{AB}. Aplican la inversa de esta homografía a la imagen grande para producir una segunda imagen grande I'. Se realiza otro recorte I'_p en la misma posición p.

I_p e I'_p se combinan en una imagen de dos canales que será pasada como input a la red. La parametrización de 4 puntos de H_{AB} se usa como label para esta imagen en la fase de entrenamiento.

2.4. Diferencias de Implementación con el Paper

Nuestra implementación posee ciertas diferencias con la propuesta por el paper por cuestiones particulares que serán detalladas en cada una de las mismas.

La más significativa es utilizar un rango de perturbación de 16 pixeles en vez de 32 en la generación del dataset. Esto se debe a que, con el rango de 32, estabamos obteniendo imágenes defectuosas.

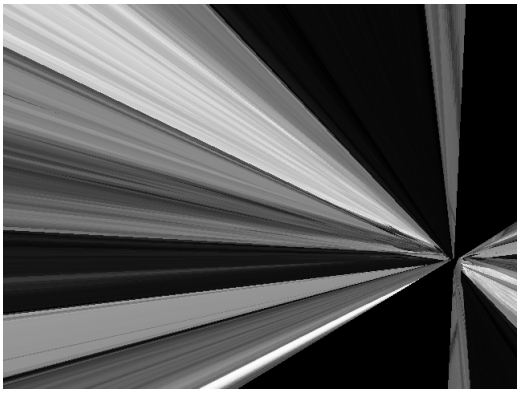

Figura 3: Perturbación Defectuosa

A modo prueba y error, encontramos que el rango de 16 era el de mayor amplitud que logramos utilizar sin que haya ocurrencias de perturbaciones extremas.

Otra diferencia notable es la reducción en el volumen de datos que utilizamos, de 500000 a 83000 aproximadamente. Las razones tienen que ver con no poseer los recursos adecuados en el momento de la experimentación, particularmente el hardware. Dicho eso, no notamos que haya una diferencia sustancial en los resultados debido a la reducción.

La metodología de testeo también difiere con la del paper. En vez de armar un dataset nuevo para testear, simplemente separamos un 10 % del dataset de 83000 imágenes para testeo, mientras que el otro 90 % se usó para entrenamiento. No consideramos que puedan haber problemas de overfitting debido a esta elección, ya que los datos están aleatorizados y el conjunto de test no es utilizado hasta dicha etapa.

Si bien el paper utiliza un learning rate variable, nosotros mantuvimos el learning rate base de 0.005. No hay una razón particular para esto más que una cuestión de conveniencia en implementación.

Por último, en vez de utilizar Caffe, nosotros decidimos optar por el módulo de Keras de TensorFlow. Esto otra vez fue por conveniencia y no creemos que debería afectar de ninguna manera el experimento, salvo bugs de la biblioteca.

3. Resultados

Los resultados que mostramos en el siguiente gráfico son una ampliación de los expuestos en el Paper original. Básicamente, tomamos el valor obtenido de la métrica Mean Average Corner Error y la comparamos con el resto de las implementaciones. Dicha métrica representa el promedio, sobre todo el set de imagenes, del error medio (en distancia euclideana) de las cuatro esquinas de cada imagen.

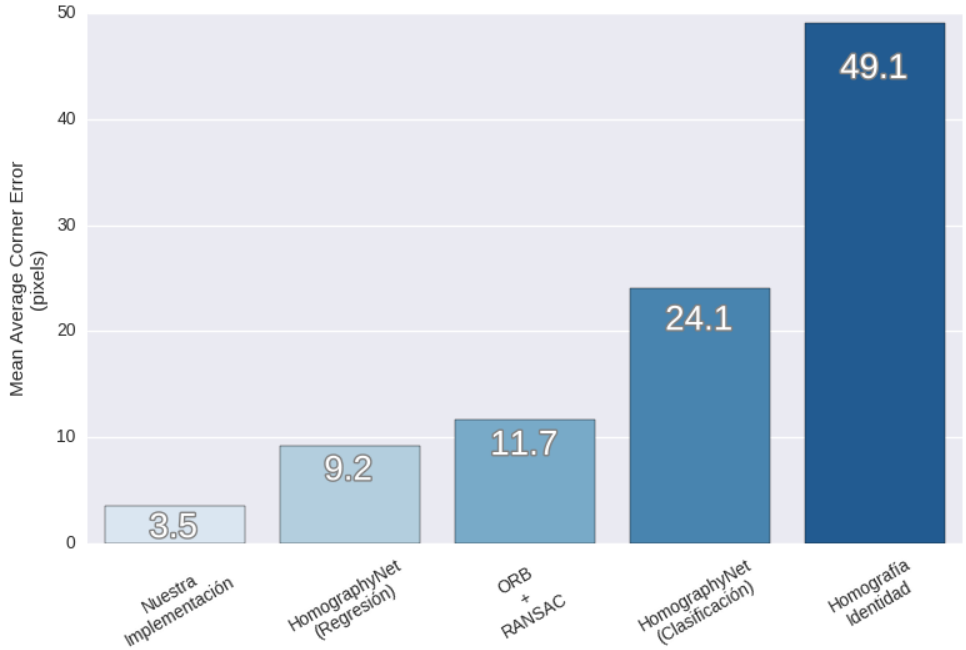

Figura 4: Comparación de estimación de homografías con los resultados del paper

Si bien, a priori, los resultados obtenidos por nuestra implementación lucen superadores, hay que tener en cuenta que aplicamos modificaciones sobre el algoritmo original de armado de dataset. En particular, la perturbación que utilizamos fue de 16 en vez de 32, que era la propuesta original. Creemos que esta es la razón por la cuál nuestro MACE es significativamente menor al de la *HomographyNet* original. Intuitivamente, uno esperaría que el error se duplique en el caso de una perturbación de 32, pero aún en ese caso, nuestra implementación superaría los resultados obtenidos en el paper.

A continuación presentamos los números crudos en etapas de entrenamiento y validación para las métricas de Error Cuadrático Medio y MACE:

```
TRAIN:
        MSE:  0.047
        MACE:  3.906
TEST:
        MSE:  0.043
        MACE:  3.526
```

A modo ilustrativo, mostramos a continuación la reconstrucción de la transformación de una imagen a partir de la homografía estimada con la red.

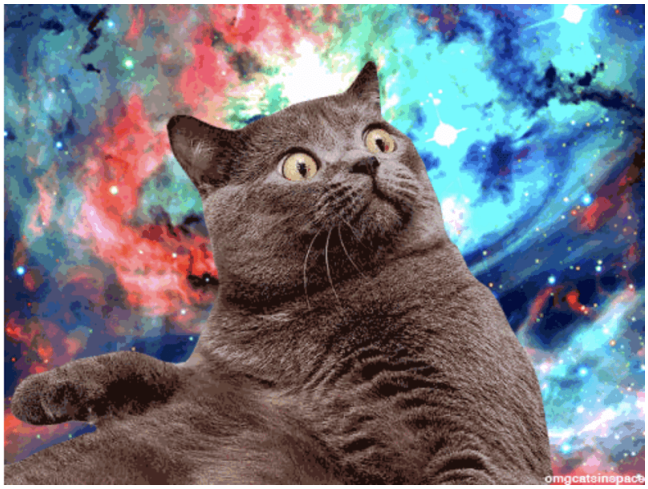

Figura 5: Imagen original

Esta es la imagen original, a partir de la cual se realizó el proceso explicado anteriormente para obtener dos imágenes: un recorte y su perturbación. Estas últimas dos fueron pasadas a la red previamente entrenada con el dataset de aproximadamente 83000 imágenes mencionado y como salida se obtuvo la homografía deseada en la parametrización de 4 puntos. A partir de esta parametrización se obtuvo la homografía en forma matricial con la función `getPerspectiveTransform` de OpenCV y se aplicó a la imagen original con la función `warpPerspective`, recortando en la misma posición que antes.

Es importante notar que es necesario guardar la posición del recorte para poder obtener la homografía en forma matricial. Siendo la posición del recorte (pX, pY), los primeros cuatro puntos a pasar en la función `getPerspectiveTransform` son los correspondientes al cuadrado de 128x128 con esquina superior izquierda en (pX, pY), y el segundo set son estos mismos sumados a las diferencias obtenidas por la red.

(a) Recorte de original

(b) Imagen perturbada (ground truth)

(c) Reconstrucción a partir de la homografía obtenida por la red

4. Conclusiones

Si bien queda pendiente pruebas sin modificaciones significativas, por ejemplo perturbación de 16 vs. 32, creemos que es una aproximación muy fiel debido a que obtuvimos buenos resultados en cuánto a métricas y pruebas empíricas.

Usar redes neuronales para estimar homografías parece ser efectivo y superador en relación a los métodos tradicionales, inclusive con datasets de tamaños substancialmente menores al utilizado en el paper, como mostramos en este trabajo.

Referencias

[1] Daniel DeTone, Tomasz Malisiewicz, y Andrew Rabinovich. *Deep Image Homography Estimation*. 2016