Universidad Autónoma de Yucatán
Facultad de Matemáticas

Red Neuronal con Convolución Dilatada para Estimación del Flujo Óptico Denso

---

TESIS

en opción al grado de:
Maestro en Ciencias de la Computación

presentada por:
I.C. Iván de Jesús Martínez Chin

asesores:
Dr. Víctor Uc Cetina
Dra. Anabel Martín González
Dr. Francisco Javier Hernández López

Mérida, Yucatán, México
14 de enero de 2020

Dedicatoria

A mi madre y padre.

Agradecimientos

Agradezco a mis asesores: Anabel Martín, Víctor Uc y Francisco Hernández por compartir su experiencia y conocimiento, además de brindar motivación. Al CONACYT por el apoyo económico. Y a mi familia y amigos.

Resumen

Esta investigación contiene trabajo experimental, en el cual se propone una red neuronal convolucional con capas de dilatación cuya tarea principal es inferir el flujo óptico denso entre un par de imágenes consecutivas. Durante el proceso de experimentación se proponen diversos modelos los cuales se irán comparando. También se pone a prueba la hipótesis de que al incrementar el campo receptivo en una red neuronal con convoluciones dilatadas se puede mejorar la capacidad de predicción o inferencia. Además que al usar dichas convoluciones es posible reducir el número de parámetros de la red neuronal conservando una predicción aceptable o mejor. Para hacer la experimentación se utiliza una base de datos sintética proporcionada por el estado del arte (*FlyingChairs*). Y por otra parte, con el objetivo de observar más a fondo y poner a prueba lo dicho anteriormente, se crearon diversas bases de datos sintéticas simples para hacer una comparativa numérica, de este modo se puede observar como al ir aumentando la complejidad de los ejemplos de entrenamiento se tienen resultados distintos en los modelos de redes neuronales.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Actualmente, detectar el movimiento en imágenes es pieza importante para diversos sistemas como la vigilancia, o vehículos autónomos [29], [41]. En algunos casos es necesario saber hacia donde se están moviendo ciertos objetos, el entorno o ambos. Por ejemplo, se podría tomar el caso de un sistema de vigilancia o detección de intrusos humanos. Si se pusiera un maniquí de una persona y se usará solo la información de un *frame* o una imagen, un sistema que (de manera ingenua) detecta personas en la escena fallaría y alertaría erróneamente. Si se tuviera la información de flujo de todo el campo visual sería de más utilidad para hacer cierta inferencia. Por tanto, sería mucho mejor tener la información de dos (o más) *frames* consecutivos. Continuando con el ejemplo, de manera sencilla existirían dos variantes para hacer esto, una sería la detección de personas en los dos *frames* y hacer un mapeo de la localización de la persona. Otra forma de hacerse, es calcular campos de flujo óptico y discriminar los movimientos que solo pertenecen a personas en la escena. Este último tendría más ventajas puesto que al calcular todos los movimientos o desplazamientos de los pixeles, se tiene más información y aunque algunos pixeles no tengan su información de desplazamiento muy precisa, se podría corregir de algún modo sabiendo que pixeles le corresponden al desplazamiento de una persona (por ejemplo por votación o haciendo segmentación).

El problema del flujo óptico es muy complejo de resolver, en las secciones posteriores se argumentará el porqué de esto. Existen diversos trabajos basados en métodos variacionales [3], [17], [32], [20]. Los cuales son muy complejos y tardados debido a que se basan en el *matching* denso de muchas características extraídas de la imagen, así como minimización de funciones de energía. Mientras que usando redes neuronales ha habido poca investigación.

Las redes neuronales artificiales (ANNs, por sus siglas en inglés), por otra parte, no

siempre han sido populares y menos para este problema. Ciertamente hubo una época en que varios investigadores abandonaron estos modelos debido a la poca capacidad de cómputo de la época, la cual era una limitante, y estos modelos tienen una gran cantidad de parámetros que deben ser entrenados. Además, son muy difíciles de analizar matemáticamente. Pese a eso, otros continuaron investigando sobre como hacer estos modelos menos pesados (en número de parámetros) y más eficientes, además, con el avance en unidades de procesamiento gráfico (GPUs) y unidades de procesamiento tensorial (TPUs) se ha vuelto más rápido entrenar y hacer inferencia con ANNs.

Sin embargo, en la actualidad las redes neuronales artificiales han ido tomando popularidad, mas aún, desde la introducción de la convolución como operación en estos modelos. Un trabajo muy popular que realzó el ánimo en estos modelos es el de [25], por años este trabajo se ha convertido en uno de los capítulos introductorios para cualquiera que deseé introducirse a las redes neuronales convolucionales y al *Deep Learning*. El uso de las redes neuronales convolucionales (CNNs, por sus siglas en inglés) ha sido una pieza esencial en el desarrollo de varios modelos de predicción y clasificación [2], [31], [16], [37]. Los científicos de datos a veces hacen uso de estos modelos para crear sistemas de predicción [9]. Sin embargo, no existen ANNs y CNNs generales que funcionen para cualquier problema, y el modelo de red muchas veces depende de los datos que se tienen. Por lo que basado en los datos, se construyen dichas redes con variedad de topologías y se hacen estudios experimentales para corroborar la capacidad de predicción de dicha red neuronal. Constantemente surgen nuevas técnicas, topologías y recursos para trabajar con CNNs, algunos de los trabajos más importantes, que muestran lo dicho anteriormente, pueden irse mencionando acorde al problema. Para el trabajo de reconocimiento de imágenes, un buen ejemplo de este tipo es el trabajo de [34], mientras que para clasificación de imágenes un ejemplo muy popular es el trabajo de [24]. Clasificar y reconocer imágenes son problemas que dieron más impulso a la investigación de CNNs. El trabajo de [25] es de reconocimiento de imágenes y en especifico para reconocimiento de dígitos escritos a mano. Actualmente existen CNNs tan avanzadas que pueden dar la segmentación semántica de las imágenes, como en el trabajo de [2], esto significa que dado una imagen se separan los campos como: personas, asfalto, vehículos, áreas verdes, entre otros. También, existe trabajos para localizar y clasificar ciertas entidades que se encuentran en la imagen como en el trabajo de [31].

Este documento contiene trabajo experimental con modelos propuestos de CNNs con convolución dilatada usando diferentes topologías para investigar su desempeño

en el problema de inferir el flujo óptico. También contiene un análisis de resultados bajo distintas configuraciones en el entrenamiento ya que la variabilidad de hyperparámetros a probar son diversos.

1.1. Antecedentes

1.1.1. Métodos para estimar el flujo óptico basados en redes neuronales

No ha habido mucha investigación acerca de esta forma de inferir el flujo óptico, sin embargo, se puede decir que hay tres trabajos considerables en el ámbito los cuales serán descritos.

En [12] se presenta un método para predecir el flujo óptico de secuencias de imágenes utilizando redes neuronales convolucionales. Su motivación de utilizar CNNs es que son buenas desempeñando tareas de clasificación y reconocimiento en imágenes, con invarianza a rotaciones y escala. Para entrenar una CNN de este tipo se necesita una base de datos etiquetada con una gran cantidad de ejemplos. De acuerdo a los autores ésta es una de las desventajas ya que encontrar datos con esas especificaciones es difícil. En este trabajo se encontraron tres conjuntos de datos: el *Middlebury* con sólo 8 pares de imágenes y con desplazamientos pequeños, el *KITTI* con 194 pares de imágenes, el cual incluye desplazamientos más largos, y el *MPI Sintel* con 1,041 pares de imágenes. De acuerdo a los autores incluso estos ejemplos son insuficientes para entrenar la red, así que crearon un conjunto de datos artificial llamado *Flying Chairs* que contiene sillas flotando con sus respectivos desplazamientos etiquetados, este conjunto de datos contiene 22,872 pares de imágenes de ejemplo.

Se proponen dos arquitecturas de red neuronal convolucional. La primera arquitectura llamada *FlowNetSimple* usa un estilo tradicional poniendo los dos *frames* de ejemplo como si fueran canales del mismo mapa de características. En esta red se utilizan 9 capas de convolución. La segunda arquitectura llamada *FlowNetCorr* está divida en dos ramas separadas, en una parte se le da un *frame* y en otra el *frame* siguiente. Estas dos etapas separadas pasan por 3 capas de convolución y finalmente se unen con una operación de correlación cruzada entre los mapas de características. La correlación cruzada es similar a la convolución a excepción por el orden de los índices y que la multiplicación se hace con dos mapas de características, es decir, aquí no hay pesos que se puedan entrenar. Una vez que se han unido las dos ramas de la arquitectura *FlowNetCorr* se pasa por 6 etapas más de convolución. Tanto en *FlowNetSimple*

y *FlowNetCorr* al final de las etapas de convolución se tiene una etapa llamada refinamiento, esta etapa es más bien 4 etapas de lo que se conoce como *upsampling*. En otras palabras, se pasa de una información compacta a un mapa de mayor resolución usando capas de deconvolución. En los resultados se utilizó la métrica de *End Point Error* (EPE), obteniendose los mejores resultados con el dataset artificial *FlyingChairs*. En el caso del conjunto de datos *Sintel* usando el mejor resultado del estado del arte, el cual es *EpicFlow*, las CNNs propuestas en este trabajo muestran ligeramente menor precisión en comparación con *EpicFlow*.

En [21] se presenta la evolución del trabajo propuesto en [12]. En este trabajo se hacen cambios que mejoran drásticamente la predicción del modelo. Uno de los cambios fundamentales que hacen que el modelo mejore es la agregación de *FlowNetS* y *FlowNetC* presentados por [12] de manera apilada, es decir, se ponen consecutivamente etapas de éstas dos redes. Sin embargo, existen más cambios entre cada etapa. Una etapa sub-secuente a otra recibe la predicción de la anterior etapa (el flujo óptico), pero también recibe los siguientes datos: Recibe de nuevo los dos *frames* al inicio de la red, además se agrega un *frame* extra el cual es el *frame 2* que ha pasado por una transformación de mapeo usando el flujo estimado. Adicionalmente a este *frame* extra transformado se agrega otro, el cual es el mismo pero con una agregación de ruido en el brillo. Además de lo anterior se menciona que es posible combinar dos tipos diferentes de red, *FlowNetS* y *FlowNetC*, y se investigó diferentes combinaciones que puedan dar buenos resultados. En este trabajo se llegó a una combinación eficiente la cual consiste en poner una red al inicio que se le llamará *bootstrap* y seguido se pone el otro tipo de red varias veces. Falta describir otra mejora que se agregó en la red, ésta consiste en una bifurcación o red siamesa que aprende en paralelo y luego se combina con el flujo de estimación de la otra rama, retomaremos esta red más adelante. Antes se describirá un punto importante que los autores enfatizan. Consiste en los datos que se presentan a la red. Durante la experimentación los autores se dieron cuenta que no sólo la calidad de los conjuntos de datos es importante, sino que también el orden en el que se presentan durante el aprendizaje. En este trabajo se retoman los conjuntos de datos descritos en [12], en especial *FlyingChairs*, además se tiene otro conjunto de datos importante llamado *Thigs3D*. Este último conjunto de datos es similar al *FlyingChairs* pero los objetos son generados mediante modelos 3D, por lo que representan una mejor aproximación a la realidad debido a las sombras y cambios de iluminación sobre un objeto. Cuando se experimentaba en la red, los autores notaron que entrenar con un conjunto de datos combinado, o uno primero y luego el otro arrojaba resultados

con diferencias significativas. El mejor resultado lo obtenían cuando primero entrenaban con *FlyingChairs* y luego con *Things3D*. La conjetura a la que llegaron fue que debido a que *FlyingChairs* es más simple, proporcionaba un entrenamiento preliminar a la red para que posteriormente pueda hacer un aprendizaje más general. Retomando la estructura de la red, se había dicho que se tiene una bifurcación, ésta red es igualmente una *FlowNetS* que ha sido entrenada con ejemplos de una mezcla especializada de *Thigs3D* y *FlyingChairs* y aplicando una no-linearidad al error con desplazamientos de bajo peso. Al final los flujos predichos de las dos ramas, en conjunto con el *frame 1*, la magnitud de los flujos y los *frames* con agregación de error en el brillo, son la entrada a una red de fusión. Esta red de fusión puesta al final, primero contrae la resolución por un factor de 2 y luego la expande otra vez. La red final fue llamada *FlowNet2*. Los resultados de este modelo fueron muy alentadores, acercándose bastante al mejor método propuesto en el estado del arte llamado *FlowFields*, pero el tiempo en el que le toma hacer una predicción es significativa, aproximadamente de 123ms usando una GPU NVIDIA GTX 1080 (128 veces más rápido que *FlowFields*). Un conjunto de datos para contrastar el desempeño de la red es en *Sintel* (versión limpia), en este conjunto la red obtuvo un *Average Endpoint Error* (AEPE) de 3.96 y *FlowFields* un 3.75, con esto se aprecia un rendimiento muy cercano pero más eficiente en términos de tiempo.

En [30] se presenta otro modelo de red neuronal para estimación de flujo óptico. A diferencia de *FlowNet2* sigue un enfoque distinto para resolver el problema. En este caso se usa el término de pirámides espaciales. De manera breve, se usan varios modelos de red neuronal del mismo tipo a diferentes resoluciones de la imagen. El flujo residual es calculado en cada resolución y transferido a la siguiente red neuronal que está en una escala espacial mayor. Para este modelo se define una función $d(\cdot)$ el cual baja la resolución de un ejemplo y también se tiene la función opuesta $u(\cdot)$ que aumenta la resolución, estos aumentos y decrementos se dan en un factor de la mitad de la dimensión, es decir, si se tiene una imagen de tamaño $m \times n$ al aplicar d la imagen resultante sera de $m/2 \times n/2$ y análogamente con u. También se considera una función $w(I, V)$ que mapea una imagen I de acuerdo a su flujo óptico V. Para denotar el modelo completo se define un conjunto $\{G_0, ..., G_k\}$ de redes neuronales covolucionales donde el subíndice indica cierta resolución a la que esta red trabaja. Así para una red a una resolución k su inferencia vendrá dada por la ecuación 1.1.1 donde v_k denota el flujo residual tal que el flujo en una resolución k es V_k 1.1.2. En la Figura 1.1.1 se observa un ejemplo de los autores para un modelo de 3 niveles de resolución para entender las entradas y salidas de cada red neuronal G_k. Spynet es significativamente más pe-

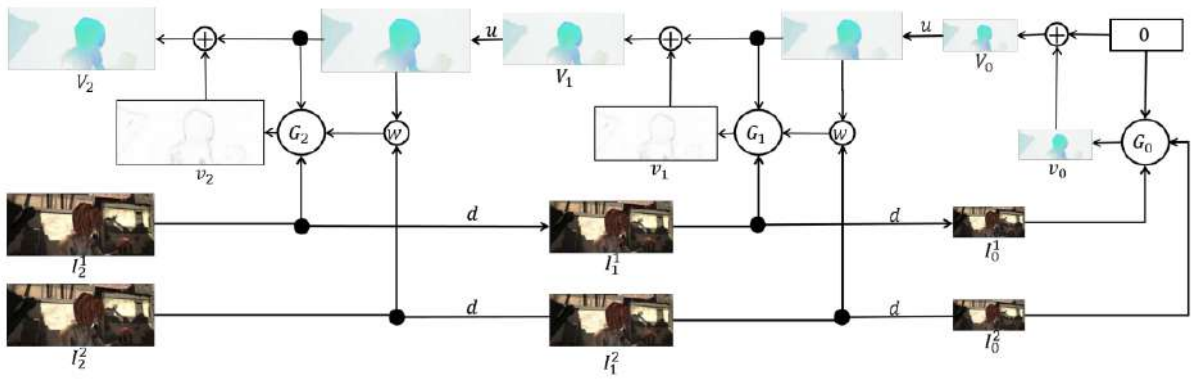


Figura 1.1.1: Modelo ejemplo del spynet de [30] donde se presentan 3 niveles de resolución. Como se observa el nivel 0 recibe las imágenes en la menor escala. Y en cada nivel intermedio, la red G_k calcula el flujo residual dada la predicción de una resolución anterior. En este caso V_2 es la mayor resolución.

queña que FlowNet y por tanto a FlowNet2, los autores argumentan que es 96% más pequeña y por tanto puede correrse en dispositivos más compactos.

$$v_k = G_k(I_k^1, w(I_k^2, u(V_{k-1})), u(V_{k-1}) \tag{1.1.1}$$

$$V_k = u(V_{k-1}) + v_k \tag{1.1.2}$$

En [19] se presenta otro avance de redes neuronales convolucionales para estimación del flujo óptico. Al igual que spynet utilizaron un enfoque piramidal. La red es denominada LiteFlowNet y consiste de dos redes compactas enfocadas en extracción de características de manera piramidal para el flujo óptico. Estas dos redes compactas las denominaron NetC y NetE. NetC es una CNN de doble flujo cuyos pesos son compartidos en los dos flujos, su objetivo es transformar la imagen I en un conjunto de características de alta dimensionalidad y multi-escala como $\{\mathscr{F}_k(I)\}$ desde la más alta resolución espacial ($k = 1$) hasta la más baja resolución ($k = L$). Las características piramidales son generadas mediante convoluciones con cierto *stride* s, donde s denota cierta reducción en la resolución de la imagen. Se usa $\mathscr{F}_i$ para denotar las características de una Imagen I_i y para ser breves se omite el subíndice k correspondiente a la resolución. En cada nivel de la pirámide un campo de flujo es inferido de las características de alto nivel $\mathscr{F}_1$ y $\mathscr{F}_2$ de las imágenes I_1 e I_2. Al igual que spynet proponen una función de mapeo (f-warp) para reducir la distancia de las características de $\mathscr{F}_1$ y $\mathscr{F}_2$. Específicamente, $\mathscr{F}_2$ es mapeado hacia $\mathscr{F}_1$ por f-warp vía el flujo estimado $\dot{\mathbf{x}}$ a

$\tilde{\mathscr{F}}_2 \doteq \mathscr{F}_2(\mathbf{x}+\dot{\mathbf{x}}) \sim \mathscr{F}_1$. Esto permite a la red inferir flujos residuales entre $\mathscr{F}_1$ y $\tilde{\mathscr{F}}_2$ que tiene una magnitud de flujo menor pero no el flujo completo el cual es más difícil de inferir. En este caso f-warp es aplicado en las características de alto nivel generadas y no en las imágenes lo que lo hace más eficiente, de acuerdo a los autores. Para permitir un entrenamiento de fin a fin (*End-to-End*) $\mathscr{F}$ es interpolado a $\tilde{\mathscr{F}}$ para cada desplazamiento de subpixel como sigue

$$\tilde{\mathscr{F}}(\mathbf{x}) = \sum_{\mathbf{x}_s^i \in N(\mathbf{x}_s)} \mathscr{F}(\mathbf{x}_s^i)(1-|x_s - x_s^i|)(1-|y_s - y_s^i|) \tag{1.1.3}$$

donde $\mathbf{x}_s = \mathbf{x}+\dot{\mathbf{x}} = (x_s, y_s)^T$ denota las coordenadas fuente del mapa de características entrante $\mathscr{F}$ que define el punto de ejemplo, $\mathbf{x} = (x, y)^T$ denota las coordenadas de la etiqueta de la rejilla regular en el mapa de características interpolado $\tilde{\mathscr{F}}$ y $N(\mathbf{x}_s)$ denota los cuatro pixeles que rodean $\mathbf{x}_s$. Posteriormente, en cada nivel de la pirámide de NetE, se hace un emparejamiento (*matching*) pixel por pixel de las características de alto nivel que hace un estimado burdo del flujo. Un refinamiento subsecuente de ese estimado mejora la precisión por sub-pixel. En primera inferencia del flujo se establece una correspondencia entre I_1 y I_2 a través de calcular la correlación entre el vector de características de alto nivel en las características de individuales $\mathscr{F}_1$ y $\mathscr{F}_2$ como sigue

$$c(\mathbf{x}, \mathbf{d}) = \mathscr{F}_1(\mathbf{x}) \cdot \mathscr{F}_2(\mathbf{x}+\mathbf{d})/N \tag{1.1.4}$$

donde c es el costo de *matching* entre el punto $\mathbf{x}$ en $\mathscr{F}_1$ y el punto $\mathbf{x}+\mathbf{d}$ en $\mathscr{F}_2$, $\mathbf{d} \in \mathbb{Z}$ es el desplazamiento del vector desde $\mathbf{x}$, N es el tamaño del vector de características. Un campo de flujo completo es calculado como sigue

$$\dot{\mathbf{x}}_m = \underbrace{M(C(\mathscr{F}_1, \tilde{\mathscr{F}}_2; \mathbf{d}))}_{\Delta\dot{\mathbf{x}}} + s\dot{\mathbf{x}}^{\uparrow s} \tag{1.1.5}$$

donde C se define como un costo por volumen agregando todos los costos de *matching* en una rejilla 3D. M es una unidad de *matching* de descriptores donde el flujo residual $\Delta\dot{\mathbf{x}}$ es inferido filtrando los costos por volumen C. La segunda etapa de inferencia de flujo es un refinamiento de subpixel cuyo único propósito es refinar el campo de flujo $\dot{\mathbf{x}}_m$ del nivel pixel a una precisión sub-pixel. Esta unidad de refinamiento es denotada por S. Al final de cada etapa de la pirámide se agrega una unidad de regularización de flujo denotada por R. El esquema completo de este modelo se muestra en la Figura 1.1.2. Este modelo como se observa puede parecer el más complejo de todos, puesto que aplica numerosas funciones de manipulación de información.

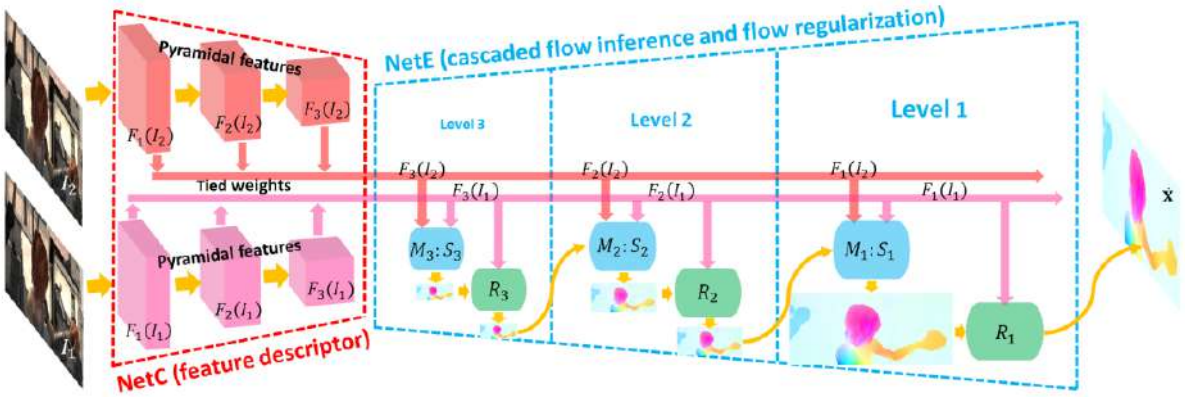


Figura 1.1.2: Estructura de LittleFlowNet [19]. Para facilitar la representación solo se muestran 3 niveles. Dado un par de imágenes (I_1, I_2), NetC genera dos pirámides de características de alto nivel ($\mathscr{F}_k(I_1)$ en rosa y $\mathscr{F}_k(I_2)$ en rojo, $k \in [1,2]$). NetE produce campos de flujo multi-escala los cuales son generados por los módulos antes descritos $M : S$, finalmente se agrega la unidad de regularización R.

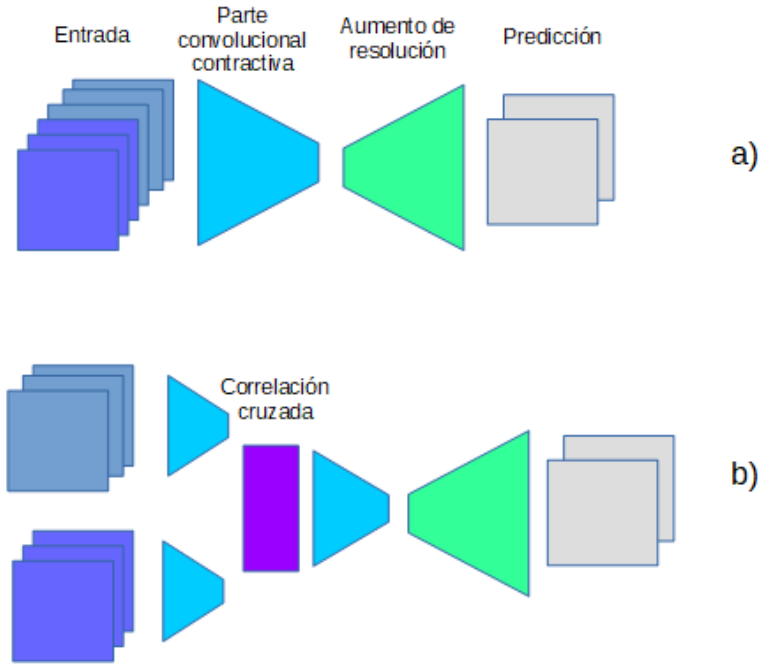


Figura 1.1.3: En a) se presenta la arquitectura de *FlowNetSimple*. En b) se presenta la arquitectura de *FlowNetCorr*.

1.1.2. Métodos variacionales o basados en heurística

Estos métodos aunque son distintos del trabajo presente, es importante tenerlos en consideración. Al contrario de los métodos anteriores ha habido mucha investigación

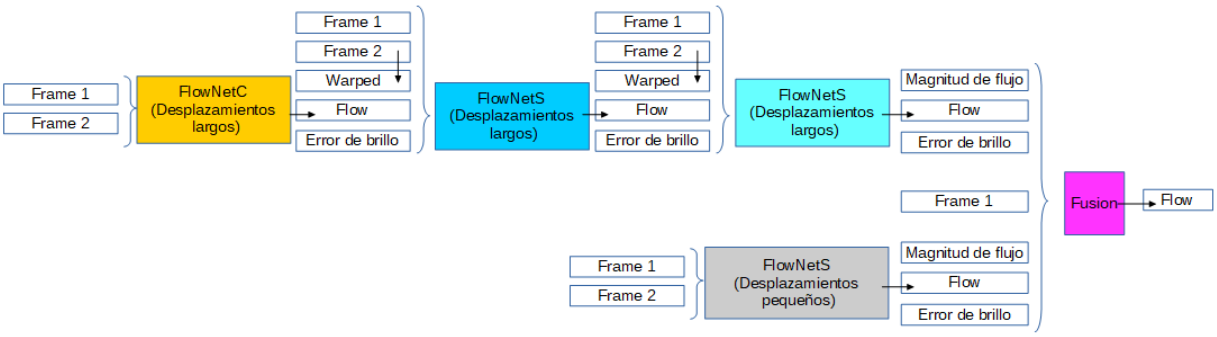


Figura 1.1.4: Arquitectura resumida de *FlowNet2*.

los cuales varían en cada trabajo. Sin embargo, describiremos algunos de los más conocidos.

El trabajo *FlowFields* [3] proviene del método *Approximate Nearest Neighbor Fields* [17]. Primeramente se enfoca en definir un solo *FlowField* el cual es un cuadro alrededor de un pixel en cierta posición y lo que se quiere es calcular los desplazamientos de cada pixel en la primera imagen con respecto a la segunda. En su enfoque se hace uso de los *kd-tree*, donde para cada cuadro alrededor de cierto pixel se calcula la transformada de *Walsh-Hadamard* (WHT), con todos los pixeles en la segunda imagen. Estos vectores son ordenados en el *kd-tree*. Luego, para la primera imagen se calculan de igual manera los vectores WHT y se buscan sus correspondientes hojas en el *kd-tree*. Se usa un error de correspondencia para saber el mejor candidato para cierto cuadro del pixel. Se realiza una propagación del flujo para estimar el flujo óptico. Los autores proponen realizar esta propagación a lo largo de varias escalas o resoluciones. Esto hace que el algoritmo sea robusto ante *outliers*. Este algoritmo fue probado en los conjuntos de datos *MPI-Sintel*, *Middlebury* y KITTI. Los resultados, hasta la actualidad, tienen el mejor reconocimiento. Sin embargo, por el tipo de lógica en la metodología posiblemente no sea paralelizable ya que requiere constantes ordenaciones en *kd-trees* lo que lo hace costoso en tiempo, caso contrario con las redes convolucionales que se benefician de constantes operaciones de algebra lineal.

El ANNF [17] es otro método basado en KD-Trees. El cual igualmente depende de *patch* y son representados en un espacio $3p^2$ dado un *patch* de $p-\text{by}-p$. Y la similitud entre dos *patches* es calculada con la distancia L_2 (definida en 2.3.3) en este espacio. La dimensionalidad de este vector es reducida mediante la transformada de WHT. Posteriormente se pasa a un proceso de construcción del KD-Tree, dado cada conjunto de

candidatos, se escoge la dimensión de mayor dispersión [1] y se divide el espacio por el valor medio de los datos del candidato en esa dimensión. Esta división entre la media sirve para garantizar el balance entre características. El conjunto de candidatos es dividido recursivamente hasta que cada nodo terminal del árbol contiene al menos m candidatos. Posteriormente, se hace una búsqueda basado en una propagación asistida. Y en los últimos pasos se hace un proceso de enriquecimiento, poda del árbol y verificación de candidato y re-categorización.

En EpicFlow [32] se presenta un método de inferencia de flujo óptico conservando los bordes para desplazamientos largos. El cual consiste en dos pasos. El primer paso es una correspondencia densa por una interpolación el cual conserva los bordes. El segundo paso consiste en una minimización de energía variacional inicializado con las correspondencias densas. La interpolación se propone para estimar el campo de correspondencia densa F entre dos imágenes I, I'. Esta interpolación se hace con un conjunto de *matches* de entrada $\mathcal{M} = \{(\mathbf{p}_m, \mathbf{p}'_m)\}$. Cada *match* $(\mathbf{p}_m, \mathbf{p}'_m) \in \mathcal{M}$ define una correspondencia entre un pixel $\mathbf{p}_m \in I$ y un pixel $\mathbf{p}'_m \in I'$. Se consideran dos opciones de interpolación: *Nadaraya-Watson* y *Locally-weighted affine estimation*. Para conservar los bordes se usa la distancia geodésica aproximada. La energía a minimizar es definida como la suma de término del datos y el término de suavidad.

En [20] se presenta otro método usando fusión de datos localmente adaptable. El modelo de fusión empleado puede ser descrito como sigue

$$E = E_{data}(\mathbf{u}, \mathbf{w}) + \eta E_{descr}(\mathbf{u}, \mathbf{w}) + \nu E_{reg}(\mathbf{w}) + \lambda E_{reg}(\mathbf{u}) \tag{1.1.6}$$

donde $\mathbf{w} = \{\mathbf{w}_l\}$ es un conjunto de M variables de peso, $l = 1, 2, ..., M$. $E_{data}(\mathbf{u}, \mathbf{w})$ mide la fidelidad del dato acoplado con el campo de flujo $\mathbf{u}$ y las variables de peso $\mathbf{w}$. $E_{descr}(\mathbf{u}, \mathbf{w})$ mide la discriminación de cada dato del modelo, y $E_{reg}(\mathbf{w})$ y $E_{reg}(\mathbf{u})$ asegura la regularización de los pesos y flujo, respectivamente. E_{data} fue diseñada para minimizar la suma ponderada de los datos del modelo y esta descrita por

$$E_{data}(\mathbf{u}, \mathbf{w}) = \sum_x \sum_{l=1}^{M} \mathbf{w}_l(\mathbf{x}) \cdot \rho_l(\mathbf{x}, \mathbf{u}) \tag{1.1.7}$$

donde $\mathbf{x} \in \mathbb{R}^2$ denota la ubicación de los índices en el dominio de la imagen. La función ρ es usada para medir la suma de diferencias absolutas. El término discriminador de datos es representado por

[1] *Dispersión* en este caso es la diferencia entre la dimensión más grande y pequeña

$$E_{descr}(\mathbf{x},\mathbf{u}) = \sum_{\mathbf{x}} \sum_{l=1}^{M} \mathbf{w}_l(\mathbf{x}) \cdot \sum_{k=1,k\neq l}^{M} e_k(\mathbf{x},\mathbf{u}_0) \tag{1.1.8}$$

donde $e(\mathbf{x},\mathbf{u}_0) = \min(|v_1|,|v_2|)$. Los términos de regularización para $\mathbf{w}$ y $\mathbf{v}$ respectivamente son dados por

$$E_{reg}(\mathbf{w}) = \sum_{x} \sum_{l=1}^{M} |\nabla \mathbf{w}_l| \tag{1.1.9}$$

$$E_{reg}(\mathbf{u}) = \sum_{x} g(\mathbf{x}) \cdot |\nabla \mathbf{u}| \tag{1.1.10}$$

donde g representa la máxima diferencia de color alrededor de una vecindad de pixeles y es descrito como

$$g(\mathbf{x}) = \exp(-\max(|\nabla I_R|,|\nabla I_G|,|\nabla I_B|)^{\kappa}) \tag{1.1.11}$$

y κ controla la magnitud de esta diferencia. El término I con un subíndice representa cierto canal de la imagen. Se aplican procesos de optimizaciones continuas para $\mathbf{u}$ y $\mathbf{w}$ con un algoritmo de proyección en una unidad simple. Además para hacer la optimización se introduce un término $\mathbf{v}$ basado en una simplificación cuadrática, así el problema esta definido por

$$\min_{\mathbf{u},\mathbf{v},\mathbf{w}} \sum_{\mathbf{x}} \sum_{l=1}^{M} \mathbf{w}_l(\mathbf{x}) \cdot \rho_l(\mathbf{x},\mathbf{v}) + \eta \sum_{l=1}^{M} \mathbf{w}_l(\mathbf{x}) \cdot \sum_{k=1,k\neq l}^{M} e_\kappa(\mathbf{x},\mathbf{u}_0) + \frac{(\mathbf{u}-\mathbf{v})^2}{2\theta} + \nu \sum_{l=1}^{M} |\nabla \mathbf{w}_l| + \lambda \cdot g(\mathbf{x}) \cdot |\nabla \mathbf{u}| \tag{1.1.12}$$

y el proceso generalizado se describe en el Algoritmo 1.

Result: Campos de flujo $\mathbf{u}, \mathbf{v}$;
Input: Dos imágenes a color I_r e I_t;
Construir pirámides para la imagen objetivo y referencia;
Inicializa valores primarios y duales $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$;
for $n = 1$ *hasta 100* **do**
    Minimización continua de $\mathbf{u}$ y $\mathbf{w}$;
    Minimización continua de $\mathbf{v}$;
end
Posprocesamiento de oclusiones;
Propagar variables al siguiente nivel de la piramide;
Repetir la optimización en el nivel actual;

Algoritmo 1: Resumen del algoritmo seguido en [20].

1.2. Definición del problema y objetivos

El problema principal de este trabajo es inferir el flujo óptico usando redes neuronales convolucionales. Se considera que estimar el flujo óptico es importante debido a que puede ser usado como mecanismo para saber hacia donde se están moviendo los objetos en una escena o si la misma escena está en movimiento, lo cual puede ser útil en sistemas de navegación. Un factor importante a tener en cuenta es la manera en que se diseñan las redes neuronales convolucionales para esta tarea. Consideramos como una hipótesis que incrementar el campo receptivo de las características es crucial para dar buenas inferencias del flujo óptico. De acuerdo a los modelos de CNNs propuestos en los antecedentes, se puede notar que el campo receptivo de una característica de sus CNNs lo incrementan de dos maneras: la primera manera es que la CNN tenga suficientes capas de convolución para que una característica englobe un conjunto de pixeles vecinos (los cuales son su campo receptivo) tal como es visto en FlowNet [12]. La segunda manera es usar un enfoque piramidal, como en spynet [30], en el que CNNs muy pequeñas se encargan de encontrar patrones en el vecindario a distintas resoluciones. Generalmente, es mejor tener un modelo más pequeño (en número de parámetros) como el spynet ya que una vez entrenado podría ser ejecutado en plataformas de bajo costo. Retomando la hipótesis del campo receptivo de una característica, existe otra manera de incrementar el campo receptivo sin la necesidad de hacer muchas convoluciones normales hasta llegar a cierto nivel de profundidad. En este modelo se utilizan convoluciones dilatadas. Las convoluciones dilatadas han mostrado su efectividad en incrementar el campo receptivo según [2].

Por tanto el objetivo propuesto para este trabajo será el siguiente:

- **Objetivo General:** Diseñar un modelo de **red neuronal convolucional** con **convolución dilatada** que incrementa el **campo receptivo** de las características de alto nivel para inferir el **flujo óptico**.
- **Objetivos específicos**:
    1. Construir un entorno de trabajo con Python y Tensorflow para desarrollar, entrenar y probar estas redes.
    2. Construir modelos partiendo de modelos sencillos para probar su desempeño.
    3. Entrenar y analizar el desempeño de cada modelo propuesto.
    4. Basado en los resultados obtenidos dar un resultado final comparado con un modelo del estado del arte. Y dependiendo de los resultados dar una explicación de las comparaciones obtenidas.

1.3. Justificación

Puesto que no existe mucha investigación acerca de redes neuronales para inferir flujo óptico y los trabajos previos [21], [30], [19], los cuales ciertamente dan buenas predicciones, son complejos y complicados de analizar. Además, no se ha usado dilatación en la convolución para corroborar si incrementar el campo receptivo con dichas capas es beneficioso para la inferencia y puede mantener una buena relación entre el número de parámetros y precisión en su inferencia. Este trabajo motiva el uso de CNNs como métodos potentes y eficientes de clasificación y regresión para resolver tareas de relativa complejidad como el de inferir el flujo óptico de dos imágenes consecutivas. Y también a investigar más acerca de topologías de red que puedan dar una buena inferencia en flujo óptico. Parte importante del diseño propuesto de CNN es que sea de fácil replicación, así como ligero en su número de parámetros.

1.4. Estructura de este documento

Este documento esta estructurado como sigue:

- Primero se dará una introducción breve al aprendizaje máquina.
- Luego se presenta la teoría fundamental de redes neuronales y flujo óptico.
- Posteriormente se dará una metodología para realizar experimentación en la que se desglosan modelos preliminares que puedan hacer una primera inferencia de flujo óptico. También se describe la base de datos a usar y las particiones para prueba, validación y entrenamiento.
- Finalmente se presenta las secciones de experimentación y resultados.

Capítulo 2

Marco Teórico

2.1. Aprendizaje Máquina: Regresión y Clasificación

De manera breve una tarea de aprendizaje máquina puede dividirse en aprendizaje supervisado y no supervisado. Para cualquier modelo de predicción, como las redes neuronales, es necesario datos de ejemplo, pero estos datos pueden o no estar etiquetados. Es decir, suponga que un ejemplo es simplemente un vector $\mathbf{X}$ el cual contiene n cantidad de características ($\mathbf{X} = x_1, x_2, ..., x_n$). Para una tarea de aprendizaje no supervisado solo basta con ejemplos del tipo mostrados anteriormente. Sin embargo, para una tarea de aprendizaje supervisado es necesario que los datos contengan su respectiva etiqueta, es decir, cada ejemplo será un par de vectores $(\mathbf{X}, \mathbf{Y})$. Este trabajo se enfoca en aprendizaje supervisado.

A su vez, una tarea de aprendizaje máquina se puede dividir en otras dos modalidades la primera es conocida como clasificación. En clasificación los datos de etiqueta (si se tienen) así como la predicción de un ejemplo debe ser un número discreto, este número representa la clase a la que pertenece el dato por tanto es evidente que él número de clases debe ser finito y discreto. Por otro lado tenemos una tarea de regresión, en este caso la etiqueta del ejemplo, así como la predicción que debe sacar el modelo deben ser números de punto flotante (se usa este nombre porque al final una computadora da la predicción pero puede entenderse como un número real). Este trabajo se enfoca en una tarea de regresión.

2.2. Redes Neuronales Convolucionales

2.2.1. Convolución

En esta sección daremos una breve introducción a la redes neuronales convolucionales, siguiendo las convenciones presentadas en [15]. Las redes neuronales convolucionales o CNNs, por sus siglas en inglés, se han vuelto una herramienta fundamental para el aprendizaje profundo.

Primeramente se define el operador de convolución para el caso de un arreglo de 2 dimensiones, como una imagen I, con un kernel igualmente de dos dimensiones, llámese K. Esta operación esta definida por la ecuación 2.2.1.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.2.1}$$

Una propiedad interesante de la convolución es que es conmutativa. Por lo que la ecuación 2.2.1 se puede escribir como la ecuación 2.2.2. Esta propiedad surge porque el kernel se ha volteado en ambas direcciones (es decir, se han invertido ambos ejes del kernel). La única razón para voltear el kernel es para obtener esta propiedad.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.2.2}$$

La propiedad de conmutatividad usualmente no es importante en cuanto nos referimos a términos de implementación en una red neuronal. De hecho muchas bibliotecas de aprendizaje máquina utilizan una función relacionada con la convolución llamada correlación-cruzada. El cual es lo mismo que la convolución pero sin voltear el kernel, esta función esta descrita en la ecuación 2.2.3.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.2.3}$$

Normalmente en las bibliotecas de aprendizaje máquina no se hace una distinción entre la convolución y la correlación-cruzada y llaman estas dos como si fueran la misma. La utilización de una operación con la otra depende del contexto en el que se aplique. En las CNNs al momento de entrenar, el kernel aprenderá los valores en determinadas posiciones del kernel por lo que es irrelevante si se voltea o no, pero si se debe tener en cuenta que pesos son los que afectan o se comparten con las siguientes fases de la red, es decir, los posiciones de los pesos se acomodan de acuerdo a la operación que se aplique y se aprenderán como lo indique la función de costo. Un ejemplo representado gráficamente de la convolución, sin voltear el kernel, se observa

en la Figura 2.2.1

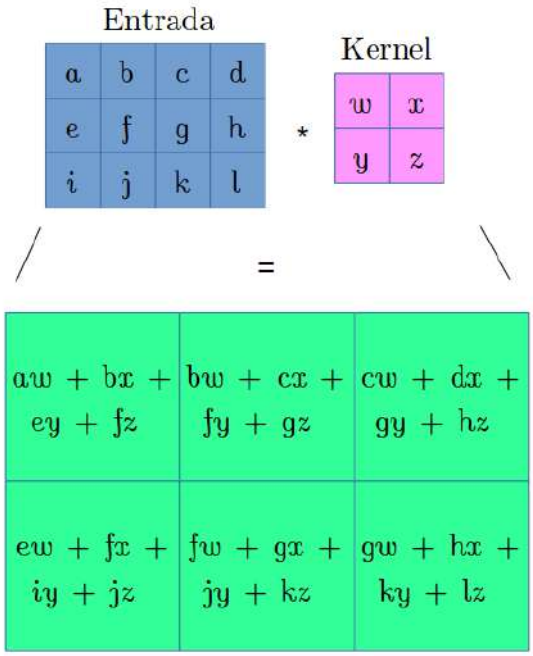


Figura 2.2.1: Representación gráfica de la operación de convolución sin voltear el kernel (es decir, correlación-cruzada).

La operación de convolución en las redes neuronales nos lleva a tres ideas importantes que mejoran el desempeño del sistema de cómputo: **Interacciones esparcidas**, **parámetros compartidos** y **representaciones equivariantes**. Cuando nos referimos a las redes neuronales convencionales (es decir, las totalmente conectadas), es de notarse que cada entrada de una capa anterior tiene interacción con todas las salidas. Por otra parte, las CNNs tienen **interacciones esparcidas**, que también suele referirse como conectividad esparcida o pesos esparcidos. En el caso de las CNNs, si lo vemos de atrás hacia adelante, una entrada no interactúa con todas las salidas, sin embargo, en capas más profundas se llega a un punto en que esa entrada empieza a tener una interacción. Para entender mejor esto se puede observar las figuras 2.2.2 y 2.2.3. Lo anterior nos lleva a un término importante conocido como el **campo receptivo**. El campo receptivo se refiere a la interacción que tiene una salida con entradas de capas anteriores.

Las interacciones esparcidas disminuyen el costo computacional al momento de hacer los cálculos en la propagación. La otra característica que conlleva la convolución son los **parámetros compartidos**, el cual, restringe que algunas conexiones tengan

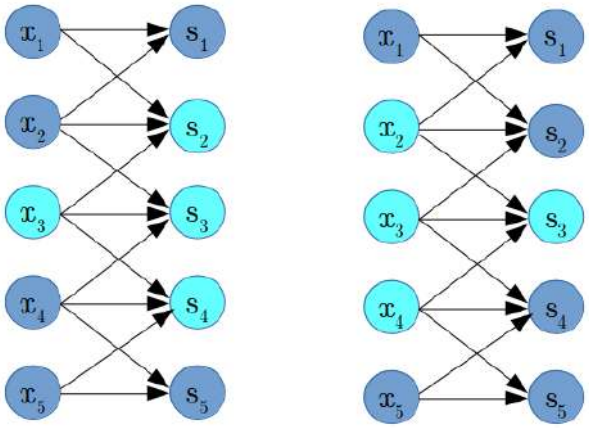


Figura 2.2.2: Representación de las interacciones esparcidas. (A la izquierda) Si lo vemos de adelante hacia atrás vemos como la entrada x_3 interactúa solo con las salidas s_2, s_3, s_4. (A la derecha) Si lo vemos de atrás hacia adelante vemos como la salida s_3 interactúa con las entradas x_2, x_3 y x_4 el cual se conoce como el campo receptivo de s_3.

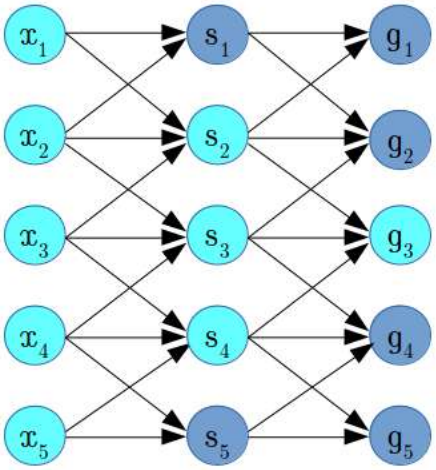


Figura 2.2.3: Representación del campo receptivo en etapas más profundas (resaltadas en color cían). Como se observa la salida g_3 tiene interacción con todas las entradas al inicio

pesos independientes. Regresando a las redes neuronales estándar, como se sabe cada neurona tiene una conexión con cada salida cuyos pesos es independiente de algún otro. Esto conlleva un costo mayor tanto en memoria como en procesamiento. En las CNNs por lo contrario algunas conexiones tienen el mismo peso, esta propiedad también se le conoce como **pesos atados**. La Figura 2.2.4 ayuda a entender mejor este concepto.

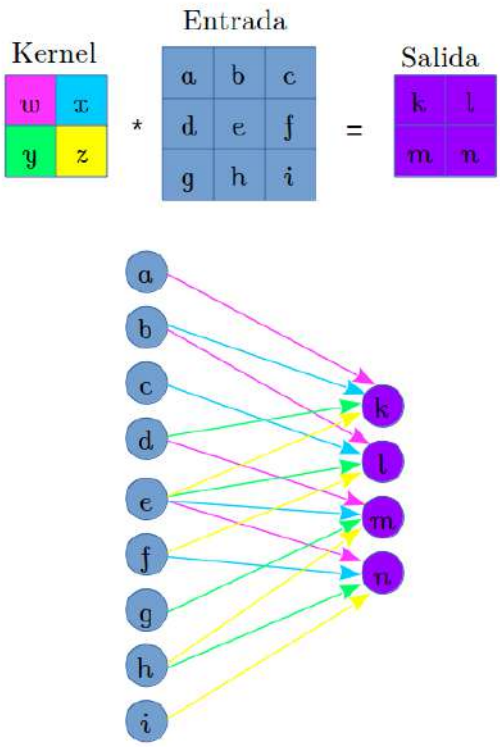


Figura 2.2.4: Representación gráfica de los **pesos atados**. (Arriba) Un ejemplo de convolución donde lo importante a observar son los colores que corresponden a los pesos del kernel. (Abajo) La convolución se arregló a la manera de una red neuronal estándar, los colores indican que corresponden al mismo peso. Como se observa, todas las salidas comparten un mismo peso, pero que proviene de una entrada en particular.

Ahora se hablará de la **equivarianza** en la convolución. En específico una capa de convolución en las CNN tiene la propiedad de **equivarianza** a la traslación. Para recordar esta propiedad definamos dos funciones $f(x)$ y $g(x)$, entonces decimos que $f(x)$ es equivariante a $g(x)$ si $f(g(x)) = g(f(x))$. Para entender como funciona esto en una capa de convolución dejamos, por ejemplo, g como una función que hace un corrimiento a una entrada (en especifico una imagen). Si primero aplicáramos la función g a la entrada y luego se realiza la convolución sería igual que primero realizar la convolución y luego aplicar g.

Además de hacer los cálculos más eficientes y ocupar menos memoria, uno de los motivos para usar convolución en la redes neuronales, puede apreciarse desde el punto de vista de procesamiento de imágenes. En procesamiento de imágenes, se realiza esta operación para resaltar bordes o características de distintos tipos. Estás

características más adelante sirven para hacer operaciones de más alto nivel, pero son fundamentales. Al realizar convolución en las CNNs sucede lo mismo, solo que dejamos a la red aprender estos filtros. Las características que extrae una capa de la CNN sirven más adelante para procesamiento de más alto nivel. Es decir, mientas más profunda es la red, los filtros se vuelven más complejos puesto que aprenden filtros basados en las características de una capa previa.

2.2.2. Convolución Transpuesta

La capa de convolución tiene una peculiaridad que es natural de la operación dado que está discretizada. Cada vez que se aplica la convolución se pierde una cierta cantidad de información en la entrada. Si se observa de nuevo el ejemplo de la Figura 2.2.1 se tiene que la entrada es de 3×4 y la salida de 2×3, esto significa que los bordes se perderán en la siguiente capa, por lo general esto depende del tamaño del kernel. Este podría solucionarse agregando bordes de 0 en la entrada para que la salida sea igual que la entrada en dimensión (lo que se conoce como *padding*), pero esto solo servirá para conservar la dimensión. En cierto caso si la red es demasiado profunda la información se compactará demasiado. Existen casos en que se desea aumentar la dimensión o resolución del mapa de características de entrada. Para lo anterior se diseñó una capa conocida como ***convolución transpuesta*** o también *fractionally strided convolution* (a veces suelen referirse como deconvolución pero existe ambigüedad en este término). Para entender mejor esta operación hay que ver la convolución como una operación de álgebra lineal. Retomando el ejemplo de la Figura 2.2.1, la entrada se arregla en un vector $\mathbf{I} = (a,b,c,d,e,f,g,h,i)^T$ y el kernel queda arreglado como se muestra en 2.2.4.

$$K = \begin{bmatrix} w & x & 0 & y & z & 0 & 0 & 0 & 0 \\ 0 & w & x & 0 & y & z & 0 & 0 & 0 \\ 0 & 0 & 0 & w & x & 0 & y & z & 0 \\ 0 & 0 & 0 & 0 & w & x & 0 & y & z \end{bmatrix} \tag{2.2.4}$$

La salida desenrollada en un vector $\mathbf{O} = (k,l,m,n)^T$ estaría dado por $\mathbf{O} = K\mathbf{I}^T$. Nótece que si se realiza la operación $\mathbf{I}K^T$ se obtiene el paso en reversa de la operación, obteniendo el tamaño original. Un ejemplo ilustrativo de esta operación en 2 dimensiones se encuentra en la Figura 2.2.5, lo importante de notar de esta operación es como de una entrada de 2×2 se pasó a una salida de 4×4.

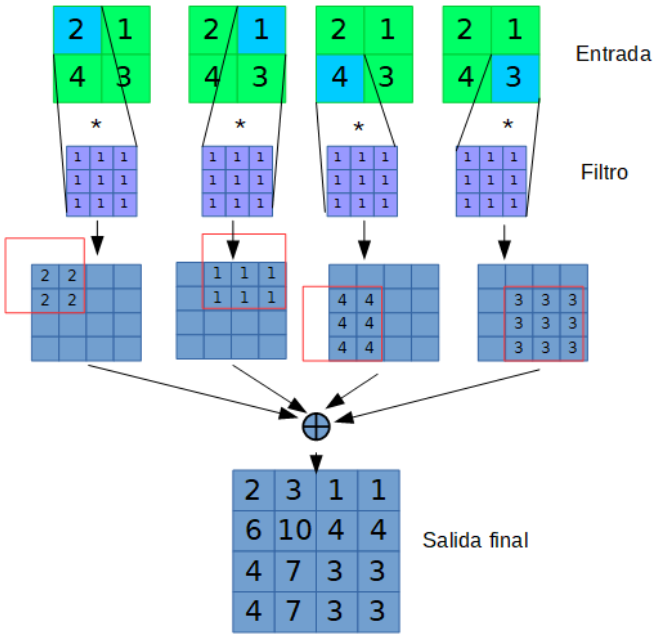


Figura 2.2.5: Ejemplo simple de la convolución transpuesta.

2.2.3. Convolución Dilatada

La convolución dilatada y también conocida en inglés como *atrous convolution*, son una manera de realizar la misma convolución pero ”inflando” los filtros, en el trabajo [42] se aprecia más a fondo el funcionamiento de esta convolución. De esta forma es similar a dejar espacios entre los pesos y dentro de estos espacios solo poner ceros. Con lo dicho anteriormente se agrega un hyper-parámetro más a la convolución el cual es el factor de dilatación d. Cuando tenemos un factor de dilatación $d = 1$ nos estamos refiriendo a una convolución normal. La idea de usar esta forma de convolución es incrementar el campo receptivo del filtro pero sin aumentar el número de parámetros y número de operaciones. Sin embargo, al dejar esos espacios ”vacíos” se puede llegar a perder cierta información de la vecindad por lo que su uso debe ser cuidadoso y en especifico para ciertos problemas. Véase la Figura 2.2.6 como ejemplo para entender este concepto.

2.2.4. Activación

Una CNN por lo general consiste en capas de tres etapas (aunque no siempre). La primera capa es la convolución que se vio previamente, la convolución genera varios mapas de características dependiendo de la forma y número de filtros en el diseño.

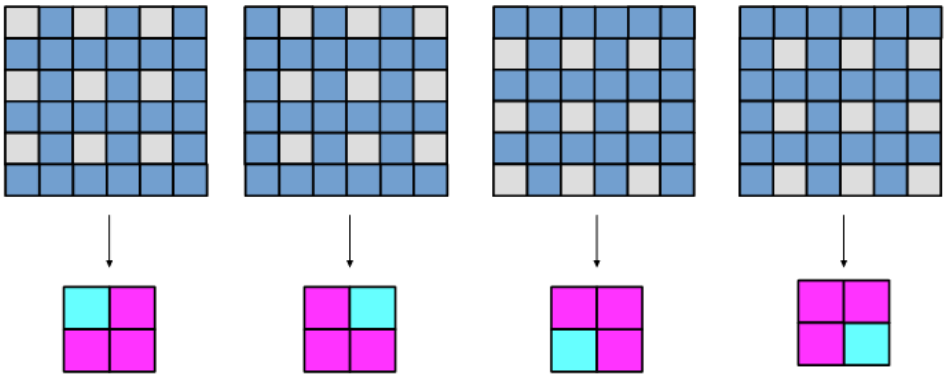


Figura 2.2.6: Ejemplo sencillo de la convolución dilatada. En este caso se tiene una entrada de 6×6 y un kernel o filtro de 3×3 dilatado por un factor de $d = 2$. Los espacios del filtro están representados en gris y como se observa existe espacios en blanco entre el filtro. También es de observar que al aplicar la dilatación la salida tiene una dimensión menor que cuando se aplica una convolución normal.

Estos mapas de características pasan por una función no lineal que se conoce como **activación**. La activación viene siendo la segunda etapa de una capa. Existen diversas funciones (ver Figura 2.2.7) de activación entre las más usadas se encuentran las siguientes: *sigmoid* 2.2.5, tangente hyperobólica 2.2.6, *Rectifier Linear Unit* (ReLU) 2.2.7, *Leaky Rectifier Linear Unit* (LReLU) 2.2.8, *Parametric Rectifier Linear Unit* y *Exponential Rectifier Linear Unit* (ELU) 2.2.9.

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2.2.5}$$

$$f(x) = \tanh(x) \tag{2.2.6}$$

$$f(x) = \max(0, x) \tag{2.2.7}$$

$$f(x) = \begin{cases} \alpha x & \text{if} \quad x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.2.8}$$

$$f(x) = \begin{cases} \alpha(\exp(x) - 1)) & \text{if} \quad x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.2.9}$$

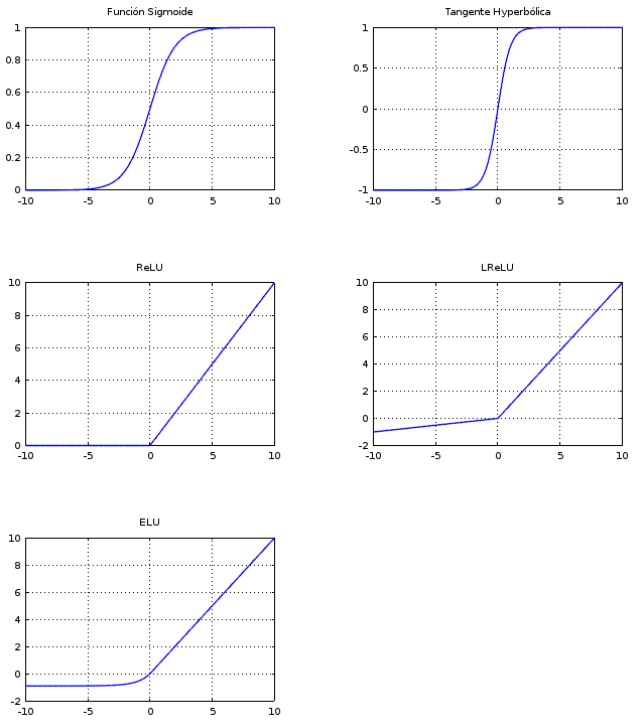


Figura 2.2.7: Funciones de activación usualmente usadas para CNNs. El único caso que no es graficado es PReLU ya que el parámetro α es parte del aprendizaje de la red. Para la gráfica de ELU se usó $\alpha = 0{,}9$ solo para hacer notar la curvatura exponencial, usualmente es un valor más pequeño. Para LReLU se usó $\alpha = 0{,}1$, normalmente se usa $\alpha = 0{,}01$

.

2.2.5. Submuestréo

Submuestréo o también conocido como *Pooling* viene siendo la tercera etapa de una capa. Una de las funciones de esta capa es la reducción de la información de la salida. Generalmente compacta la información con base en estadísticas de la salida. Una de las funciones de submuestreo más conocidas es el *maxpooling*, el cual toma el máximo de una región rectangular de $S \times S$. Para esta operación se define un espaciamiento F y la región de $S \times S$, similar a la convolución, la ventana de $S \times S$ se desliza a lo largo de la salida dando saltos de F y se realiza la operación. Algo interesante de notar es que si el espaciamiento F es de 1 y se calculara el promedio de la ventana, seria equivalente a hacer la convolución con un kernel de solamente 1's. Para el ejemplo de *maxpooling* se puede observar la Figura 2.2.8. Esta operación se cataloga como *downsampling* debido a su efecto de reducir la dimensión.

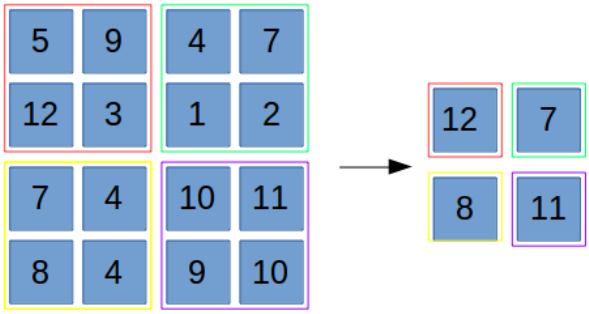


Figura 2.2.8: Ejemplo de aplicación de *maxpooling*, el tamaño de la región es de 2×2 y el espaciamiento es de 2. Es decir la región de la ventana da pasos de a 2 pixeles y calcula el máximo en una región de 2×2.

2.2.6. *Unpooling*

Esta operación puede ser considerada como la operación opuesta del submuestréo. Por ejemplo en *maxpooling* reducimos la dimensión tomando el valor máximo de una ventana en la entrada. En *unpooling* incrementamos la dimensión espacial de la imagen generalmente repitiendo el valor de un pixel en la vecindad o agregando ceros a sus vecinos. Uno de los más conocidos es el *maxunpooling*. Cuando tenemos redes *Fully Convolutional*, las que se describirán más tarde, usualmente son simétricas. Al referirnos con simetría es que si en cierta etapa aplicamos *maxpooling*, más adelante debe estar su correspondiente *maxunpooling*. Al momento de hacer *maxpooling* y tomar el valor máximo representativo, podemos perder cierta información del mapa de características como los bordes u otros detalles finos. Una manera de tratar de recuperar esa información espacial (aunque no toda) es que en la etapa de *upsampling* pongamos el valor máximo en el lugar donde fue tomado en su respectiva etapa de *downsampling*. En la Figura 2.2.9 se puede apreciar un ejemplo de esta operación, tomando en cuenta que la estructura de red es simétrica.

2.2.7. Dropout

Dropout cae en el tema de regularizadores, puesto que tiene el efecto de regularizar los pesos del modelo. La idea de dropout es desactivar o ignorar ciertas neuronas de manera estocástica al entrenar por cada paso de entrenamiento. Es decir, en un paso de entrenamiento se selecciona de manera ”aleatoria” unas neuronas y estas neuronas no se toman en cuenta para hacer el paso de propagación hacía adelante y hacia atrás, esto es todo un paso de entrenamiento, en el siguiente paso de entrena-

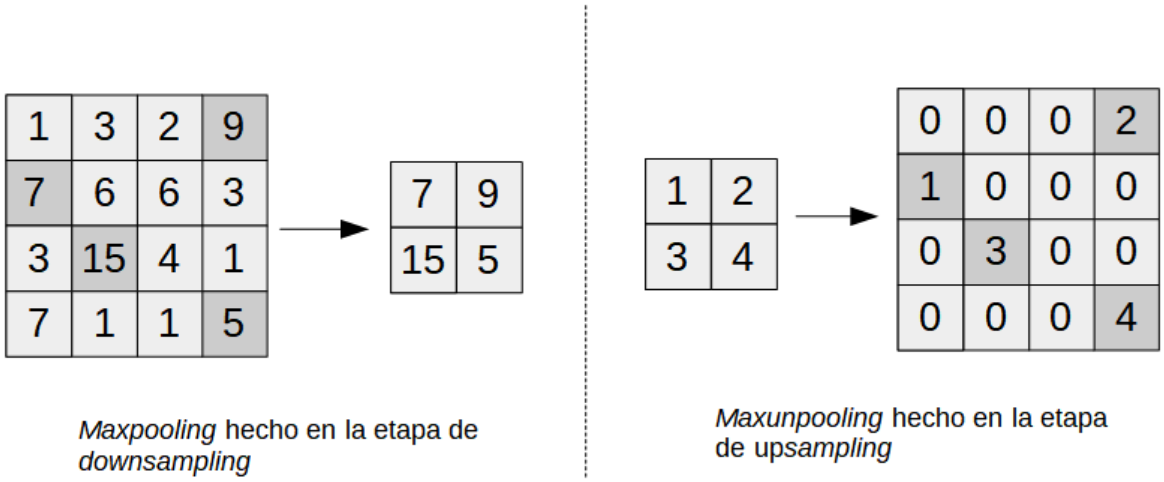


Figura 2.2.9: Ejemplo en una red *Fully convolutional* simétrica, donde el *maxpooling* tiene su respectivo *maxunpooling*. Como se observa la posición de donde se tomó el valor máximo en la etapa de *downsampling* es recordada para asignarse posteriormente en la etapa de *upsampling* con el objetivo de perder menos detalles del mapa de características.

miento se vuelven a seleccionar otras neuronas de manera aleatoria y se repite este procedimiento. Esto implica, que al cancelar estas neuronas se obliga a otras neuronas a centrarse en características especificas sin depender de la totalidad de la salida de otras capas de neuronas. Demasiada cooperación entre neuronas puede hacer dependiente unas neuronas y podrían fallar en distinguir ciertas características lo que conlleva a un problema de *over-fitting*. Cada vez que un conjunto de neuronas son ignoradas lleva a tener un modelo distinto por cada iteración. Si suponemos un modelo de red que contiene N neuronas entonces el número de subconjuntos de redes que pueden formarse son N^2. Promediando la predicción de cada uno de los diferentes modelos reduce la varianza del ensamble y reduce el *over-fitting* y en general puede llegar a tenerse mejores predicciones. En la Figura 2.2.10 se tiene un ejemplo para aclarar este concepto.

La siguiente cuestión es la política para cancelar las neuronas, mencionamos anteriormente que puede ser de manera aleatoria, sin embargo, el termino aleatorio en estadística puede ser representado mediante una distribución de probabilidad. Una de las maneras más comunes de realizar dropout es que en cada capa de la red a cada neurona se le asigne una distribución de Bernoulli 2.2.10. Denotando una salida de una red en cierta capa l como $\mathbf{z}^l$ y un tensor $\mathbf{r}^l \sim Bernoulli(p)$ entonces la salida real de la capa sera $\mathbf{z}^l \cdot \mathbf{r}^l$ (esta es una operación punto a punto). En esta representación

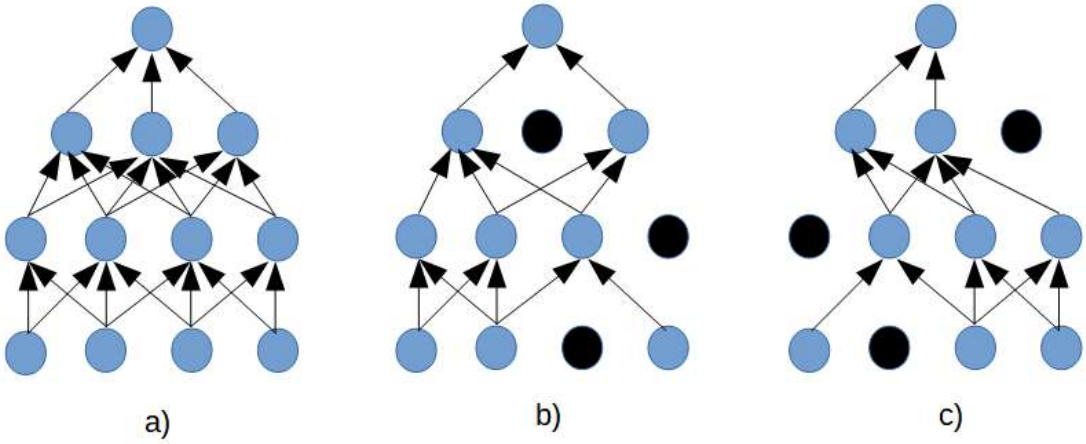


Figura 2.2.10: Ejemplo de aplicación de dropout. En a) se tiene un modelo de red ejemplo en el cual todas sus neuronas están activas, es decir sin dropout. En b) tenemos el mismo modelo el cual está en una iteración k de entrenamiento, en esa iteración las neuronas en negro se han desactivado, por tanto todas sus conexiones entrantes y salientes son como si no existieran. En c) se ejemplifica el mismo modelo pero en otra iteración más adelante, aplicando que en cada paso de entrenamiento el modelo cambia de acuerdo a las neuronas que se hayan seleccionado para desactivar.

omitimos la definición de las dimensiones de la salida y entrada para representarlo genéricamente. Pero $\mathbf{r}^l$ al estar distribuida con Bernoulli significa que solo contendrá ceros y unos dando el efecto de cancelar la neurona. Análogamente se realiza con la retropropagación de los gradientes, al multiplicar los gradientes por esta misma distribución los pesos no se actualizarán. Este método es bien explicado en [35]. Solo para enfatizar, este método solo se aplica cuando el modelo de red esta en modo de entrenamiento, cuando el modelo ya esta entrenado y listo para despliegue ya no se elige que neuronas se deben eliminar, se utiliza el modelo como comúnmente se hace.

$$f(x) = p^x(1-p)^x,\ x \in \{0,1\} \tag{2.2.10}$$

2.2.8. Introducción a Tensores como tipo de dato y flujo de información

Usualmente la información la representamos como cierta estructura de datos en cualquier modelo de predicción. La forma comúnmente usada es el vector, si embargo en ocasiones se necesita una extensión de esta representación. Por ejemplo para representar una imagen es escala de grises se necesitaría una representación matri-

cial. Para representar una imagen multicolor se necesita una matriz de múltiples canales. Y a veces es necesario extensiones aún mayores. Para esto se diseño una estructura llamada **Tensor** el cual es la generalización de cualquier representación multi-dimensional matemática. TensorFlow [1] y otras bibliotecas hacen uso de esta estructura de datos para representar la información y el cómputo a través de estos. Para ser breves un Tensor, desde el punto de vista de computación, se puede decir que cuenta con tres características fundamentales que lo definen:

- Rango. El rango de un tensor define el número de dimensiones con el que cuenta. Otras formas de referirse a este término son como **orden** o **grado**.
- Forma. La forma (o *shape* en inglés) especifica cada una de las dimensiones del Tensor. Esta característica es más especifica y usualmente se usa un vector para representarlo.
- Tipo de Dato. El tipo de dato que almacena este Tensor.

La Tabla siguiente contiene algunas especificaciones de tensores de acuerdo al rango para entender este concepto.

Rango	Entidad Matemática	Shape Ejemplo
0	Escalar	[]
1	Vector de n-valores	$[n]$
2	Matriz de $n \times m$	$[n, m]$
3	Matriz de c-canales	$[n, m, c]$
4	Vector de matrices de c-canales	$[n, m, c, k]$

Sin embargo, el orden de las dimensiones depende de como se definan para cada problema, puesto que esta representación es una generalización, individualmente puede ser interpretada como se convenga. La forma más común de representar ejemplos para redes neuronales convolucionales en imágenes es que la información sea contenida en tensores de rango 4 donde el shape esta definido como $[B, H, W, C]$, donde B representa la dimensión del *batch* o número de ejemplos por paso de entrenamiento, H representa la dimensión de las filas de la imagen, W representa la dimensión de las columnas y C la dimensión de los canales. Sin embargo, otros formatos pueden ser adoptados.

2.2.9. Aritmética de operaciones

Normalmente es necesario saber el comportamiento de las operaciones de convolución y submuestréo con respecto a las modificaciones que le hacen a una entrada y

lo que se espera de salida. Cuando se trabaja con entornos de trabajo de *Deep Learning* de alto nivel como Keras [11] normalmente uno no se preocupa por eso ya que las funciones calculan automáticamente las dimensiones de salida. Siguiendo el estándar tomado en [13] se establecen ciertas formulas para encontrar los valores de dimensión de una salida al aplicarle cierta operación. Ya hemos discutido el funcionamiento de la convolución anteriormente. Ahora, se establecerán las siguientes definiciones: sea k el tamaño de un kernel o filtro de convolución cuadrado (mismo número de filas y columnas), se define el *striding* como s el cual son los saltos de pixeles que puede tomarse durante la convolución tanto en la filas como columnas, se define un *padding* p como las columnas y filas de ceros que se agregan intencionalmente a una entrada en sus bordes (esto último usualmente se hace para asegurar que el tamaño de la salida sea igual que la entrada).

No *padding* de ceros, con *stride* unitario

Este caso es la convolución discreta que se aplica normalmente sobre señales e imágenes, puesto que $p = 0$ y $s = 1$. Y se tiene una entrada de tamaño i (asumiendo que es cuadrada). Para este caso se puede establecer la relación 1. Véase la Figura 2.2.11 como ejemplo de esta relación.

Relación 1. *Para cada i y k y para $s = 1$ y $p = 1$,*

$$o = (i - k) + 1$$

donde o es el tamaño de salida después de aplicar la operación.

Padding con ceros, con *striding* unitario

Ahora consideremos que $p > 1$, es decir, en los bordes de la entrada se le agregan filas y columnas de ceros definidos por p. Entiéndase que si por ejemplo se agrega $p = 2$ entonces quiere decir que se agregan 2 filas al inicio y final, y dos columnas al inicio y al final. También se considera $s = 1$. Dicho lo anterior se tiene la relación 2. Véase la Figura 2.2.12 como ejemplo.

Relación 2. *Para cualquier i y k y p con $s = 1$ se tiene que*

$$o = (i - k) + 2p + 1$$

Same padding

Algunas veces es útil que nuestra operación tanto de convolución como submuestréo tenga la propiedad de que el tamaño de la entrada sea igual que la salida ($o = i$). Esto puede ser logrado estableciendo cierto *padding* que cumple la relación 3. En pocas palabras usando un padding $p = \lfloor k/2 \rfloor$ obtenemos la misma dimensión de entrada, con la única condición que k es impar. Véase la Figura 2.2.13 como ejemplo.

Relación 3. *Para cualquier i y para k un número impar $k = 2n+1,\ n \in \mathbb{N}$, con $s = 1$ y $p = \lfloor k/2 \rfloor = n$.*

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k-1) \\ &= i + 2n - 2n \\ &= i \end{aligned}$$

Full padding

En el caso contrario que se quisiera que la salida sea de una dimensión mayor hay que agregar el *padding* adecuado. Véase la Figura 2.2.14 como ejemplo.

Relación 4. *Para cualquier i y k, y para $p = k-1$ y $s = 1$*

$$\begin{aligned} o &= i + 2(k-1) - (k-1) \\ o &= i + (k-1) \end{aligned}$$

Sin *padding* de ceros, *stride* no unitario

Ahora se considera el caso en que el filtro o kernel pueda moverse dando saltos entre pixeles es decir $s > 1$. Para este caso se puede establecer la relación 5.

Relación 5. *Para cualquier i, k y s. Y para $p = 0$*

$$o = \lfloor \frac{i-k}{s} \rfloor + 1$$

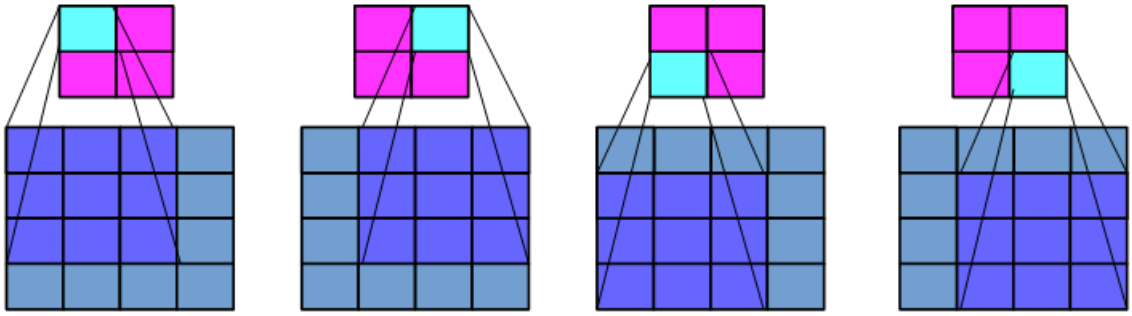


Figura 2.2.11: Ejemplo de convolución cuando se tiene la relación 1, los recuadros están apilados y vistos desde arriba para no crear confusión. En este ejemplo particular tenemos un kernel de 3×3 y el tamaño de entrada es de $4x4$. Note que la salida es de 4×4 cumpliendo la relación dicha.

Con *padding* de ceros, *stride* no unitario

Este es el caso más general puesto que los valores de todos los parámetros pueden ser cualquiera. Para este caso se puede establecer la relación

Relación 6. *Para cualquier i, k y s.*

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1$$

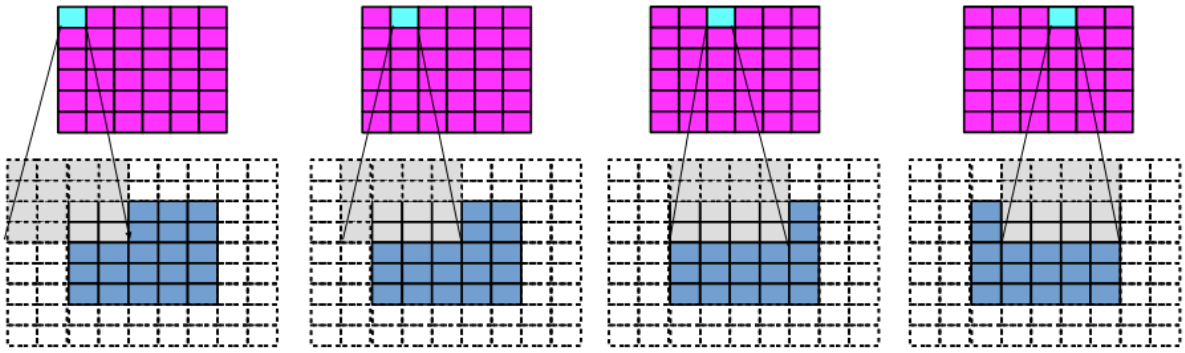


Figura 2.2.12: En este ejemplo se presenta un *padding* arbitrario y *stride* unitario. Para este ejemplo en particular $i = 5$, $k = 3$ y $p = 2$.

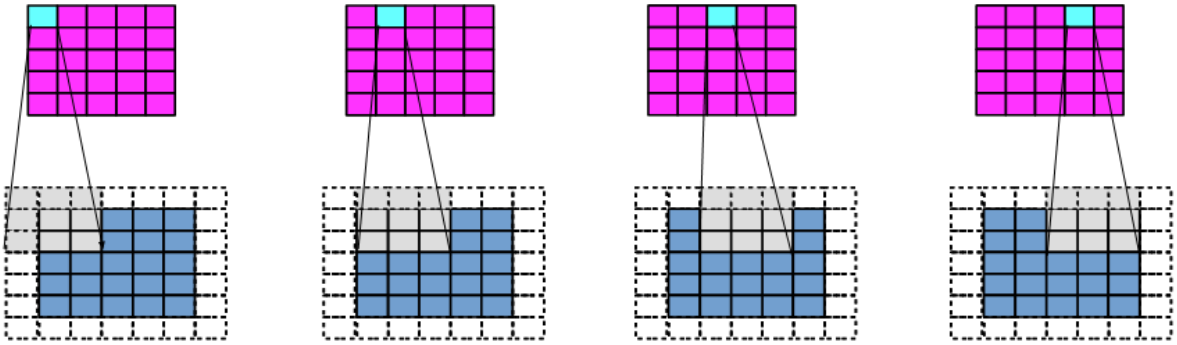


Figura 2.2.13: En esta Figura se ejemplifica el *same padding*. En este caso se tiene $k = 3$, $i = 5$. Nótese que al aplicar la relación 3 se obtiene $p = 1$.

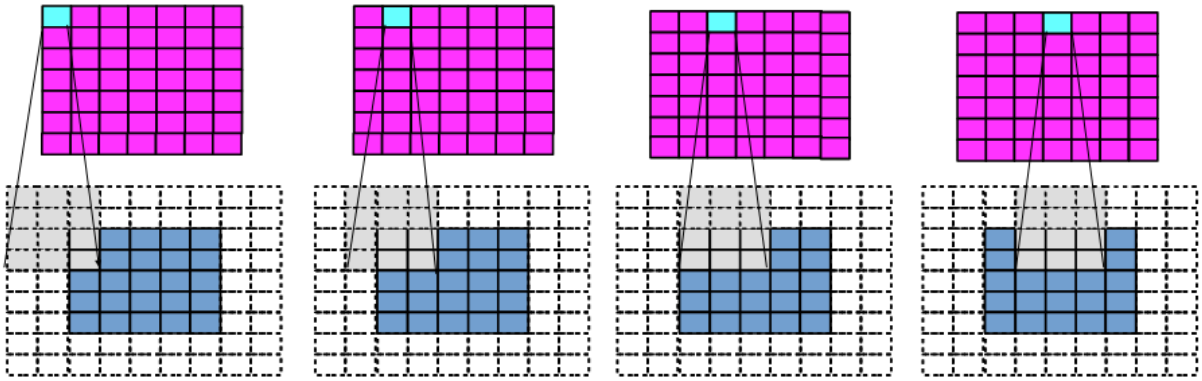


Figura 2.2.14: Ejemplo de *full padding*. En este caso se tiene $i = 5$, $k = 3$, $s = 1$ y $p = 2$.

Sin *padding* de ceros, *stride* unitario, transpuesta

Este caso es cuando se aplica una convolución transpuesta sin *padding* de ceros con *stride* unitario. Normalmente la convolución transpuesta tiene el efecto inverso de la convolución normal así que sería razonable pensar como la convolución transpuesta para recuperar el tamaño de cierto mapa de características.

Relación 7. *Para una convolución descrita por $s = 1$, $p = 0$ y cualquier k se tiene una convolución transpuesta asociada descrita por $k' = k$, $s' = s$ y $p' = k - 1$. Y su tamaño de salida es*

$$o' = i' + (k - 1)$$

Convolución dilatada general.

Usando las relaciones descritas previamente e incluyendo un factor de dilatación d en el kernel es posible establecer una relación para la convolución con dilatación como sigue

Relación 8. *Para cualquier i, k, p, s y un factor de dilatación d,*

$$o = \lfloor \frac{i + 2p - k - (k-1)(d-1)}{s} \rfloor + 1$$

2.3. Optimización: Descenso de gradiente

2.3.1. Función de Costo o *Loss*

Para que una red pueda aprender cierto patrón es necesario definir un objetivo. Es decir, hay que definir una función que dado la predicción devuelva que tan bien o mal es esa predicción. Esta función se conoce como **Loss** o costo. El error se calcula basado en esta función y este error se usa para actualizar los parámetros de la red. Una de las funciones **Loss** más conocidas es la *Softmax Cross-Entropy*, el cual es usada para problemas de clasificación. En general la función de *Loss* depende del problema a resolver. La función *Softmax Cross-Entropy* para un ejemplo de entrenamiento está definida por la ecuación 2.3.1 donde $z_i^{(L)}$ es la salida *Softmax* i-esima al final de la red o la predicción de esta misma. Una única salida *Softmax* $z_k^{(L)}$ esta definido por la ecuación 2.3.2.

$$C = \sum_{i=1}^{n} -y_i \log P(y_i = 1) = \sum_{i=1}^{n} -y_i \log(z_i^{(L)}) \tag{2.3.1}$$

$$z_k = \frac{\exp(s_k^{(L)})}{\sum_{j=1}^{n} \exp^{s_j^{(L)}}} \tag{2.3.2}$$

Otras funciones de costo muy importantes especialmente para usar en tareas de regresión son la función L_1 y L_2. La función L_2 también es conocida como mínimos cuadrados, y es usual encontrarla en tareas de ajustes de curva. La función L_2 la podemos ver en la ecuación 2.3.3, donde f es el modelo con los parámetros θ y y_i, $\mathbf{x}_i$ son la etiqueta y su ejemplo. La función L_1 también es conocida como diferencia absoluta y se puede ver en la ecuación 2.3.4. Un detalle técnico de la función L_1 es que

no es diferenciable en el punto $x = 0$. Muchas implementaciones simplemente toman como ese punto la derivada igual a cero.

$$L_2 = \sum_{i=1}^{n} (y_i - f(\theta; \mathbf{x}_i))^2 \tag{2.3.3}$$

$$L_1 = \sum_{i=1}^{n} |y_i - f(\theta; \mathbf{x}_i)| \tag{2.3.4}$$

Función de Regularización

Muchas veces la función de costo viene acompañada de un **regularizador**. La regularización es un término extra que se le agrega a la función de costo y técnicamente es otra función como combinación lineal de todos los parámetros del modelo multiplicados por un peso. Dicho así, dado una función de costo $C(\theta)$ y una función de regularización $R(\theta)$. La función de costo **total**, es decir, la que se tiene que minimizar sería $C_t = C + \lambda R$, donde λ es un hyper-parámetro más que tiene que ser encontrado empíricamente. Ahora bien, existen dos regularizadores muy utilizados en la actualidad los cuales son las mismas funciones L_1 y L_2. La diferencia es que son la suma de los parámetros del modelo, en las ecuaciones 2.3.5 y 2.3.6 se puede ver la forma general de una función de costo total con sus respectivos regularizadores.

$$C_t(\mathbf{x}, \mathbf{y}; \mathbf{w}) = C(\mathbf{x}, \mathbf{y}; \mathbf{w}) + \underbrace{\lambda \sum_i w_i^2}_{L_2} \tag{2.3.5}$$

$$C_t(\mathbf{x}, \mathbf{y}; \mathbf{w}) = C(\mathbf{x}, \mathbf{y}; \mathbf{w}) + \underbrace{\lambda \sum_i |w_i|}_{L_1} \tag{2.3.6}$$

2.3.2. Descenso de gradiente

Además de la función de *Loss*, para el aprendizaje es necesario un método de optimización para usar el error en la actualización de los pesos de la red. El método comúnmente usado es el descenso del gradiente. Este es un método iterativo que empieza con parámetros aleatorios en el modelo. Usando una función de costo $C(\theta)$ donde θ representa los parámetros del modelo (en este caso los pesos de la red) y se sabe que el gradiente de C con respecto a θ nos da la dirección de máximo incremento de $C(\theta)$ en un sentido lineal en el punto en que el gradiente de la función es evaluado. Así, para obtener la dirección de mínimo decremento se usa el negativo del gradiente.

La regla de actualización de los pesos θ en descenso del gradiente en un tiempo $(t+1)$ se presenta en la ecuación 2.3.7. Donde η representa la tasa de aprendizaje y $\theta^{(t)}$, $\theta^{(t+1)}$ son los parámetros del modelo en el tiempo t y $(t+1)$ respectivamente. Un problema de este método es que son modelos de *full-bactch*, lo que implica que para calcular el gradiente es necesario usar todo los datos de entrenamiento, y si el conjunto de datos es grande, entonces sería costoso en tiempo.

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)}) \tag{2.3.7}$$

El descenso de gradiente puede entenderse como tomar pequeños pasos y observar la siguiente posición con mayor descenso, una imagen ilustrativa de este método se encuentra en la Figura 2.3.1. Este ejemplo es una manera muy general de visualizarlo, sin embargo, en el caso de las redes neuronales la función a optimizar contiene miles o millones de parámetros lo que hace difícil tener un panorama de como visualizarlo. Una de las situaciones que hay que considerar es el tipo de función o costo que se requiere para encontrar el óptimo, muchas veces nos encontramos en una situación de varios mínimos locales, y este método usualmente puede caer en uno de ellos. Además el hyperparámetro principal de este método, es decir η, influye mucho en el comportamiento por cada iteración, un tamaño de paso muy grande fácilmente nos deja en una situación en el que la actualización de parámetros se queda estancada o dando saltos, si embargo el tamaño de paso es algo relativo, si la función es demasiado convexa entonces un paso muy grande podría ocasionar problemas, mientas que ese mismo tamaño de paso podría ser pequeño para otra función que no es tan convexa. Dicho lo anterior, encontrar el tamaño de paso adecuado para el problema es una tarea extensa y requiere de múltiples experimentos. Una ilustración de lo mencionado puede observarse en la Figura 2.3.2.

2.3.3. Descenso de gradiente estocástico

Para solucionar el problema del *Full-Batch* del descenso de gradiente se diseñó un algoritmo llamado descenso de gradiente estocástico (SGD por sus siglas en inglés), el cual es basado en las mismas reglas del descenso de gradiente pero utilizando solo unos cuantos puntos del conjunto de datos para actualizar los pesos. La dirección de descenso en cada paso no es basado en $C(\theta)$ sino en $C^{(i)}(\theta)$ el cual es la función de costo basado en el i-ésimo ejemplo de entrenamiento. Los parámetros θ del modelo son actualizados como normalmente se hace en descenso del gradiente y se hace varias pasadas sobre la base de datos hasta que se obtenga una convergencia deseada.

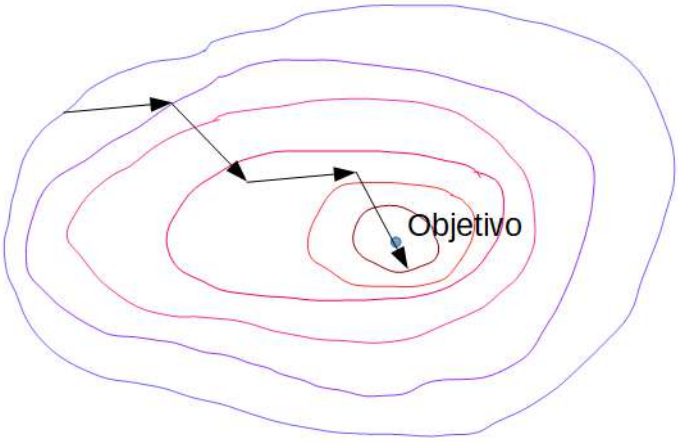


Figura 2.3.1: Descenso de gradiente (o GD por sus siglas en inglés) visualización simple. En este caso se supone una función que tiene dos parámetros. GD toma pasos de acuerdo al valor de gradiente en el punto en donde esta posicionado. Usualmente, luego de varias iteraciones se aproxima al óptimo, pero esta aproximación dependerá de la configuración de los hyperparámetros del método.

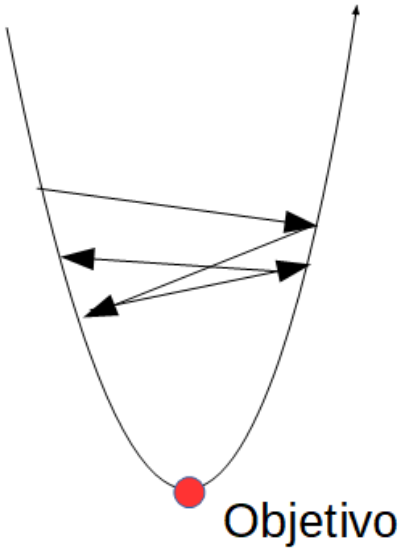


Figura 2.3.2: Visualización de GD cuando el tamaño de paso es demasiado grande o también el tamaño de paso es grande para una función con cierto grado de convexidad. Se observa como los pasos quedan atorados en un lugar y no es capaz de aproximarse más al óptimo.

Por lo general, descenso de gradiente estocástico tiende a converger más rápido, sin embargo, llega a ser más ruidoso porque la dirección de descenso en cada iteración

no es sobre toda la base de datos, es por eso que se recomienda que los puntos o datos para calcular el gradiente en cada iteración sean lo más aleatorios posible y los suficientes para que el gradiente sea lo suficientemente preciso.

2.3.4. ADAM

ADAM significa *Adaptive Moment Estimation* y dicho informalmente es una extensión del descenso de gradiente estocástico [23]. Los autores de este método de optimización claman que es más eficiente que otros y más estable. En la actualidad es uno de los más usados debido a que se encuentra implementado en muchas de las bibliotecas de **Machine Learning** y se han comprobado las características mencionadas anteriormente. Por ejemplo, el trabajo de [12] usa este método. ADAM define varias variables auxiliares para que pueda trabajar el algoritmo. La primera variable es β_1 y se conoce como decaimiento promedio del gradiente y la segunda variable es β_2 y sirve como decaimiento promedio del cuadrado del gradiente. Los valores de β_1 y β_2 deben encontrarse en el rango de $[0, 1)$ los valores típicos usados para estos son $0{,}9$ y $0{,}001$ respectivamente. El Algoritmo 2 explica de manera general como aplicar este procedimiento. Como podrá observarse este método desarrolla un mecanismo similar a ir cambiando el tamaño de paso en iteraciones más avanzadas. Algunos de los resultados de los autores se muestran en las figuras 2.3.3 y 2.3.4. Cabe mencionar que algunas de las características de ADAM vienen heredadas de los métodos predecesores de los cuales se muestran en la misma gráfica de los resultados.

Result: θ_t (pesos óptimos del modelo)
Input: $f(\theta)$ (función objetivo);
Input: θ_0 (pesos iniciales del modelo o vector de parámetros);
Inicializa α (tamaño de paso);
Inicializa $(\beta_1, \beta_2) \in [0, 1)$ (decaimientos exponenciales);
$\mathbf{m}_0 \leftarrow 0$ (Inicializa el vector del primer momento);
$\mathbf{v}_0 \leftarrow 0$ (Inicializa el vector del segundo momento);
$t \leftarrow 0$ (Inicializa el paso);
while θ_t *no converja* **do**
  $t \leftarrow t+1$;
  $\mathbf{g}_t \leftarrow \nabla_\theta f(\theta_{t-1})$ (obtén el gradiente con respecto la función objetivo en el paso t);
  $\mathbf{m}_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \mathbf{g}_t$;
  $\mathbf{v}_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t^2$;
  $\hat{\mathbf{m}}_\mathbf{t} \leftarrow \mathbf{m}_t / (1-\beta_1^t)$;
  $\hat{\mathbf{v}}_\mathbf{t} \leftarrow \mathbf{v}_t / (1-\beta_2^t)$;
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{\mathbf{m}}_\mathbf{t} / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$;
end

Algoritmo 2: Algoritmo de ADAM. ϵ solo es un valor muy pequeño para asegurar que no se haga cero el divisor, un valor efectivo es $10e-8$. La representación $\mathbf{g}^2$ representa la potencia a la 2 por cada elemento del vector. En general las operaciones con vectores son por elemento. La representación β_1^t y β_2^t representa potencia a la t. Y $f(\theta)$ representa la función objetivo en su forma estocástica.

2.3.5. Retropropagación en capas convolucionales

Usualmente es conveniente entender como se daría el algoritmo de *backpropagation* para actualizar los pesos de una red en capas convolucionales. Calcular el gradiente de una función es crucial en este procedimiento, en la actualidad muchas bibliotecas de *Machine Learning* usan un método muy novedoso para hacer este cálculo conocido como *Automatic Differentiation* (AD) [5] o también conocida como *Algorithmic Differentiation*. El uso de este método suele ocultar procedimientos tediosos para entrenar una modelo basado en descenso de gradiente, pero ocultar esto suele alejar al investigador de un conocimiento fundamental.

Siguiendo el enfoque presentado en [28] para entender como sería un ejemplo simple del cálculo de los gradientes para capas convolucionales, véase la Figura 2.3.5.

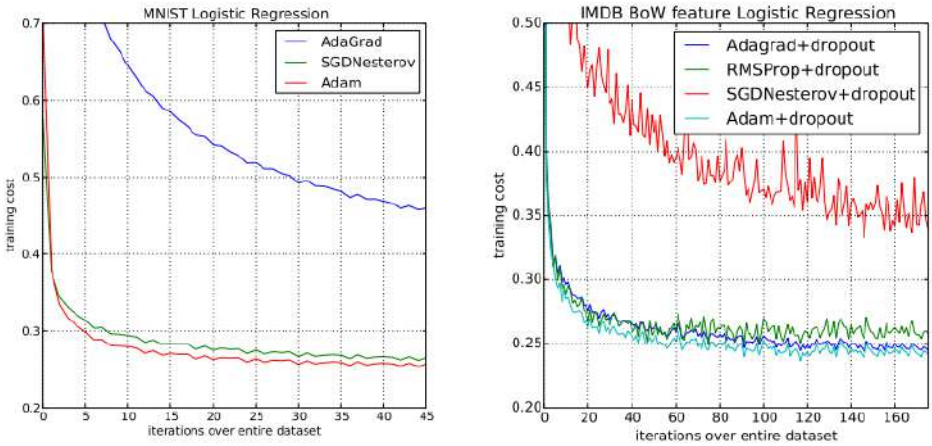


Figura 2.3.3: Resultados tomado de los autores de ADAM [23]. En la imagen de la izquierda vemos los resultados usando un modelo de regresión logística con la base de datos MNIST. Y en la izquierda la base de datos IMDB BoW (*Bag of Words*) los cuales son vectores de características. Se muestra también métodos anteriores como *AdaGrad*, *SGDNesterov* y RMSProp.

Los valores del mapa de características resultante se pueden denotar como sigue

$$s_{11} = w_{22}a_{11} + w_{21}a_{12} + w_{12}a_{21} + w_{11}a_{22}$$
$$s_{12} = w_{22}a_{12} + w_{21}a_{13} + w_{12}a_{22} + w_{11}a_{23}$$
$$s_{21} = w_{22}a_{21} + w_{21}a_{22} + w_{12}a_{31} + w_{11}a_{32}$$
$$s_{22} = w_{22}a_{22} + w_{21}a_{23} + w_{12}a_{32} + w_{11}a_{33}$$

o en un modo generalizado

$$s_{ij} = \sum_{n=1}^{2}\sum_{m=1}^{2} w_{(3-m)(3-n)}a_{(i-1+m)(j-1+n)}$$

Definamos el gradiente de la función de costo L con respecto a un valor del mapa resultante como sigue

$$\frac{\partial L}{\partial s_{ij}} = \delta_{ij}$$

Primero calcularemos el gradiente de la función de costo con respecto al peso w_{22}. El peso está asociado con todas las s_{ij} y por lo tanto debería contener un componente del gradiente de todas las δ_{ij}.

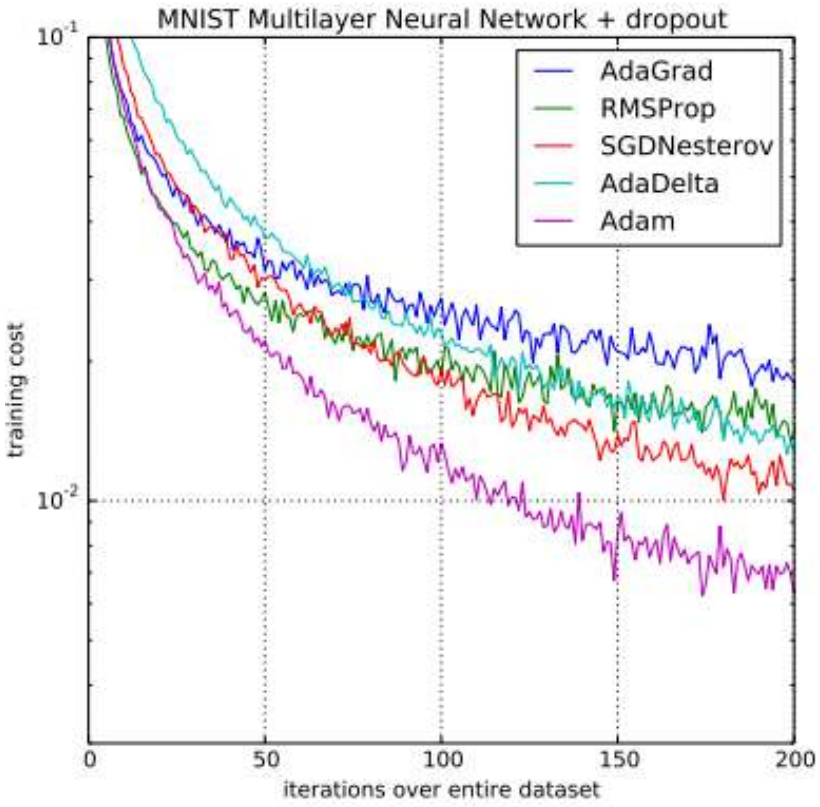


Figura 2.3.4: Resultados tomados de los autores de ADAM [23]. En la imagen se muestra un ejemplo de red neuronal (*multi layer perceptron*) y comparaciones con otros métodos de optimización. El modelo de red también cuenta con Dropout. Los resultados propuestos coinciden con las características que claman los autores sobre el método.

$$\frac{\partial L}{\partial w_{22}} = \sum_{j=1}^{2}\sum_{i=1}^{2} \frac{\partial L}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial w_{22}} = \delta_{ij} \frac{\partial s_{ij}}{\partial w_{22}}$$

Se puede derivar fácilmente las siguientes relaciones tomando los pesos que intervienen en la derivada.

$$\frac{\partial s_{11}}{\partial w_{22}} = a_{11}, \ \frac{\partial s_{12}}{\partial w_{22}} = a_{12}, \ \frac{\partial s_{21}}{\partial w_{22}} = a_{21}, \ \frac{\partial s_{22}}{\partial w_{22}} = a_{22}$$

y por lo tanto

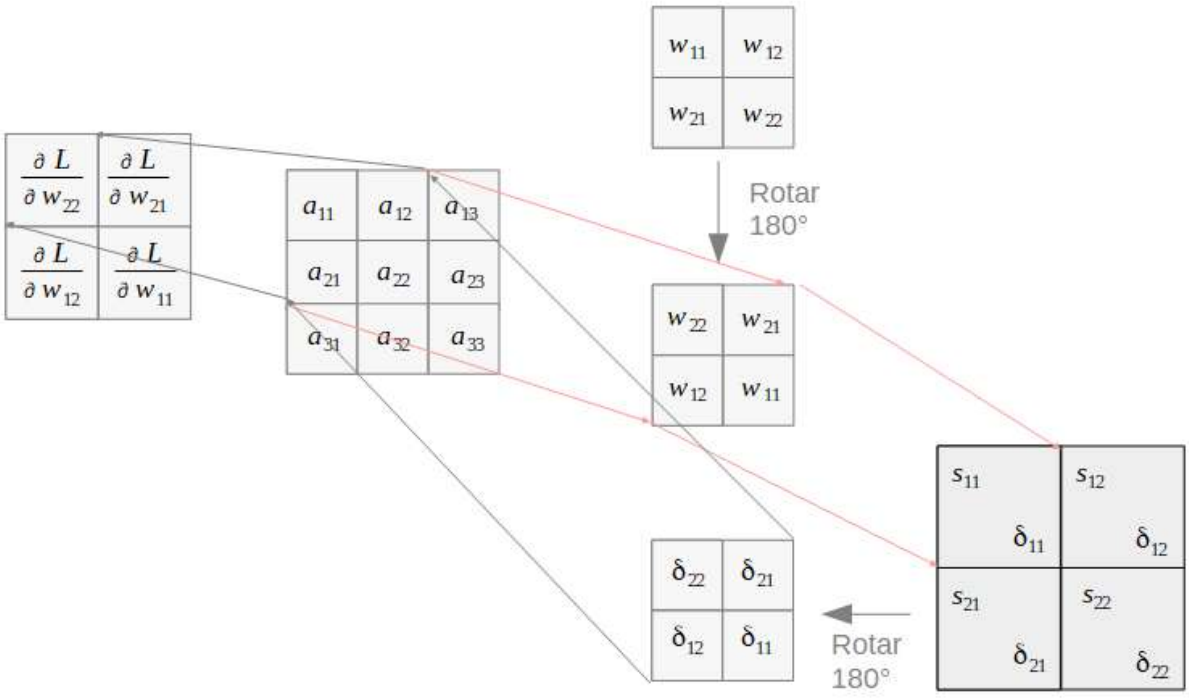


Figura 2.3.5: Ilustración ejemplo para *backpropagation* en capas convolucionales.

$$\frac{\partial L}{\partial w_{22}} = \delta_{11} a_{11} + \delta_{12} a_{12} + \delta_{21} a_{21} + \delta_{22} a_{22}$$

Se continua repitiendo este procedimiento para todos los demás pesos

$$\frac{\partial L}{\partial w_{21}} = \sum_{j=1}^{2} \sum_{i=1}^{2} \frac{\partial L}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial w_{21}} = \sum_{j=1}^{2} \sum_{i=1}^{2} \delta_{ij} \frac{\partial s_{ij}}{\partial w_{21}}$$

$$\frac{\partial s_{11}}{\partial w_{21}} = a_{12}, \ \frac{\partial s_{12}}{\partial w_{21}} = a_{13}, \ \frac{\partial s_{21}}{\partial w_{21}} = a_{22}, \ \frac{\partial s_{22}}{\partial w_{21}} = a_{23}$$

$$\frac{\partial L}{\partial w_{21}} = \delta_{11} a_{12} + \delta_{12} a_{13} + \delta_{21} a_{22} + \delta_{22} a_{23}$$

$$\frac{\partial L}{\partial w_{11}} = \sum_{j=1}^{2}\sum_{i=1}^{2}\frac{\partial L}{\partial s_{ij}}\frac{\partial s_{ij}}{\partial w_{11}} = \sum_{j=1}^{2}\sum_{i=1}^{2}\delta_{ij}\frac{\partial s_{ij}}{\partial w_{11}}$$

$$\frac{\partial s_{11}}{\partial w_{11}} = a_{22},\ \frac{\partial s_{12}}{\partial w_{11}} = a_{23},\ \frac{\partial s_{21}}{\partial w_{11}} = a_{32},\ \frac{\partial s_{22}}{\partial w_{11}} = a_{33}$$

$$\frac{\partial L}{\partial w_{11}} = \delta_{11}a_{22} + \delta_{12}a_{23} + \delta_{21}a_{32} + \delta_{22}a_{33}$$

$$\frac{\partial L}{\partial w_{12}} = \sum_{j=1}^{2}\sum_{i=1}^{2}\frac{\partial L}{\partial s_{ij}}\frac{\partial s_{ij}}{\partial w_{12}} = \sum_{j=1}^{2}\sum_{i=1}^{2}\delta_{ij}\frac{\partial s_{ij}}{\partial w_{12}}$$

$$\frac{\partial s_{11}}{\partial w_{12}} = a_{21},\ \frac{\partial s_{12}}{\partial w_{12}} = a_{22},\ \frac{\partial s_{21}}{\partial w_{12}} = a_{31},\ \frac{\partial s_{22}}{\partial w_{12}} = a_{32}$$

$$\frac{\partial L}{\partial w_{11}} = \delta_{11}a_{21} + \delta_{12}a_{22} + \delta_{21}a_{31} + \delta_{22}a_{32}$$

En un modo más generalizado cada derivada parcial definida anteriormente se puede definir como sigue

$$\frac{\partial L}{\partial w_{ij}} = \sum_{n=1}^{2}\sum_{m=1}^{2}\delta_{mn}a_{(i-1+m)(j-1+n)}$$

y arreglando la ecuación anterior en una versión matricial quedaría como sigue

$$\begin{bmatrix} \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{21}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{11}} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} \delta_{11} & \delta_{21} \\ \delta_{12} & \delta_{22} \end{bmatrix}$$

donde el símbolo $\oplus$ significa la correlación cruzada. Esto es equivalente a la convolución con el kernel rotado 180°:

$$\begin{bmatrix} \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{21}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{11}} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} \delta_{22} & \delta_{12} \\ \delta_{21} & \delta_{11} \end{bmatrix}$$

2.3.6. Diferenciación Automática

Los problemas comunes de optimización se basan fundamentalmente en el cálculo de gradientes, en especifico el descenso del gradiente. Cuando un problema es relativamente sencillo se tienen varias rutas para calcular el gradiente las cuales son: algebraicamente y aproximaciones numéricas. Las redes neuronales profundas son extremadamente complejas para resolver los gradientes tanto algebraicamente como numéricamente. Con el objetivo de sobre llevar esto, se diseñó un método de diferenciación novedoso, el cual es *Diferenciación Automática* (AD por sus siglas en inglés), también suele llamarse diferenciación algorítmica. Este método es el núcleo de funcionamiento de algunas bibliotecas como TensorFlow y Torch. AD no tiene ningún parecido con los métodos convencionales y su metodología esta fuertemente basada en la regla de la cadena de cálculo. Para hacer uso de este método generalmente se define una abstracción del cálculo completo del problema el cual se conoce como grafo de cómputo. Y como dice el nombre, representamos todas las operaciones en un grafo. AD en conjunto con la regla de la cadena se basa en la hipótesis de que cualquier derivada de una función compleja puede ser derivada a partir de múltiples descomposiciones hasta llegar a funciones cuyas derivadas son conocidas (como la suma, multiplicación y funciones trigonométricas). Para entender esto vamos a tomar el ejemplo descrito en [5] y representemos una función 2.3.8 en un grafo de cómputo 2.3.6 donde se han añadido un conjunto de variables intermedias entre los nodos del grafo, estás variables se describirán posteriormente pero es trivial la asignación de cada una. AD trabaja en dos modos principalmente.

$$y = f(x_1, x_2) = ln(x_1) + x_1 x_2 + \sin(x_2) \tag{2.3.8}$$

Forward Mode

Acumulación hacia adelante (*forward acummulation mode*) es la manera más simple de este método. Retomemos la función ejemplo 2.3.8 para calcular la derivada de

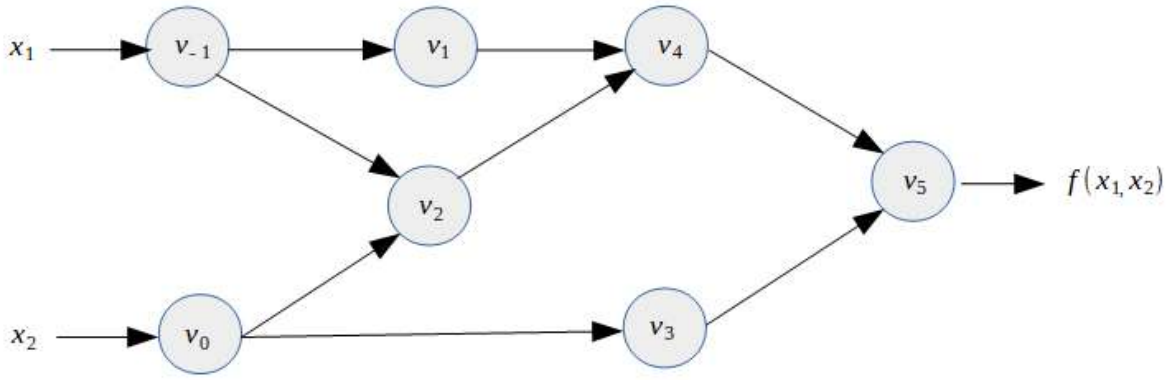


Figura 2.3.6: Grafo de cómputo de la ecuación 2.3.8.

f con respecto a x_1, primero se asocia cada variable intermediara con su derivada

$$\dot{v}_1 = \frac{\partial v_i}{\partial x_1}$$

Aplicando la regla de la cadena en cada operación elemental del trazo primario generamos su correspondiente trazo de derivada. En la Tabla 2.3.1 se observa el trazo completo para calcular la derivada de f con respecto a x_1. AD en funciones del tipo $\mathbb{R}^n \to \mathbb{R}$ requiere n evaluaciones para calcular el gradiente.

Trazo primario hacia adelante	Trazo de derivadas hacia adelante
$v_{-1} = x_1 = 2$	$\dot{v}_{-1} = \dot{x}_1 = 1$
$v_0 = x_2 = 5$	$\dot{v}_0 = \dot{x}_2 = 0$
$v_1 = \ln(v_{-1}) = \ln(2)$	$\dot{v}_1 = \dot{v}_{-1} / v_{-1} = 1/2$
$v_1 = v_{-1} v_0 = 2 * 5$	$\dot{v}_2 = \dot{v}_1 v_0 + \dot{v}_0 v_{-1} = 1 * 5 + 0 * 2$
$v_3 = \sin(v_0) = \sin(5)$	$\dot{v}_3 = \dot{v}_0 \cos(v_0) = 0 * \cos(5)$
$v_4 = v_1 + v_2 = 0{,}693 + 10$	$\dot{v}_4 = \dot{v}_1 + \dot{v}_2 = 0{,}5 + 5$
$v_5 = v_4 - v_3 = 10{,}693 + 0{,}959$	$\dot{v}_5 = \dot{v}_4 - \dot{v}_3 = 5{,}5 - 0$
$y = v_5 = 11{,}652$	$\dot{y} = 5{,}5$

Tabla 2.3.1: *Forward mode* para calcular la derivada con respecto a x_1 evaluado en el punto $(2, 5)$.

Reverse Mode

AD en acumulación hacia atrás corresponde a un modo generalizado de la propagación hacia atrás, en el que se propagan las derivadas hacia atrás dado una entrada. Esto se realiza complementando cada variable intermediaria v_i con un adjunto

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i}$$

el cual representa la sensitividad de cierta salida y_j con respecto a los cambios en v_i. Este modo es llevado a cabo en dos fases, en la primera fase se ejecuta en modo hacia adelante para incializar las variables intermediarias v_i y grabando las dependencias en el grafo de cómputo. En la segunda fase las derivadas son calculadas propagando los adjuntos $\bar{v}_i$ en reversa de las salidas hacia las entradas. En la Tabla 2.3.2 se observa el paso en reversa del ejemplo 2.3.6. En dicha Tabla tómese como ejemplo v_0, en la Figura 2.3.6 vemos que la única manera de afectar y es a través de v_2 y v_3 así sus contribuciones para alterar y son dados por

$$\bar{v0} = \bar{v}_2 \frac{\partial v_2}{\partial v_0} + \bar{v}_3 \frac{\partial v_3}{\partial v_0}$$

En la Tabla 2.3.2 esta contribución es calculado en dos pasos incrementales

$$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} \text{ y } \bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$$

razón por la cual en algunas filas de la Tabla se observan dos operaciones adjuntas.

Trazo primario hacia adelante	Trazo de derivadas adjuntas hacia atrás
$v_{-1} = x_1 = 2$	$\bar{x}_1 = \bar{v}_{-1} = 5{,}5$
$v_0 = x_2 = 5$	$\bar{x}_2 = \bar{v}_0 = 1{,}716$
$v_1 = \ln(v_{-1}) = \ln(2)$	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}} = \bar{v}_{-1} + \bar{v}_1 / v_{-1} = 5{,}5$
$v_1 = v_{-1} v_0 = 2 * 5$	$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_0 + \bar{v}_2 * v_{-1} = 1{,}716$
	$\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}} = \bar{v}_2 * v_0 = 5$
$v_3 = \sin(v_0) = \sin(5)$	$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} = \bar{v}_3 * \cos v_0 = -0{,}284$
$v_4 = v_1 + v_2 = 0{,}693 + 10$	$\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2} = \bar{v}_4 * 1 = 1$
	$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_4 * 1 = 1$
$v_5 = v_4 - v_3 = 10{,}693 + 0{,}959$	$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 * (-1) = -1$
	$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4} = \bar{v}_5 * 1 = 1$
$y = v_5 = 11{,}652$	$\bar{v}_5 = \bar{y} = 1$

Tabla 2.3.2: *Reverse mode* para calcular la derivada con respecto a x_1 evaluado en el punto $(2,5)$. Después de hacer el paso hacia adelante (del lado izquierdo) las operaciones conjuntas a la izquierda son evaluadas en reversa. Las operaciones $\partial y/\partial x_1$ y $\partial y/\partial x_2$ son calculadas en el mismo paso en reversa, empezando con el adjunto $\bar{v}_5 = \bar{y} = \partial y/\partial y = 1$.

2.4. Diseños comunes de redes

2.4.1. Redes con salida totalmente conectada

Estas redes son comunes en la tarea de clasificación. Se llaman así debido a que las primeras capas de la red son etapas de convolución pero las últimas pueden transformarse en un perceptrón multi-capa normal mediante una operación de redimensionado de un tensor para dejarlo en un tensor de rango 1 (es decir un vector). Son reconocidas por los trabajos vistos en [24] y [25]. En la Figura 2.4.1 se ilustra un ejemplo simple.

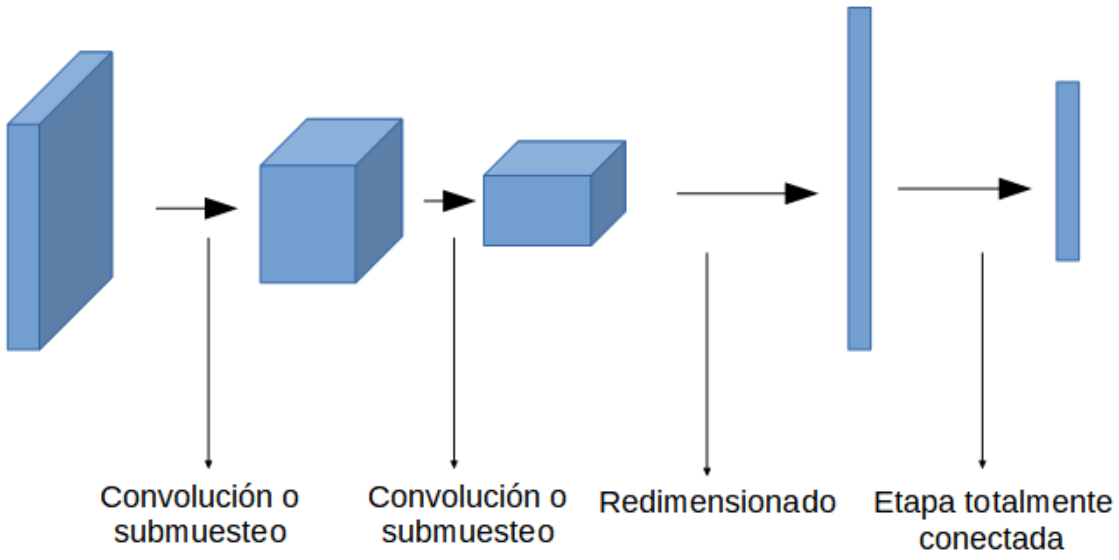


Figura 2.4.1: Figura de ejemplo de red con salida totalmente conectada. Este es un ejemplo simple, pero el número de capas de convolución así como la etapa en donde se aplica el redimensionado depende del autor. En estas redes es importante tener en cuenta el tamaño de los ejemplos. Ya que el modelo de red solo funcionará con esa dimensión.

2.4.2. Redes totalmente convolucionales

Estas redes, como su nombre lo indica, son construidas puramente con operaciones de convolución y derivados (como submuestreo). Su salida final de estas redes es igualmente un mapa de características o una imagen que puede tener uno o varios canales dependiendo de la tarea o problema. Al contener solo etapas de convolución tienen la peculiaridad de que pueden aceptar cualquier dimensión de entrada (con la

dimensión nos referimos al ancho y alto de la entrada) y no hay que redefinir y entrenar otro modelo para hacer inferencia con entradas de otra dimensión. Es decir se pueden usar los mismos pesos pre-entrenados para construir un modelo de inferencia que reciba dimensiones distintas a las que fue entrenado. Sin embargo, es claro que la dimensión de salida dependerá de la entrada. En la Figura 2.4.2 se tiene una ilustración ejemplo de esta estructura. A veces denotaremos estas redes como FCNN por sus siglas en inglés (*Fully Convolutional Neural Network*)

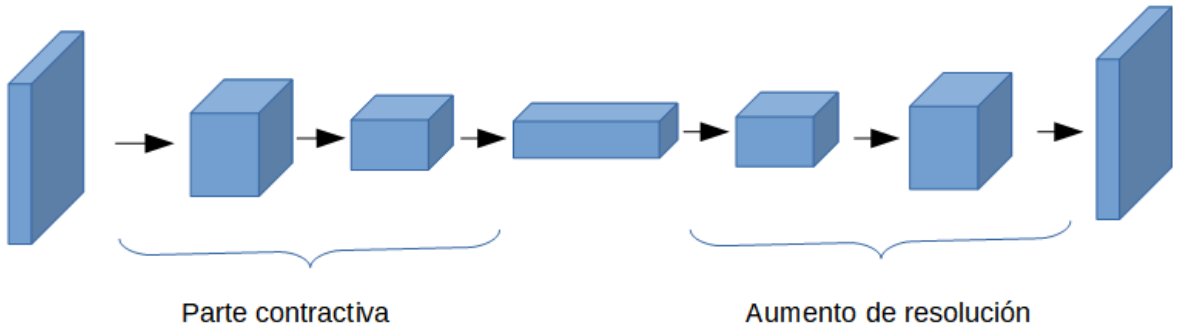


Figura 2.4.2: Ilustración ejemplo de estructura de una red *Fully Convolutional*. En la etapa contractiva se aplican cualquier operación de reducción como la convolución o submuestréo. En la parte de aumento de resolución se aplican operaciones contrarias como convolución transpuesta. Este es un ejemplo simple, las estructuras pueden tener más variabilidad, pero el punto es que no tienen ninguna etapa totalmente conectada.

2.4.3. Redes con bloques residuales.

Este tipo de topología puede ser encontrado en el trabajo de ResNet [18]. Después de cada serie de convolución con cierta activación la entrada de la operación es retroalimentada con la salida de la operación. En los modelos típicos las operaciones se hacen secuencialmente, se trata de ajustar el último mapa de características para arrojar la predicción, mientras que en este caso como la ResNet, trata de aprender con base en los mapas residuales y no un mapéo directo de la entrada a la salida. Este concepto es basado en la hipótesis de que es más fácil ajustar un mapa residual que el mapéo directo original. En la Figura 2.4.3 se puede observar un bloque residual.

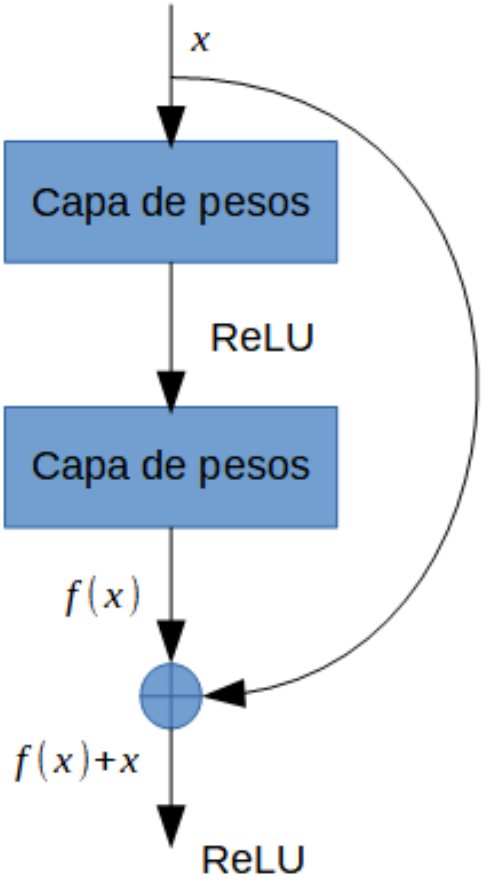


Figura 2.4.3: Figura de bloque residual.

2.5. Flujo Óptico

El flujo óptico para ser definido requiere un conjunto de *frames* consecutivos capturados por una cámara o dispositivo óptico. Es decir, ahora una imagen será definida mediante una función $I(x, y, t)$, $I : \mathbb{Z}^3 \rightarrow \mathbb{R}^C$, donde C es el número de canales en la imagen, (x, y) es la posición de un pixel en el tiempo t.

En un escenario que es capturado consecutivamente mediante una cámara y un objeto moviéndose en la escena, uno esperaría que los valores de pixeles que corresponden al objeto simplemente se desplacen conservando sus mismos valores. Dado dos imágenes y un tiempo definido t', $I(x, y, t')$ e $I(x, y, t' + 1)$, el objetivo fundamental del flujo óptico es encontrar una función que relacione pares de pixeles de los $frames$ consecutivos dado sus desplazamientos en x e y. Es decir, dado un pixel $p = (x^*, y^*)$ de la imagen en el tiempo t' debe existir un desplazamiento $d = (\Delta x, \Delta y)$ tal

que $I(x^*, y^*, t') = I(x^* + \Delta x, y^* + \Delta y, t' + 1)$. El objetivo del flujo óptico es encontrar los desplazamientos para cada pixel en cada tiempo t y $t+1$. Para que lo anterior se cumpla es necesario asumir la **condición de constancia de brillo** la cual indica que: el valor de dos pixeles correspondientes a dos *frames* consecutivos es el mismo. La constancia de brillo no sucede como se esperaría puesto que los dispositivos que capturan la luz son propensos a ruido y pese a que la escena sea controlada esto casi nunca va a pasar. Por tanto el diseño de métodos que capturen el flujo óptico es un problema retador en visión computacional. En la Figura 2.5.1. El flujo óptico también puede entenderse, de manera análoga, como la velocidad de desplazamiento de los pixeles. Definiendo $I(x, y, t)$ como la intensidad de un pixel en la posición (x, y) y $(u(x, y, t), v(x, y, t))$ como el flujo de ese pixel la ecuación de constancia de brillo esta descrita por 2.5.1. Como se describe en [4], si se lineariza la ecuación 2.5.1 usando la expansión de series de Taylor nos da la aproximación descrita por 2.5.2. Con lo anterior se tiene la ecuación que define la restricción del flujo óptico descrita en 2.5.3.

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.5.1}$$

$$I(x, y, t) \approx I(x, y, t) + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} \tag{2.5.2}$$

$$u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0 \tag{2.5.3}$$

El problema de apertura

Es usual que el cálculo del flujo óptico sea hasta la fecha un problema complejo. Esto es debido a que estimar el flujo óptico solo con imágenes es complicado, una de las mayores desventajas es la perdida de la percepción de profundidad, puesto que una imagen solo es una representación plana de la escena. Sin embargo existe un problema aún mayor, este problema se le conoce como **problema de apertura**. Para esto demos un ejemplo, considere que esta dentro de una cabina el cual solo tiene vista hacia el exterior en una ventana cuadrada muy pequeña, en la ventana logra apreciar que esta viendo la carga o remolque de un camión, pero no logra distinguir más sobre ello, además que la textura del remolque es bastante suave ¿podría afirmar si el camión se esta moviendo y hacia que dirección? Es evidente, con la información que se le proporcionó al sistema visual no se puede llegar a esa conclusión, esto es debido a que la apertura de la ventana es tan pequeña que no deja apreciar características

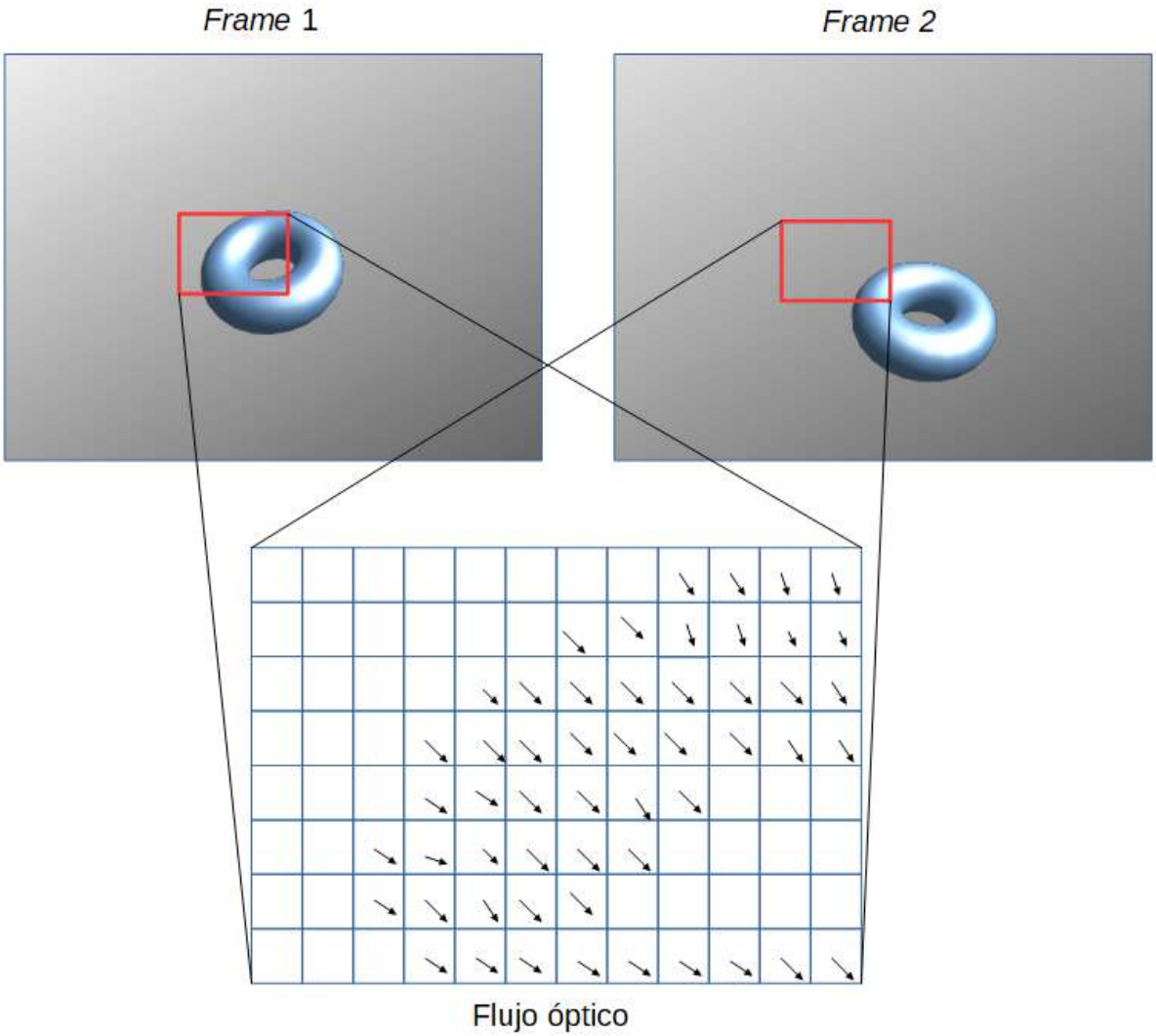


Figura 2.5.1: Ilustración del flujo óptico como un campo vectorial de desplazamientos de los pixeles.

clave para afirmar la dirección de movimiento. Nótese, que se mencionó ”apertura” de la ventana, puesto que es de ahí de donde viene el nombre. La apertura de la ventana o el cuadro donde se proyecta la información visual debe ser lo suficientemente extensa para dejar entrar puntos visuales clave que puedan ayudar a calcular la dirección de los vectores de movimiento. Véase la Figura 2.5.2 como ejemplo. El problema de apertura hasta la actualidad sigue siendo un tema importante y continúa siendo uno de los principales inconvenientes.

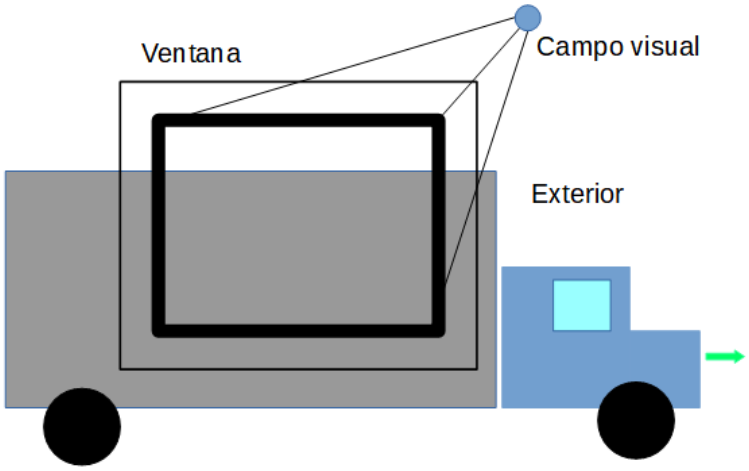


Figura 2.5.2: Ejemplo del problema de apertura. La ventana representa el espacio visual que se puede observar al exterior. Como se observa, con solo la información percibida de la ventana es difícil afirmar si el camión se está moviendo.

2.5.1. Codificación y visualización del flujo óptico

Tanto para almacenar y representar el flujo óptico es necesario guardar los dos componentes del vector de movimiento de los pixeles. Esto generalmente se hace mediante una imagen de dos canales con datos de punto flotante, el primer canal contiene los desplazamientos x y el segundo canal los desplazamientos y que en [4] suelen referirse como (u, v). Estos archivos tienen la extensión ".flo" y de acuerdo a la página oficial de Middlebury deben contener las siguientes características:

- El archivo debe ser una imagen de 2 canales que almacena valores de punto flotante codificados en orden little-endian [1].
- Un valor de flujo es considerado *unknow* (o invalido) si se cumple $|u|$ o $|v|$ son mayores que $1e9$.
- Los bytes del 0-3 al encabezado del archivo deben contener la etiqueta "PIEH" en ASCII. Que en *little-endian* sucede cuando es el flotante $202021{,}25$. Solo como una medida de verificación de que los flotantes están representados correctamente. Los bytes del 4-7 contiene el ancho, los bytes del 8-11 contiene el alto (de la imagen). Los demás bytes contienen la información del flujo (ancho*alto*2*4 bytes en total). Los valores de u y v son intercalados y ordenados por fila. Es de-

[1] Little-endian es una manera de ordenar los bits del menos al más significativo

cir, debe seguir el patrón $u[row_0, col_0]$, $v[row_0, col_0]$, $u[row_0, col_1]$, $v[row_0, col_1]$, ...

Para visualizar el flujo óptico de los archivos ".flo" se diseñó un estándar para representar la magnitud y dirección de manera intuitiva mediante una paleta de colores. Como se observa en la Figura 2.5.3 se usa ese diseño de paleta y es intuitivo que el color nos dice la dirección del vector y la intensidad del color nos dice la magnitud. De este modo un ejemplo de imagen de flujo óptico debería verse como en la Figura 2.5.4.

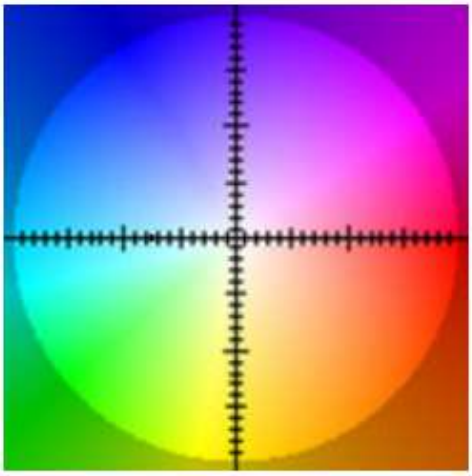


Figura 2.5.3: Código de colores para representar la magnitud y dirección de los vectores.

Figura 2.5.4: Ejemplo de aplicación de la codificación de la paleta de colores para representar el flujo óptico tomado de [4]. A la izquierda se encuentra el primer *frame*, al centro el segundo *frame* y a la derecha la imagen resultante al visualizar el campo vectorial.

2.5.2. Métrica para evaluación del flujo óptico

De acuerdo a [4] existe una métrica para medir la precisión de un método para inferir el flujo óptico el cual se conoce como *Average-Enpoint-Error* 2.5.4. Y técnicamente

es la media de la magnitud de los vectores de errores, los vectores de errores son la diferencia del vector predicho con el vector verdadero ($\mathbf{v}_{predic} - \mathbf{v}_{real}$). Esta función también puede ser usada como el costo para minimizar pero no necesariamente.

$$\| \mathbf{E} \|_1 = \sum_{x,y} |E_{x,y}| \approx \sum_{x,y} \sqrt{\| E_{x,y} \|^2 + \epsilon^2} \tag{2.5.4}$$

Capítulo 3

Metodología

Como se ha visto en la sección de antecedes, existen CNNs recientes que estiman el flujo óptico. Sin embargo, su uso y entendimiento puede parecer complejo debido a que se usan diversas técnicas de manipulación de información para mantener la fidelidad de estos, como son los mapeos de pixeles y transformaciones. Una cuestión sería si es posible diseñar un modelo de red que cumpla los mínimos requerimientos para dar una buena inferencia, sin usar demasiadas capas o procesamiento.

Desde un enfoque generalizado la metodología para este trabajo esta dividido en cuatro pasos principales:

1. Construir un entorno de trabajo con Python y Tensorflow para desarrollar, entrenar y probar estas redes.
2. Obtener modelos de red neuronal convolucional, partiendo de modelos sencillos.
3. Hacer experimentación con las redes propuestas.
4. Probar los rendimientos de las redes generadas. Y contrastarlas con el estado del arte.

3.1. Hipótesis del campo receptivo

Retomando el estado del arte y la definición del problema, se argumentó que un factor importante es el incremento del campo receptivo de una característica. Por lógica un humano normalmente para saber si algo se esta moviendo se tiene que observar alrededor del objeto. Similarmente se debería hacer en un modelo de estos. Inclusive

en los modelos que no son basados en redes neuronales de algún modo se tiene que observar la vecindad de un pixel para saber a donde se esta moviendo. Este argumento nos ha llevado a decidir que un modelo de red neuronal que incremente el campo receptivo de una característica de alto nivel nos llevará a dar buenas inferencias. Una manera intuitiva que uno podría tener para incrementar el campo receptivo es que la característica de un pixel involucre hacer inferencia sobre todos los demás pixeles de las imágenes proporcionadas. Lo dicho anteriormente es similar a decir que se haga una red neuronal cuya entrada sea las dos imágenes I_1 e I_2 solo para hacer regresión en el flujo de un solo pixel. Esto evidentemente sería un costo de cómputo enorme, si se tienen una imagen solo de 20 (ancho y alto) se tendrían que hacer 400 inferencias por la red solo para generar un mapa de flujo óptico de un ejemplo. Por tanto no podemos hacer esto, la manera de proceder es generar una *Fully Convolutional Neural Network* en la que las características de la capa de salida involucra el suficiente campo receptivo para hacer predicción.

En CNN's existen 4 maneras para incrementar el campo receptivo de una característica de alto nivel:

- Incrementar el tamaño del filtro. Es razonable pensar que si queremos que cierta característica englobe muchos pixeles vecinos necesitaríamos un filtro grande, pero esto puede traer consecuencias en el número de parámetros así como el procesamiento requerido.
- Aumentar el *stride* de la convolución. Esto implica que el filtro de saltos más largos a través de la entrada durante el proceso de convolución. Si el *stride* es grande en comparación al tamaño del filtro esto ya no se cumplirá.
- Hacer que la CNN sea demasiado profunda. Incrementar las capas de la red implica que en cierta profundidad una característica tendrá la suficiente información para englobar los pixeles necesarios para su salida. Sin embargo, al igual que incrementar el tamaño del filtro, involucra aumentar el número de parámetros de la red.
- Usar convoluciones con dilatación. Como ejemplo véase la Figura 3.1.1.

En la convolución dilatada podemos ver que el campo receptivo incrementa pero algunas características del vecindario no son incluidas, además que la información se reduce aún más. Por tanto, se debe considerar una buena combinación de estas convoluciones con dilatación, y sería el análogo a las pirámides de resolución.

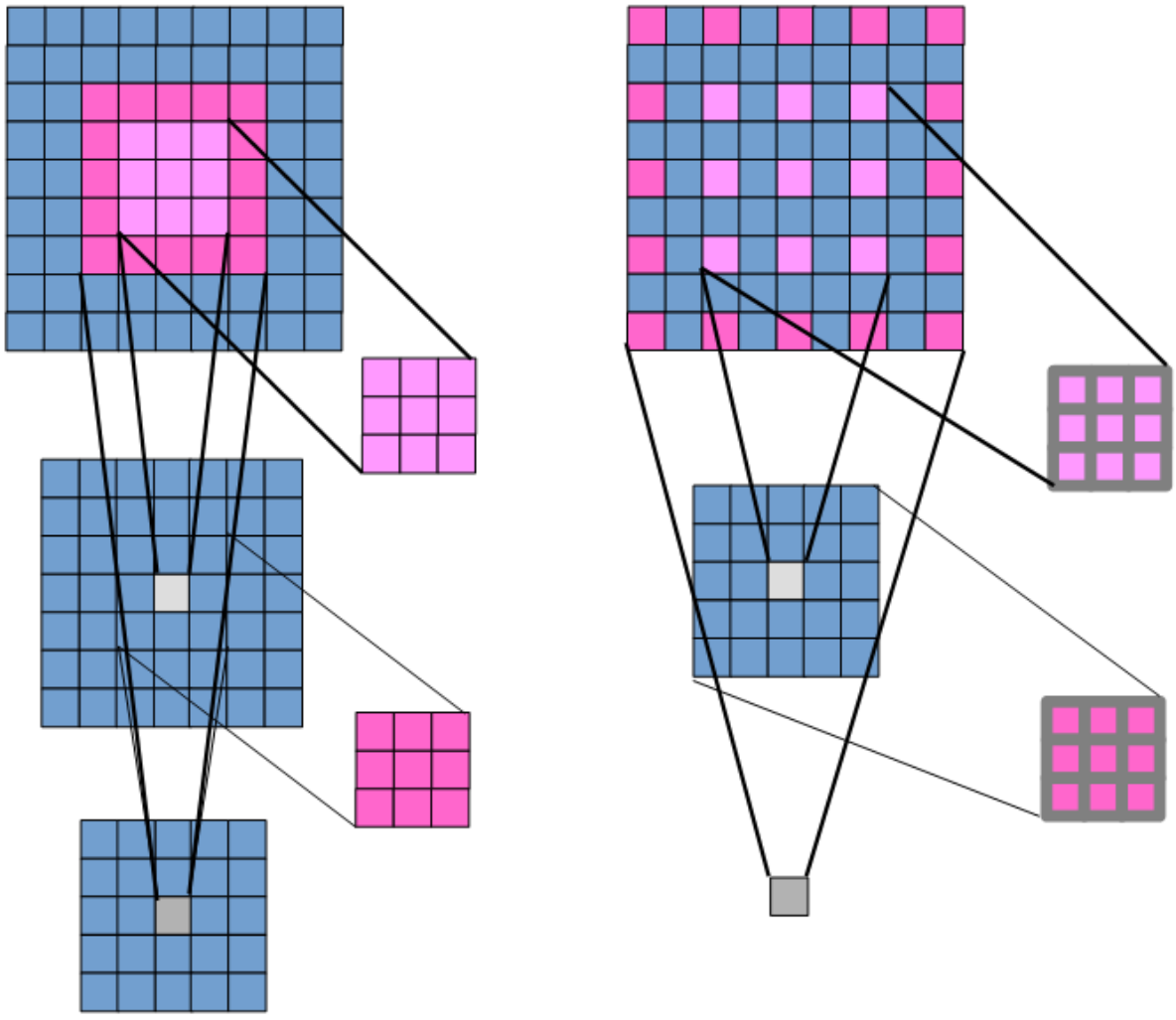


Figura 3.1.1: Representación del campo receptivo a través de la convoluciones de una característica de alto nivel. Del lado izquierdo tenemos una convolución estándar aplicando dos etapas con filtros de 3×3 (los cuales están a un lado), representamos el lugar que abarca el filtro con lineas y el color representa desde la entrada que campo receptivo le corresponde a la característica de acuerdo a la convolución que se le aplicó. Del mismo modo, del lado derecho dos etapas de convolución con factor de dilatación $d = 2$, como podemos observar el campo es más amplio, pero no involucra a todos los vecinos de cierto pixel de entrada, sin embargo, esta información puede llegar a ser útil en etapas posteriores de la red.

3.2. Modelos preliminares

Un paso inicial para probar la complejidad del problema de estimación del flujo óptico es tener un modelo no tan complejo como base, la cual debe ser una red totalmente

convolucional. Además que el problema es de regresión, se debe tener en consideración esto con el diseño de red. Por lo que las etapa final y algunas de las capas finales de la red no tienen función de activación. Con lo dicho anteriormente, el primer modelo puesto a prueba tiene la topología vista en la Figura 3.2.1. Como se observa la red consta de una etapa de convolución en la que se aplica dilatación con $D = 0$, $D = 1$ y $D = 2$. La idea de hacer esto es que la dilatación extiende el campo receptivo, por lo que puede aprender patrones en escalas espaciales mayores. Sin embargo, esto también se podría hacer aplicando un filtro más extenso, pero se tendría mucho más parámetros que con la convolución dilatada. Las etapas subsecuentes contienen una parte de contracción y otra de expansión usando convolución y convolución transpuesta.

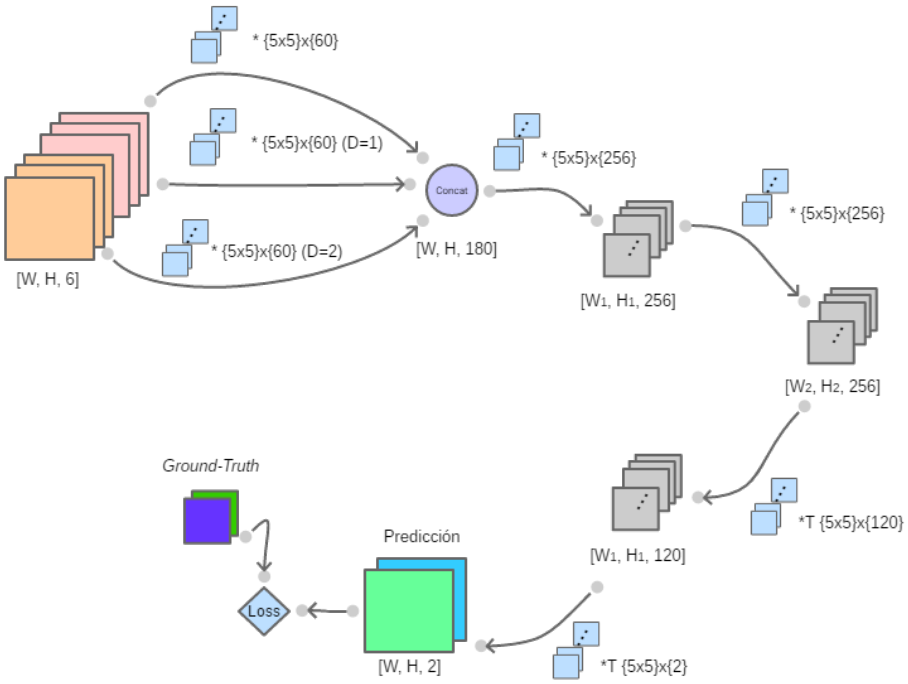


Figura 3.2.1: Diagrama del modelo 1 para hacer pruebas de complejidad del problema. Como se observa este modelo es bastante sencillo. Las flechas con ” * ” representan convolución y ” *T ” convolución transpuesta. El identificador D representa la cantidad de dilatación del filtro. A un lado de cada operación de convolución se presenta la información correspondiente a esa operación.

Otro modelo similar al anterior puede ser propuesto fácilmente solamente añadien-

do más capas de convolución y su respectiva expansión usando convolución transpuesta. Esto se muestra en la Figura 3.2.2.

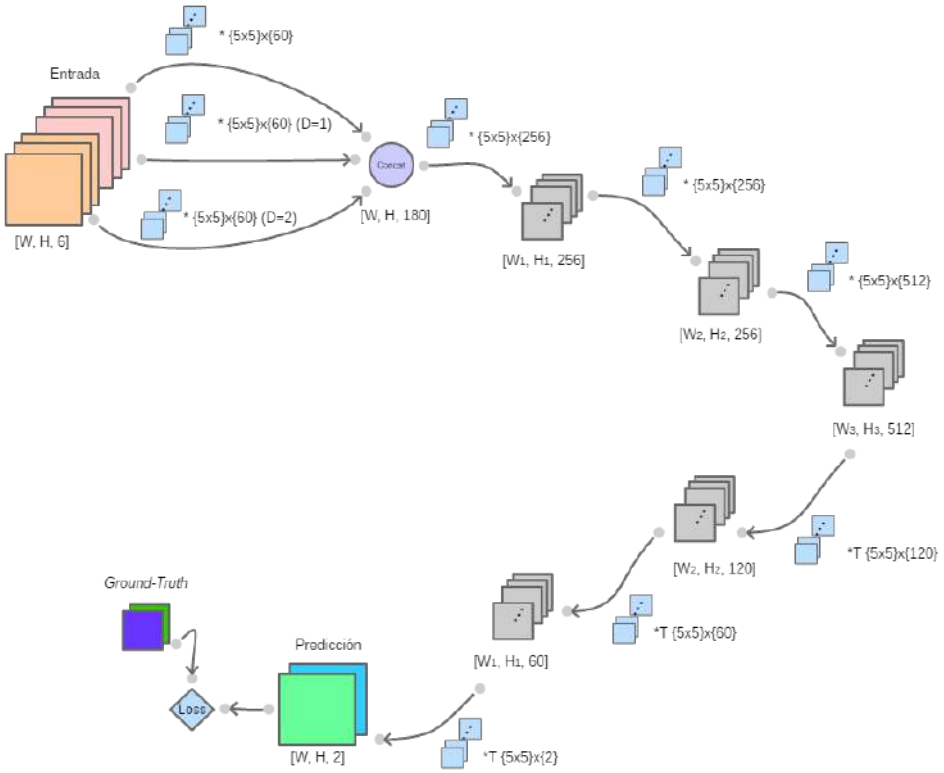


Figura 3.2.2: Diagrama del modelo 2 para hacer pruebas de complejidad del problema. Como se observa este es similar al modelo 1 (3.2.1), solo para ejemplificar que se puede expandir usando más capas.

Es posible que usando los modelos 1 y 2 (3.2.1 y 3.2.2 respectivamente) aún no se tengan resultados lo suficientemente buenos. Debido a que en estas dos CNNs, al pasar por cada capa se va perdiendo cierta información para dar lugar a características de alto nivel. Como es de observarse en [12], la etapa de refinamiento implica una recurrencia (o bloque residual) basado en concatenar mapas de características o predicciones anteriores para usar información que fue predecida en cierta resolución. También, por su contraparte en un método heurístico como [26], se observa que para calcular el flujo óptico es beneficioso usar una pirámide de distintas resoluciones. Con lo dicho anteriormente, un modelo que trata de usar estas características se presenta en la Figura 3.2.3. Es de observar que las dilataciones en la primera capa se han eliminado porque sería ”análogo” a concatenar la entrada con una resolución más pequeña. También una de las principales características es como se definió el *Loss*, usando predicciones a distinta resolución que se propaga hacia resoluciones más altas. Para

hacer el escalamiento de las imágenes se usa interpolación bilineal. Sin embargo, para hacer el escalamiento del *ground-truth* es necesario modificar el factor de escala de los vectores de desplazamiento. Un enfoque sencillo para hacer esto es usar la misma interpolación bilineal, pero multiplicando el tensor de salida por el factor de escalamiento. Este factor es bastante simple de calcular si consideramos W_i, H_i como el ancho y alto del mapa de entrada y W_o, H_o como el ancho y alto de salida. Además sabemos que el mapa de entrada y salida deben tener el mismo *aspect ratio* [1], entonces debe existir un factor f_p tal que $W_i f_p = W_o$, $H_i f_p = H_o$. Finalmente el tensor final T_{corr} con el nuevo tamaño del *ground-truth* y la corrección de escala se obtiene $T_{corr} = (1 - f_p) T_{res}$, donde T_{res} es el tensor de salida después de aplicar reducción de tamaño con interpolación bilineal.

Estos modelos pueden ser pieza inicial para obtener resultados y concatenarlos de manera secuencial con pesos fijos para el refinamiento de la predicción. Parte de la metodología es experimentar con los mismos modelos con diferentes tamaños del lote del gradiente estocástico, así como distintos valores de la tasa de aprendizaje. Sin embargo, es posible que los modelos cambien al observar los resultados de las experimentaciones.

3.3. Modelo CNN con dilataciones en cascada.

Basado en los modelos preliminares y la hipótesis del campo receptivo un modelo generalizado para dicho problema es representado en la Figura 3.3.1. En cada rama de convoluciones seriadas se fija el parámetro de dilatación, en un intento de imitar las múltiples resoluciones. De este modo cada rama se encarga de aprender características que están en una resolución mayor.

3.4. Entorno de trabajo

Parte de este trabajo de investigación es proveer un entorno de trabajo que facilite la construcción y prueba de modelos para estimar flujo óptico. En este trabajo se optará por realizar esto en Python con Tensorflow [1] como biblioteca para trabajar con modelos de redes neuronales. Se optó por este, debido a que ha sido popular en los últimos años para investigación basado en *Machine Learning*. Sin embargo, entender

[1] *Aspect ratio* es la proporción de ancho y alto.

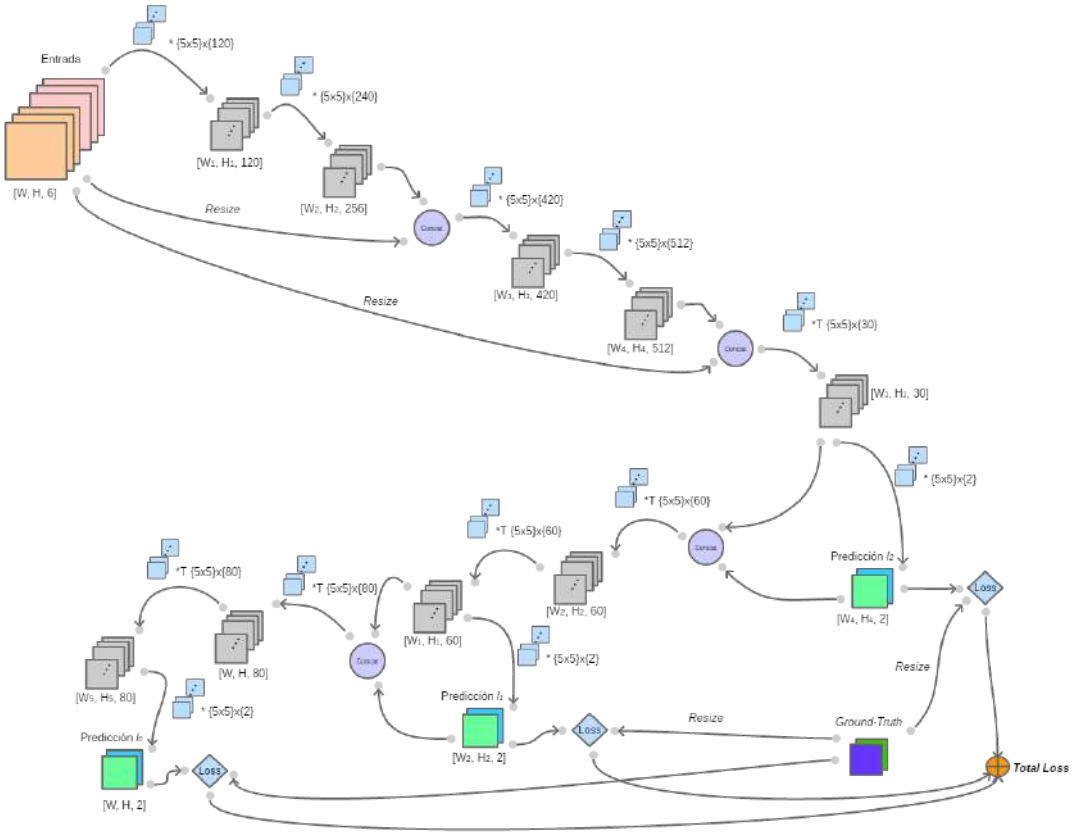


Figura 3.2.3: Diagrama del modelo 3. Como se observa este modelo es ligeramente más complejo puesto que tiene etapas que concatenan la entrada a una resolución más pequeña. Una de las diferencias con los modelos anteriores es que la primera etapa no contiene dilatación en sus filtros.

su modo de operación requiere tiempo ya que el enfoque esta basado mediante grafos de cómputo. También solo se contará con un servidor el cual contiene una GPU [2] Nvidia Tesla K40.

3.5. Bases de datos y evaluación

Para entrenar los modelos se utilizará la base de datos *Flying Chairs* [12] que también fue usada en [21]. Esta base de datos cuenta con 22872 ejemplos de flujo óptico con su respectivo *ground-truth*. Fue generada sintéticamente usando modelos 3D de sillas y poniendolas sobre fondos distintos, con los cuales se calculan sus desplaza-

[2] *Graphic Processing Unit*, unidad para procesar gráficos

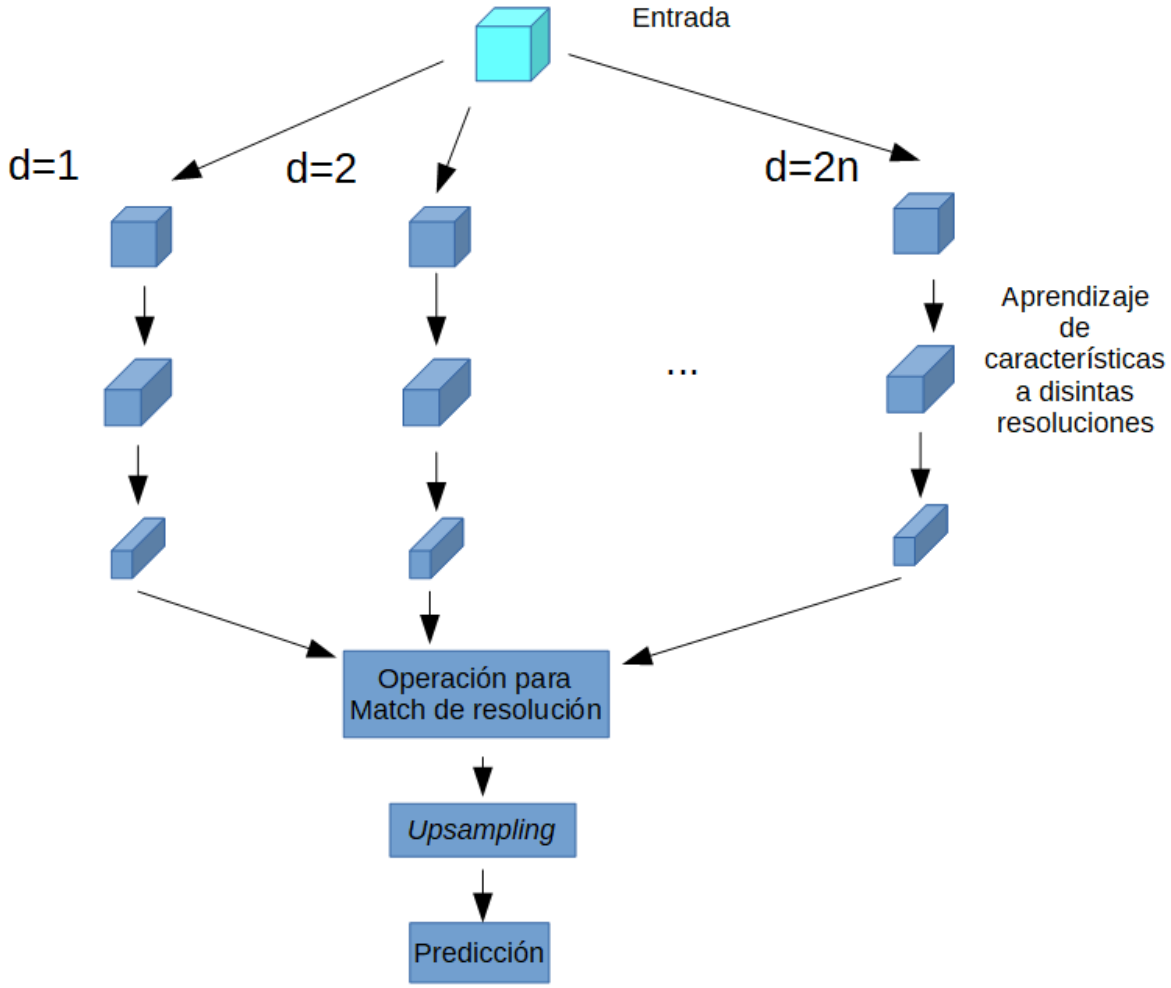


Figura 3.3.1: Modelo generalizado para implementar convolución con dilatación a distintas resoluciones. Cada rama incrementa su factor de dilatación en un factor de 2, donde n es el número de ramas. En este ejemplo el número de convoluciones aplicadas en cada rama son de 3, pero pueden ser más.

mientos y rotaciones, y a partir de esos datos se genera el *ground-truth*. En [21] se propone una partición de prueba y entrenamiento, en esta partición se cuentan con 640 ejemplos de prueba (el 3%) y el resto de entrenamiento (97%).

También existen bases de datos proporcionadas y usadas en [21], que siguen el mismo concepto de *FliyingChairs*. Una de estas bases de datos que también será de importancia es FlyingChairsSDHom, el cual es una versión de *FlyingChairs* pero con desplazamientos pequeños.

Figura 3.5.1: Ejemplo de *FlyingChairs*. A la izquierda el *frame* 1, al centro el *frame* 2 y a la derecha el *ground-truth* codificado en el estándar de Middlebury.

Finalmente, también se proporciona una base de datos mucho más extensa que las dos anteriores, *FlyingThings3D*. Esta base de datos, también fue generada sintéticamente pero usando modelos 3D de objetos, y usando la misma metodología se calcula el *ground-truth* a partir de los desplazamientos y rotaciones que se hacen a los objetos, y que ya se conocen.

Se decidió usar solo *FlyingChairs* ya que se considera que los ejemplos son suficientes.

3.5.1. División de prueba y entrenamiento

En este trabajo tendremos dos fases de experimentación. En la primera fase se explorará la complejidad del problema haciendo experimentación con pocos ejemplos y usando *K-Folds cross validation*. Durante esta fase observaremos el comportamiento de las curvas de error de prueba y entrenamiento y basado en esos datos decidiremos cuantas épocas son adecuadas para entrenar en este trabajo.

En la segunda fase dividiremos la base de datos en tres porciones para validación cruzada. Extraeremos 80 ejemplos del conjunto de entrenamiento usado en [21] y los ejemplos 1122, 1127, 1129, 1139, 1143, 1151, 1181, 1196, 1214 seleccionados del conjunto de prueba. El resto de ejemplos se usarán para entrenar pero no con todo el conjunto, incrementaremos el conjunto para entrenar cuando se crea conveniente. Además que no se cuenta con el espacio suficiente en RAM para cargarla y pre-procesarla toda y hacer constantes cargas de los ejemplos alentaría el entrenamiento. Parte importante del modelo de red es también el método de optimización que se usará. Para este trabajo se optó por usar el método de optimización ADAM [23] debido a que pre-

senta características como un aprendizaje que converge más rápido y más estable que otros métodos, además que es uno de los más recientes en el estado del arte. Sin embargo, también se optará por usar el descenso de gradiente estocástico en algunos casos.

3.5.2. Detalles de la función de evaluación

Para evaluar cada uno de los modelos propuestos, ya sea con el conjunto de prueba o conjunto de evaluación, se usa solo la función *Average endpoint error* 2.5.4 (la cual es una versión generalizada), de manera precisa queda definida como sigue

$$AEPE = \frac{1}{nm} \sum_{\mathbf{p} \in I} \parallel \mathbf{u_{true}}(\mathbf{p}) - \mathbf{u_{predic}}(\mathbf{p}) \parallel$$

donde $u_{predic}(\mathbf{p})$ es el flujo estimado en la posición $\mathbf{p}$ en la imagen I y $u_{true}(\mathbf{p})$ el flujo real, n y m son el ancho y alto de la imagen.

Capítulo 4

Experimentación y resultados

4.1. Experimentación preliminar y exploratorio.

La sección actual tiene como propósito explorar y dar una observación preliminar de la problemática con las redes neuronales para estimar el flujo óptico. Con el objetivo de hacer pruebas preliminares no sé utilizó toda la base de datos, puesto que si se entrena cierto modelo y no resulta efectivo será un desperdicio de tiempo y cómputo.

En un primer enfoque, se experimentó con modelos sencillos de redes que determinen de manera empírica un modelo suficiente para desarrollar la tarea de estimación del flujo óptico. Primeramente se usó el modelo mostrado en la Figura 3.2.1, como puede observarse este modelo es demasiado pequeño. Durante la experimentación se notó cadencias en su aprendizaje puesto que los parámetros no eran suficientes, por tanto se dejó de experimentar y se descartó.

Con el modelo 2, presentado en la Figura 3.2.2, se empezó a tener mejores resultados. Este modelo se entrenó solamente con 30 ejemplos de la base de datos. Usando ADAM con una tasa de aprendizaje de $0{,}0001$, tamaño de *batch* de 1 durante 500 épocas. Las función de costo correspondiente se presenta en la gráfica 4.1.1. Como se observa luego de las primeras iteraciones el costo empieza a ser muy ruidoso. En la Figura 4.1.2 se observa el resultado de utilizar un ejemplo del conjunto de entrenamiento como prueba. Este experimento de hacer inferencia con un mismo ejemplo de entrenamiento solo sirve a modo de *debug*.

Sin embargo, es evidente que cuando se presenta un ejemplo nunca antes visto al modelo no logra hacer una generalización de la inferencia. Esto se puede observar en la Figura 4.1.3 donde se usa un ejemplo que nunca ha visto la red. Sin embargo, a

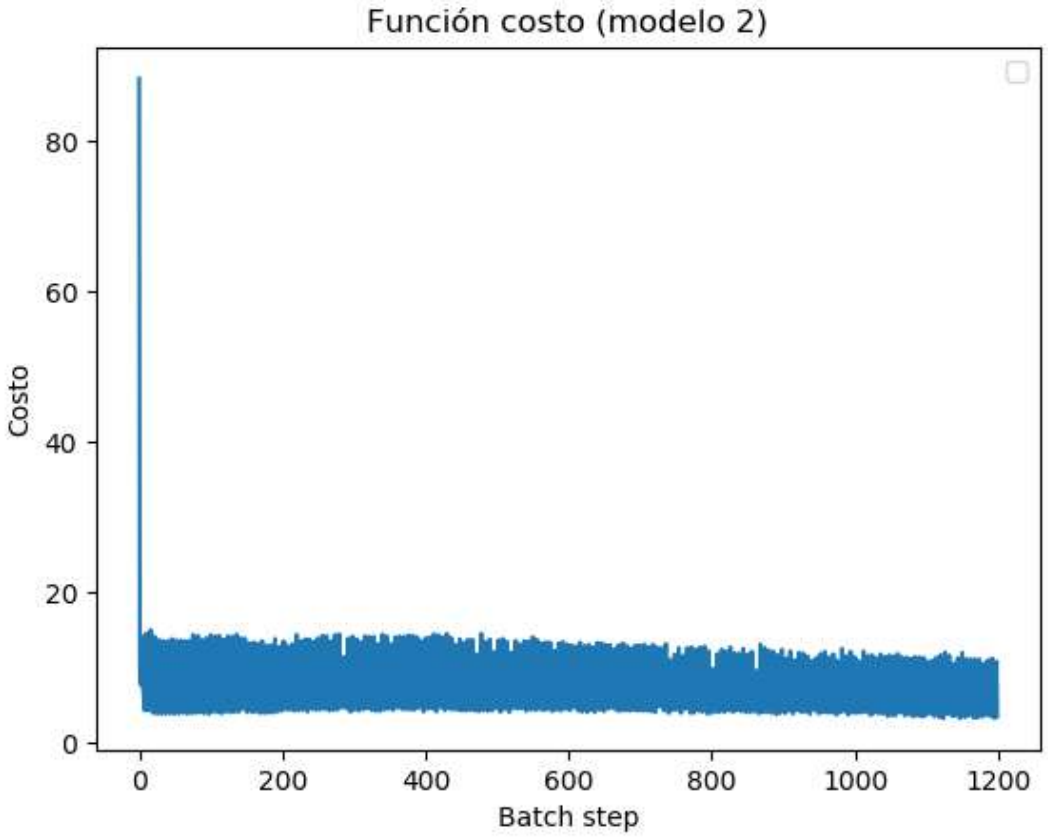


Figura 4.1.1: gráfica del costo por cada paso de *batch* (en esta gráfica no todos los valores son graficados, en este caso cada 5 pasos de *batch*).

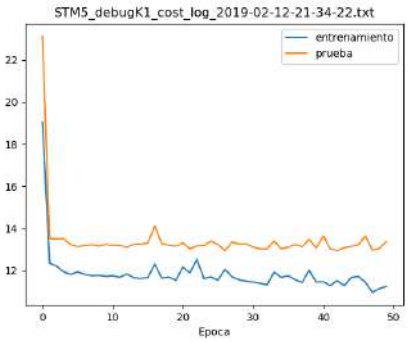
pesar de eso, se puede empezar a notar lugares donde el color intenta aproximarse a como sería el *ground-truth*.

Se hizó pruebas con validación *K-Folds Cross validation*, con el modelo descrito en 3.2.2, y solo se utilizó una pequeña partición de la base de datos. Se usó 50 ejemplos particionados en 4 y usando una partición como validación en cada iteración, es decir, el parámetro de K-Folds fue $k = 4$. Se entrenó durante 50 épocas con una taza de aprendizaje de $0{,}0001$, usando ADAM. La gráfica del costo por época de la iteración 1, 2, 3 y 4 se representan en la Figura 4.1.4. Mientras que el promedio de las 4 iteraciones de *K-Folds* se representa en la Figura 4.1.5. Como se observa se tienen comportamientos bastante ruidosos en el costo de entrenamiento, los cuales se notan más en la segunda y tercera iteración, mientras que en el costo del conjunto de validación tienen comportamientos bastante erráticos (ver cuarta iteración). Sin embargo, observando el promedio en la Figura 4.1.5 se alcanza a percibir una tendencia de bajada en el costo de entrenamiento mientras que el costo de validación solo tiene una

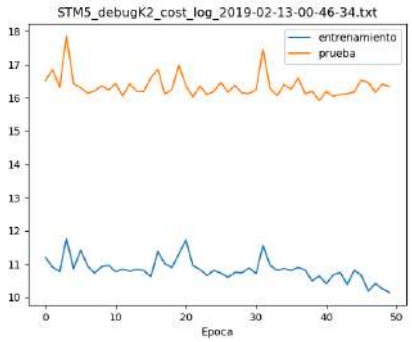
Figura 4.1.2: Inferencia usando un ejemplo del mismo entrenamiento con el modelo 2 (3.2.2). A la izquierda se encuentra el *ground-truth* y a la derecha la inferencia. Es de notarse que la inferencia no debe ser perfecta, pero aún así logra arrojar un indicio de que los pesos se están aproximando a un óptimo.

Figura 4.1.3: Inferencia usando un ejemplo nunca antes visto por la red del modelo 2 (3.2.2). A la izquierda se encuentra el *ground-truth* y a la derecha la inferencia. Como se puede observar la red aún cuenta con problemas de generalización.

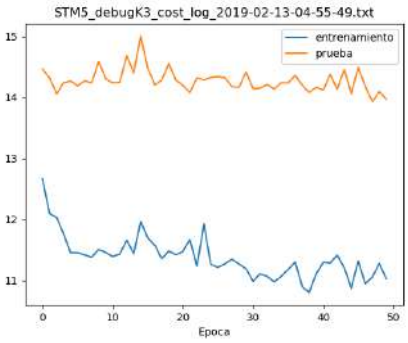
bajada que se observa en las primeras 10 iteraciones, esto podría significar que cerca de las 10 iteraciones es suficiente para entrenar sin que exista *over-fitting*.



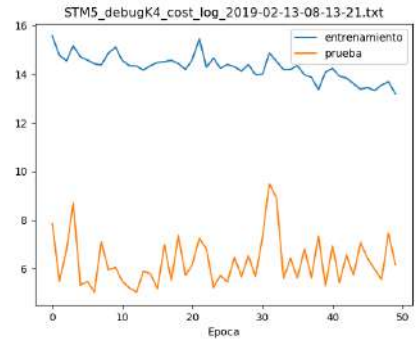
(a) Partición 1



(b) Partición 2



(c) Partición 3



(d) Partición 4

Figura 4.1.4: Validación cruzada en 4 particiones aplicada en el modelo 2, en estas gráficas se muestra la función de costo. 3.2.2 usando 50 ejemplos (10 de prueba y 40 de entrenamiento).

Posteriormente y usando el mismo modelo, se incremento el número de ejemplos a 200 usando las mismas características de entrenamiento descrito anteriormente. Esto significa que en cada iteración (de las 4) se usa 150 ejemplos para entrenar y 50 para validar. También se entrenó durante 50 épocas con una taza de aprendizaje de $0{,}0001$, usando ADAM. La gráfica del costo por época de la iteración 1, 2, 3 y 4 se representan en la Figura 4.1.6. Mientras que el promedio de las 4 iteraciones de *K-Folds* se representa en la Figura 4.1.7. Como podemos ver en las gráficas, se continúa teniendo un comportamiento muy ruidoso, y de nuevo parece que entrenar más de 10 épocas no parece tener un beneficio sobresaliente. Y observando algunos casos como en la partición 3, se tiene un comportamiento inestable.

Basado en los resultados vistos anteriormente se decidió re-diseñar un nuevo modelo y posponer la experimentación con el modelo 3. El modelo 4 está definido en la Tabla 4.1.1 y como se podrá observar tiene una estructura similar a la de la UNET [33] con la diferencia de que tiene varias concatenaciones de la entrada en distintas resoluciones con el objetivo de seguir haciendo una homologa a la estructura piramidal de

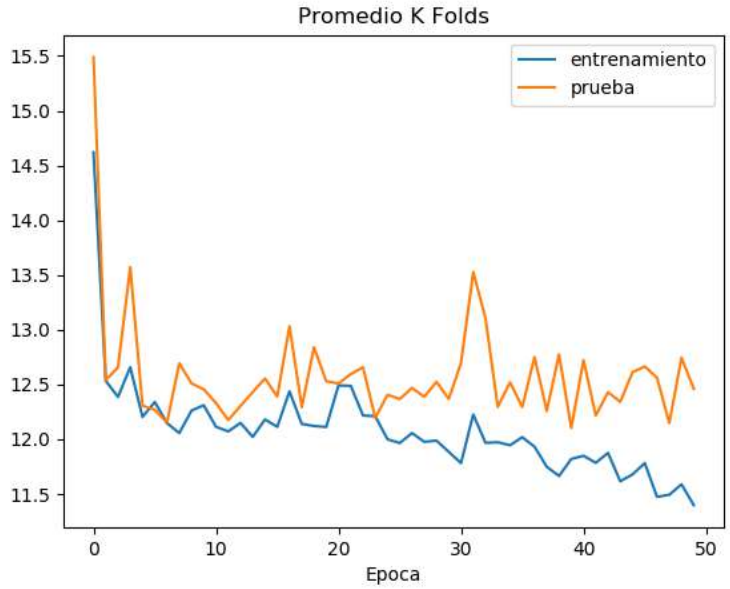
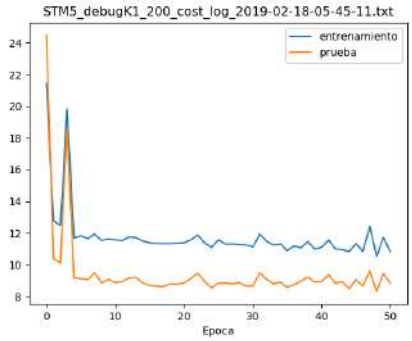


Figura 4.1.5: En esta gráfica se muestra el costo promedio de las 4 particiones por época de entrenamiento del modelo 3.2.2

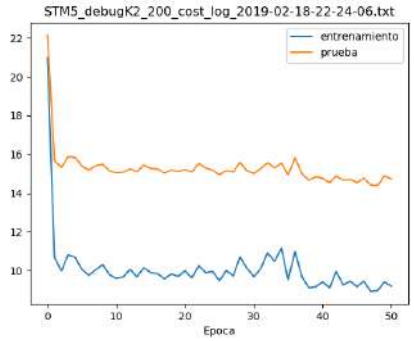
.

resolución como en el método de Lukas-Kanade [26].

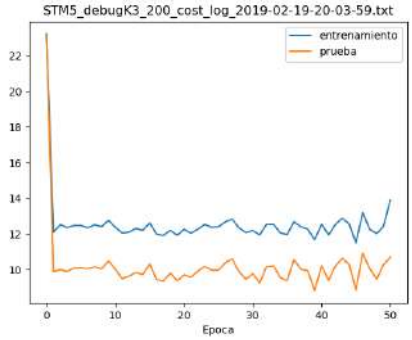
Se prosiguió realizando pruebas, usando el modelo 4, se realizó *K-Folds* con 200 ejemplos de entrenamiento y se uso una partición de 10, es decir, por cada iteración se usarán 190 ejemplos para entrenar y 10 ejemplos para validar. Esta vez se optó por usar descenso de gradiente común con una tasa de aprendizaje de $1e-05$ solo para observar si también posee un comportamiento ruidoso en el descenso del costo. Las gráficas de costo y validación de las iteraciones 1 a la 10 están representadas por la Figura 4.1.8. El costo promedio en el entrenamiento y prueba de todas las iteraciones del *K-Folds* se representa en la Figura 4.1.9.



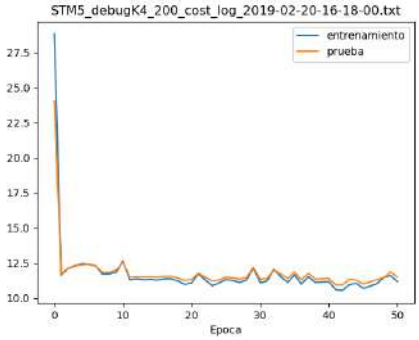
(a) Partición 1



(b) Partición 2



(c) Partición 3



(d) Partición 4

Figura 4.1.6: Validación cruzada en 4 particiones aplicada en el modelo 2 3.2.2 usando 200 ejemplos (50 de prueba y 150 de entrenamiento).

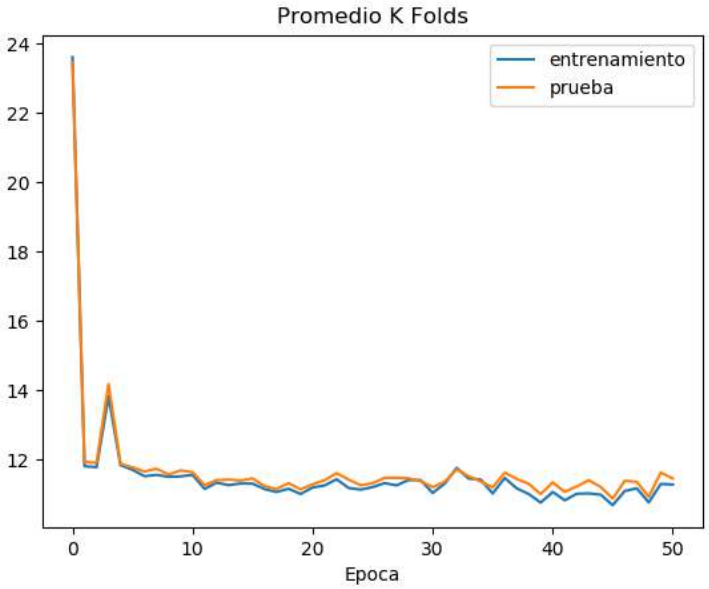
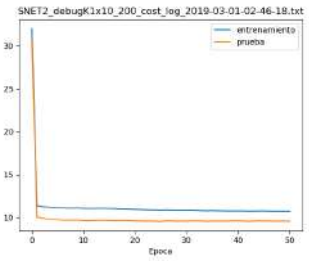


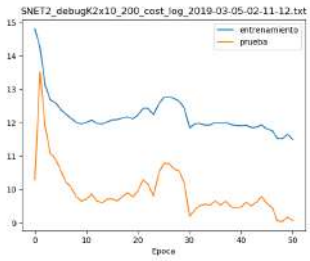
Figura 4.1.7: En esta gráfica se muestra el costo promedio de las 4 particiones por época de entrenamiento del modelo 3.2.2 usando 200 ejemplos.

Capa	Entrada	Salida	Tipo	# Filtros	Dim. filtro	Activación
1	In	O_1	Convolución	64	7	ReLU
2	O_1	O_2	Convolución	120	5	ReLU
3	O_2	O_3	Convolución	120	5	ReLU
4	O_3	O_4	Convolución	240	5	ReLU
5	O_4 & In	O_5	Convolución	60	7	ReLU
6	O_5	O_6	Convolución	120	5	ReLU
7	O_6	O_7	Convolución	120	5	ReLU
8	O_7	O_8	Convolución	240	5	ReLU
9	O_8 & In	O_9	Convolución	60	7	ReLU
10	O_9	O_{10}	Convolución	120	5	ReLU
11	O_{10}	O_{11}	Convolución	120	5	ReLU
12	O_{11}	O_{12}	Convolución	240	5	ReLU
13	$O12$	O_{13}	Transpuesta	60	5	NA
14	O_{13}	O_{14}	Transpuesta	50	5	NA
15	O_{14}	O_{15}	Transpuesta	40	5	NA
16	O_{15}	O_{16}	Transpuesta	30	7	NA
17	O_{16} & O_8	O_{17}	Transpuesta	50	5	NA
18	O_{17}	O_{18}	Transpuesta	40	5	NA
19	O_{18}	O_{19}	Transpuesta	30	5	NA
20	O_{19}	O_{20}	Transpuesta	20	7	NA
21	O_{20} & O_4	O_{21}	Transpuesta	40	5	NA
22	O_{21}	O_{22}	Transpuesta	30	5	NA
23	O_{22}	O_{23}	Transpuesta	20	5	NA
24	O_{23}	O_{24}	Transpuesta	10	7	NA
25	O_{24}	O_{25}	Padding	NA	NA	NA
26	O_{25}	O_f	Convolución	2	5	NA

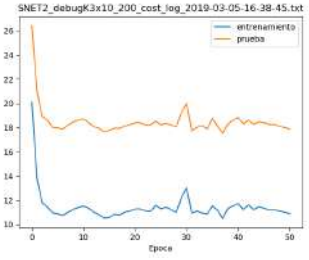
Tabla 4.1.1: Estructura del modelo 4. Se usa O_{ij} para denotar la salida de la capa. El símbolo & se usa para denotar una concatenación, mientras que In simplemente significa que es el ejemplo de entrada. Se omiten los procesamientos de escalamiento de la entrada para coincidir con las dimensiones de cierta salida, pero es necesario dicho proceso en el cual se utiliza escalamiento con interpolación bilineal.



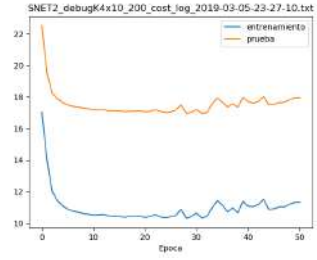
(a) Partición 1



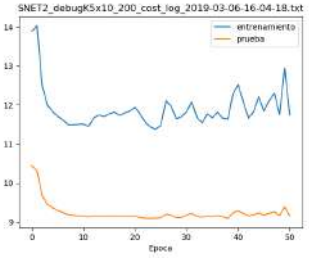
(b) Partición 2



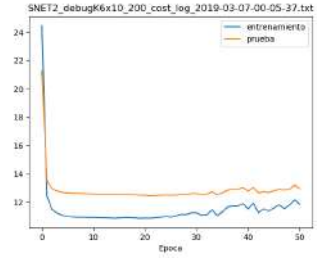
(c) Partición 3



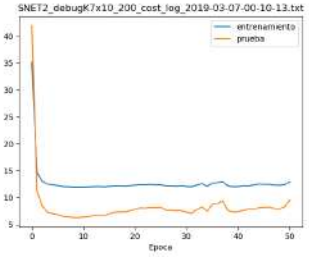
(d) Partición 4



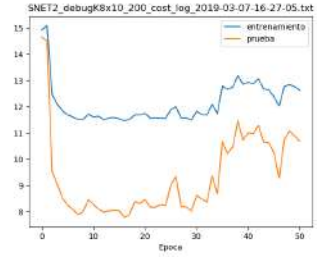
(e) Partición 5



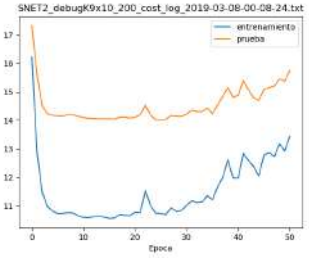
(f) Partición 6



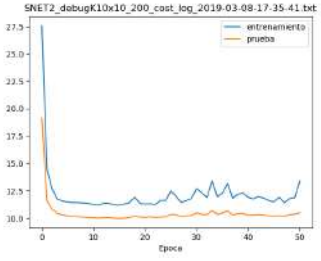
(g) Partición 7



(h) Partición 8



(i) Partición 9



(j) Partición 10

Figura 4.1.8: Validación cruzada en 10 particiones aplicada en el modelo 4 4.1.1 usando 200 ejemplos (20 de prueba y 180 de entrenamiento).

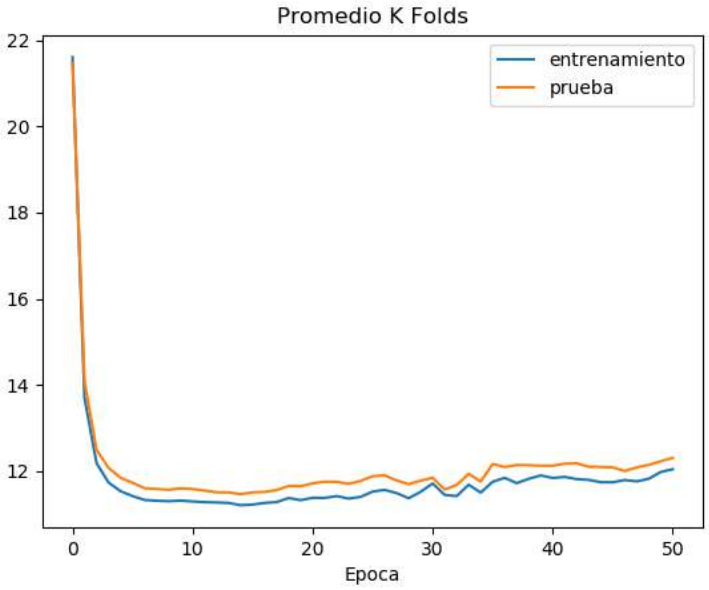


Figura 4.1.9: En esta gráfica se muestra el costo promedio de las particiones del *K-Folds* usando el modelo descrito en el cuadro 4.1.1 con tasa de aprendizaje de $1e-5$ y descenso de gradiente estocástico.

4.2. Experimentación (segunda etapa)

Para encontrar una tasa de aprendizaje adecuada se decidió entrenar el modelo 2, 3 y 4 con diferentes valores de este parámetro. Desde este punto, todos los modelos comparten la característica de entrenamiento. Las condiciones para este experimento se fijaron como sigue:

- Se estableció los primeros 320 ejemplos como entrenamiento y los siguientes 80 (del 321 al 400) como ejemplos para validación.
- Por cada modelo, se prueba con los valores $0{,}0001$, $0{,}00001$, y $0{,}000001$ de tasa de aprendizaje.
- Se entrenan durante 7 épocas, puesto que en la sección anterior se observó que era un valor suficiente.
- Para graficar la curva de costo de error y prueba, se calcula el costo promedio de los respectivos conjuntos de ejemplos cada 25 pasos de *batch*, es decir, cada 25 pasos de actualización de los pesos.
- La función de costo (como entrenamiento) a utilizar es L_1.

Para tener una mejor visualización de la comparación se graficó la media y la desviación estándar de cada configuración de entrenamiento, la gráfica correspondiente a los resultados del experimento descrito anteriormente puede observarse en la Figura 4.2.1. En este experimento es notable que el peor modelo ha sido el 3, una posible explicación es que al cambiar el tamaño de los ejemplos para calcular el costo en una resolución dada se pierda cierta información o cambia la precisión del dato ya que al aplicar el factor de escalamiento el flujo ya no es exacto. Dicho lo anterior, es preferible no aplicar operaciones que requieran la modificación de escala de un ejemplo. Mientras que el modelo 4 y 2 apenas son diferenciables en el gráfico, pero numéricamente el modelo 4 es ligeramente mejor, esto puede observarse en la Tabla 4.2.1. Las gráficas de costo del entrenamiento del experimento descrito anteriormente pueden observarse en la Figura 4.2.2, en estas gráficas observamos que cuando se usa una tasa de aprendizaje alto $(0{,}0001)$ se tiene un pico de subida en el costo que posteriormente decrece, esto podría significar que no es una tasa de aprendizaje adecuada, sin embargo, en el gráfico 4.2.1 notamos que en el conjunto de prueba no hay una diferencia significativa con una tasa de aprendizaje de $0{,}00001$.

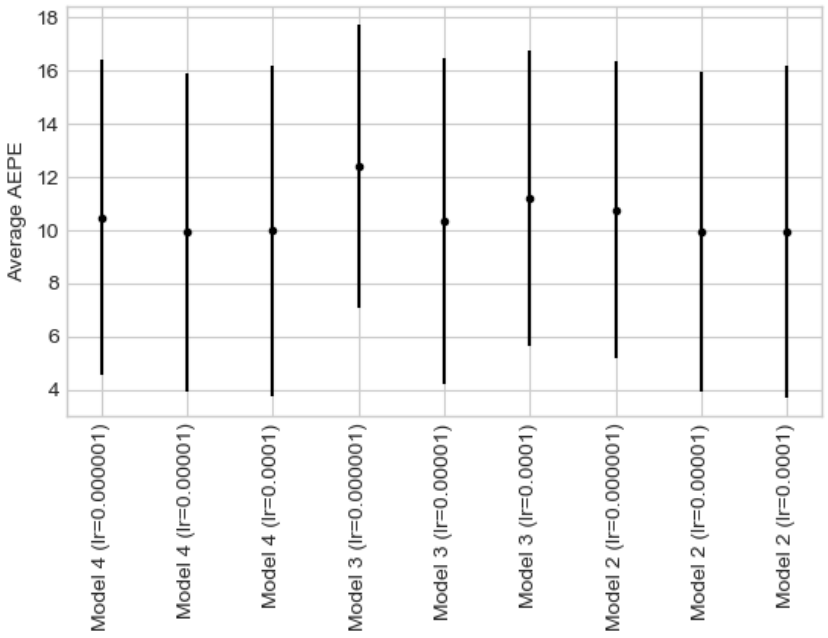


Figura 4.2.1: Comparación de la configuración de entrenamiento de cada uno de los modelos entrenados. El punto significa la media del *Average Enpoint Error* de los 80 ejemplos tomados de prueba, mientras que la barra indica la desviación estándar. Como es de observarse el peor resultado es obtenido con el modelo 3. Mientras que no hay mucha diferencia entre el modelo 4 y 2. Es notable que estos últimos 2 tienen cierta similitud en sus resultados tomando en cuenta la tasa de aprendizaje.

Modelo	Media	Desviación estándar
Model 4 (lr=0.000001)	10.4823081	5.92217183
Model 4 (lr=0.00001)	9.92308864	5.9792455
Model 4 (lr=0.0001)	9.99716119	6.20704336
Model 3 (lr=0.000001)	12.40962843	5.322845
Model 3 (lr=0.00001)	10.33264819	6.10675454
Model 3 (lr=0.0001)	11.20599681	5.56594868
Model 2 (lr=0.000001)	10.77072493	5.58547466
Model 2 (lr=0.00001)	9.94977164	5.99467444
Model 2 (lr=0.0001)	9.95360812	6.2238073

Tabla 4.2.1: Resultados de la experimentación junto con sus medias y desviaciones estándar. Esto es el *average enpoint error* re-calculado para evitar conflictos. Nótese que en el modelo 3 la función de costo es diferente y por eso se hizo lo anterior. Las medias y desviaciones estándar son de los 80 ejemplos que se separaron para hacer la validación cruzada.

Capa	Entrada	Salida	Tipo	# Filtros	Dim. filtro	Activación
1	In	O_1	Convolución	64	7	ReLU
2	O_1	O_2	Convolución	120	5	ReLU
3	O_2	O_3	Convolución	120	5	ReLU
4	O_3	O_4	Convolución	240	5	ReLU
5	O_4	O_5	Convolución	60	7	ReLU
6	O_5	O_6	Convolución	120	5	ReLU
7	O_6	O_7	Convolución	120	5	ReLU
8	O_7	O_8	Convolución	240	5	ReLU
9	O_8	O_9	Convolución	60	7	ReLU
10	O_9	O_{10}	Convolución	120	5	ReLU
11	O_{10}	O_{11}	Convolución	120	5	ReLU
12	O_{11}	O_{12}	Convolución	240	5	ReLU
13	O_{12}	O_{13}	Transpuesta	60	5	NA
14	O_{13}	O_{14}	Transpuesta	50	5	NA
15	O_{14}	O_{15}	Transpuesta	40	5	NA
16	O_{15}	O_{16}	Transpuesta	30	7	NA
17	O_{16} & O_8	O_{17}	Transpuesta	50	5	NA
18	O_{17}	O_{18}	Transpuesta	40	5	NA
19	O_{18}	O_{19}	Transpuesta	30	5	NA
20	O_{19}	O_{20}	Transpuesta	20	7	NA
21	O_{20} & O_4	O_{21}	Transpuesta	40	5	NA
22	O_{21}	O_{22}	Transpuesta	30	5	NA
23	O_{22}	O_{23}	Transpuesta	20	5	NA
24	O_{23}	O_{24}	Transpuesta	10	7	NA
25	O_{24}	O_{25}	Padding $p=3$	NA	NA	NA
26	O_{25}	O_f	Convolución	2	5	NA

Tabla 4.2.2: Estructura del modelo 5. Se usa O_{ij} para denotar la salida de la capa. El símbolo & se usa para denotar una concatenación, mientras que In simplemente significa que es el ejemplo de entrada. Este modelo cuenta con $\mathbf{6554542}$ **parámetros**. Este modelo 5 es base para el modelo 6, el cual añade dropout después de la capa 2, 4, 6, 8 y 12. Con probabilidad de cancelar la neurona de $0{,}6$.

Con los resultados vistos anteriormente, en la Figura 4.2.1, se nota que en la mayoría de los modelos hay una falta de generalización. Se añadió un modelo nuevo basado en modificaciones de la UNET, el cual es similar al modelo 4 4.1.1 pero quitando las concatenaciones de la entrada en distinta resolución, llámese **modelo 5**, en la Tabla 4.2.2 se define con detalle el modelo 5. Notando que una tasa de aprendizaje adecuada es de $0{,}00001$ se fijó a este valor. El modelo 3 3.2.3 fue desechado completamente debido a su alto error. Se añadió el **modelo 7** que usa el modelo general propuesto en la metodología 3.3.1 y definido específicamente por la Tabla 4.2.3. En la Figura 4.2.3 se presenta los resultados (con el conjunto de prueba) después de entrenar solo el

modelo 5 y 7, el resultado del modelo 4 se tomó de la gráfica del experimento anterior (4.2.1) solo como referencia, en este experimento 4.2.3 el modelo 5 fue entrenado con 1000 ejemplos a diferencia del modelo 7 (con 320 ejemplos), como se hizo al principio. Además, en la misma figura se observa un modelo 5 con dropout, el cual puede ser tomado como el **Modelo 6**, y en especifico solo añade dropout (con probabilidad de cancelar la neurona de 0.6) cada dos capas de convolución en la etapa de *downsampling* del modelo 5. Se nota que hasta ahora el modelo 5 tiene ligeramente mejor resultado pero no podemos corroborar esto aún.

Observando que aumentar el número de ejemplos es un factor importante se decidió aumentar el número de ejemplos. Con lo dicho anteriormente, para diferenciar resultados entre un modelo con y sin convoluciones dilatadas se entrenó solo el modelo 7 y modelo 5 con las mismas especificaciones:

- Tamaño de batch de 20
- 5000 ejemplos de entrenamiento
- Tasa de aprendizaje de $0{,}0001$. Se decidió usar esta tasa de aprendizaje debido que como el batch es mayor entonces el costo tiende a ser menos ruidoso.

En la Figura 4.2.4 se puede comparar los resultados de este último experimento con el conjunto de prueba. Se mantuvieron los resultados con 1000 ejemplos del modelo 5 en el experimento anterior de la Figura 4.2.3 solo para comparar. En este último experimento observamos que aumentar el número de ejemplos y el tamaño de batch mejora la predicción en ambos modelos. Sin embargo, el modelo 5 aún sigue siendo ligeramente mejor.

Para verificar más a fondo la influencia del número de ejemplos con el que es entrenado se decidió entrenar el modelo 7 con 18000 ejemplos y el resto de especificaciones como se describieron en el anterior entrenamiento. Pero no sé notó una mejoría significativa, esto lo podemos observar en la Figura 4.2.5.

Haciendo referencia a los últimos resultados del **modelo 7** entrenado con 5000 ejemplos y batches de 20, con tasa de aprendizaje de $0{,}0001$. Obtenemos los resultados vistos en la Tabla 4.2.4.

El modelo 7 propuesto en este trabajo contiene una primera aproximación del flujo óptico a poco costo de parámetros ($\mathbf{3734220}$). Esto puede verse en la Tabla 4.2.4. Los resultados, vistos en la Figura 4.2.6, como podrán notarse no son comparables con FlowNet2, puesto que FlowNet2 tiene muchos más parámetros (cerca de 160 millones) para generalizar y dar una mejor predicción. También, cabe mencionar que los resultados no pasan por ningún post-procesamiento para conservar bordes y mantener el

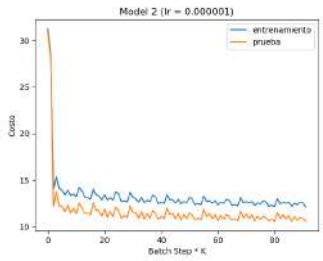
Capa	Entrada	Salida	Tipo	# Filtros	Dim. filtro	Activación
1	In	O_{11}	Convolución $d=1$	60	5	ReLU
2	O_{11}	O_{12}	Convolución $d=1$	80	5	ReLU
3	O_{12}	O_{13}	Convolución $d=1$	120	5	ReLU
4	In	O_{21}	Convolución $d=2$	60	5	ReLU
5	O_{21}	O_{22}	Convolución $d=2$	80	5	ReLU
6	O_{22}	O_{23}	Convolución $d=2$	120	5	ReLU
7	In	O_{31}	Convolución $d=4$	60	5	ReLU
8	O_{31}	O_{32}	Convolución $d=4$	80	5	ReLU
9	O_{32}	O_{33}	Convolución $d=4$	120	5	ReLU
10	In	O_{41}	Convolución $d=6$	60	5	ReLU
11	O_{41}	O_{42}	Convolución $d=6$	80	5	ReLU
12	O_{42}	O_{43}	Convolución $d=6$	120	5	ReLU
13	O_{43}	O_{p1}	*Padding* $p=9$	NA	NA	NA
14	O_{p1}	O_{t1}	Transpuesta	80	7	ReLU
15	O_{33} & O_{t1}	O_{p2}	*Padding* $p=9$	NA	NA	NA
16	O_{p2}	O_{t2}	Transpuesta	60	7	ReLU
17	O_{23} & O_{t2}	O_{p3}	*Padding* $p=3$	NA	NA	NA
18	O_{p3}	O_{t3}	Transpuesta	40	7	ReLU
19	O_{13} & O_{t3}	O_{f1}	Transpuesta	30	5	ReLU
20	O_{f1}	O_{f2}	Transpuesta	20	5	ReLU
21	O_{f2}	O_{f3}	Transpuesta	10	5	ReLU
22	O_{f3}	O_{pf}	Padding $p=3$	NA	NA	NA
23	O_{pf}	O_f	Convolución	2	7	NA

Tabla 4.2.3: Estructura del modelo 7. Se usa O_{ij} para denotar la salida de la capa. El símbolo & se usa para denotar una concatenación, mientras que In denota el ejemplo de entrada. Este modelo cuenta con **3734220 parámetros** para ser entrenados.

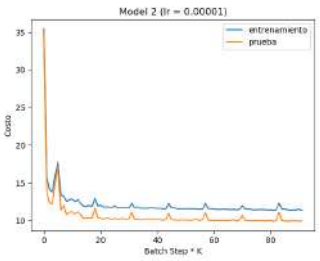
flujo regular mientras que FlowNet2 si. También podemos observar los resultados del modelo 5 en la Figura 2.3.4. Si observamos los resultados del modelo 5 y 7 nos daremos cuenta que son muy parecidos, sin embargo, el modelo 7 como tiene mayor campo receptivo, tiende a tener más pixeles con mejor orientación o más de acuerdo al flujo real.

# Ejemplo FlyingChairs	Modelo 7	FlowNet2
1122	6.7193	1.4725
1127	19.907642	8.5733
1129	2.635594	1.2909
1139	1.9210024	0.7905
1143	4.2178893	0.5706
1151	10.31673	2.3566
1181	2.9278612	0.6769
1196	4.9561753	1.0405
1214	3.9331377	0.7053

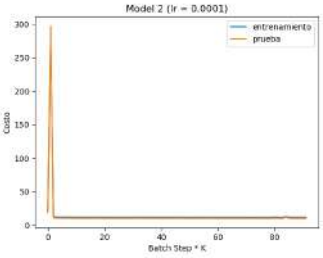
Tabla 4.2.4: Contraste de resultados de AEPE del modelo 7 y FlowNet2. Los ejemplos fueron seleccionados de la partición de validación y prueba de FlyingChairs para asegurar que no hayan sido ejemplos de entrenamiento en FlowNet2.



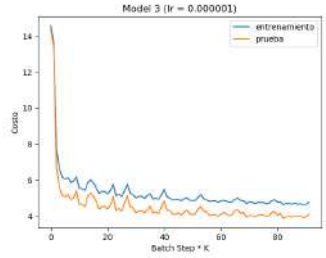
(a) Modelo 2 (tasa de aprendizaje $0{,}000001$).



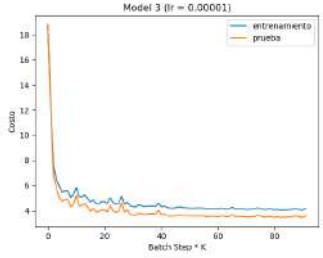
(b) Modelo 2 (tasa de aprendizaje $0{,}00001$).



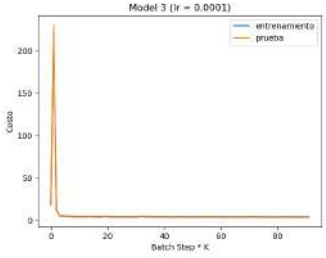
(c) Modelo 2 (tasa de aprendizaje $0{,}0001$).



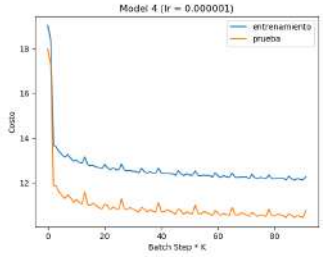
(d) Modelo 3 (tasa de aprendizaje $0{,}000001$).



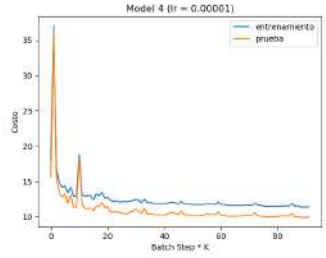
(e) Modelo 3 (tasa de aprendizaje $0{,}00001$).



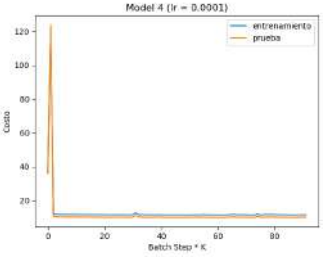
(f) Modelo 3 (tasa de aprendizaje $0{,}0001$).



(g) Modelo 4 (tasa de aprendizaje $0{,}000001$).



(h) Modelo 4 (tasa de aprendizaje $0{,}00001$).



(i) Modelo 4 (tasa de aprendizaje $0{,}0001$).

Figura 4.2.2: Gráficas de costo de entrenamiento y prueba (de los 80 ejemplos seleccionados). Este costo se calcula cada 25 operaciones de entrenamiento.

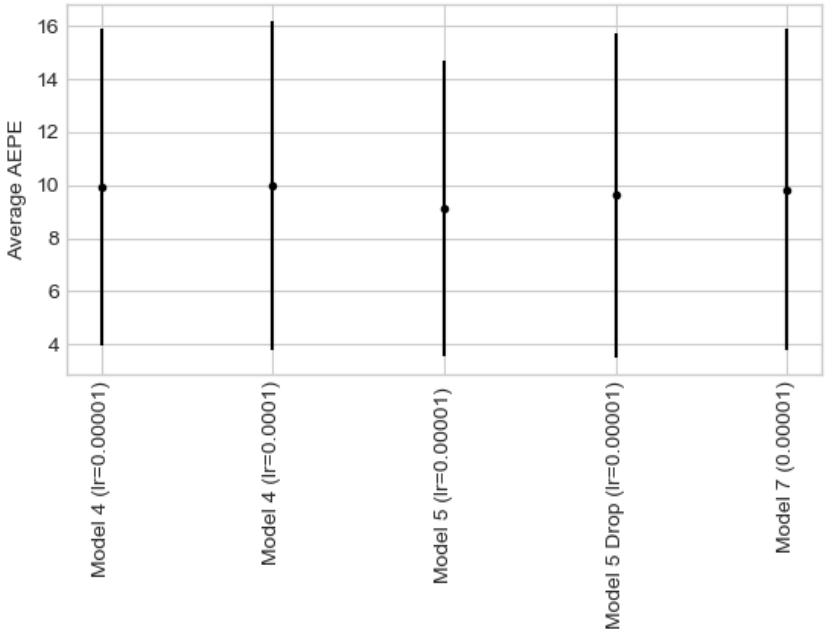


Figura 4.2.3: Comparación de la configuración de entrenamiento de cada uno de los modelos entrenados. El modelo cuatro continúa teniendo las mismas especificaciones que la Figura anterior 4.2.1. El modelo 5 (y con dropout) usan 1000 ejemplos de entrenamiento.

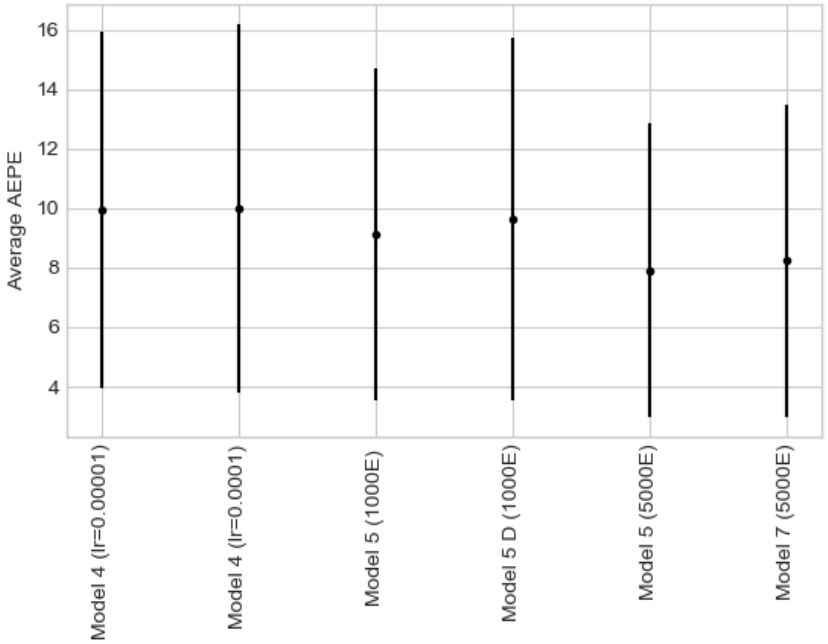


Figura 4.2.4: Comparación de la configuración de entrenamiento de cada uno de los modelos entrenados. Los modelos con el indicador (5000E) son los últimos que han sido entrenados con las especificaciones descritas de 5000 ejemplos. Para comparar se mantuvo el entrenamiento con 1000 ejemplos del modelo 5 (denotado 1000E), también se conserva el resultado del modelo 4 de la experimentación al principio como referencia.

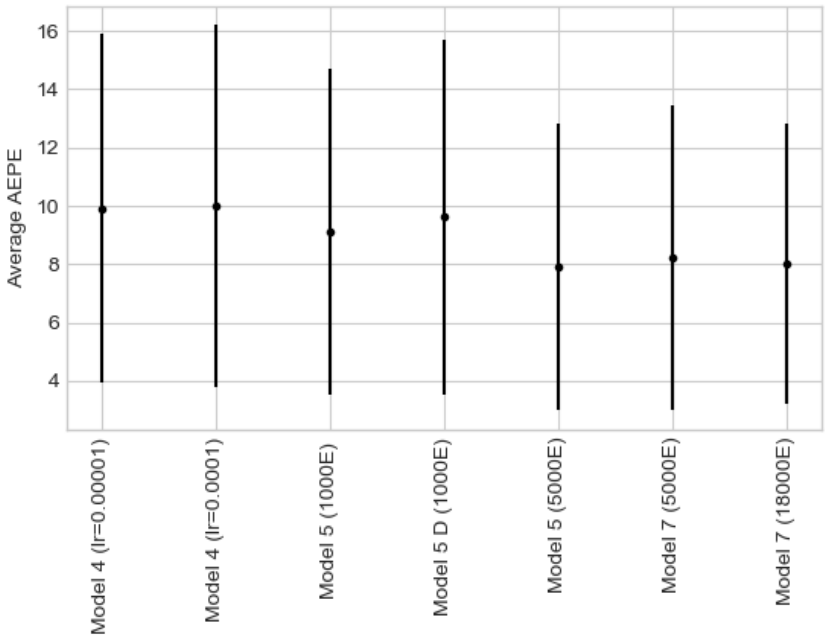


Figura 4.2.5: Comparación de la configuración de entrenamiento de cada uno de los modelos entrenados. Los modelos con el indicador (5000E) son los últimos que han sido entrenados con las especificaciones descritas de 5000 ejemplos. Para comparar se mantuvo el entrenamiento con 1000 ejemplos del modelo 5 (denotado 1000E), también se conserva el resultado del modelo 4 de la experimentación al principio como referencia. El modelo 7 al final (denotado 18000E) fue entrenado con 18000 ejemplos.

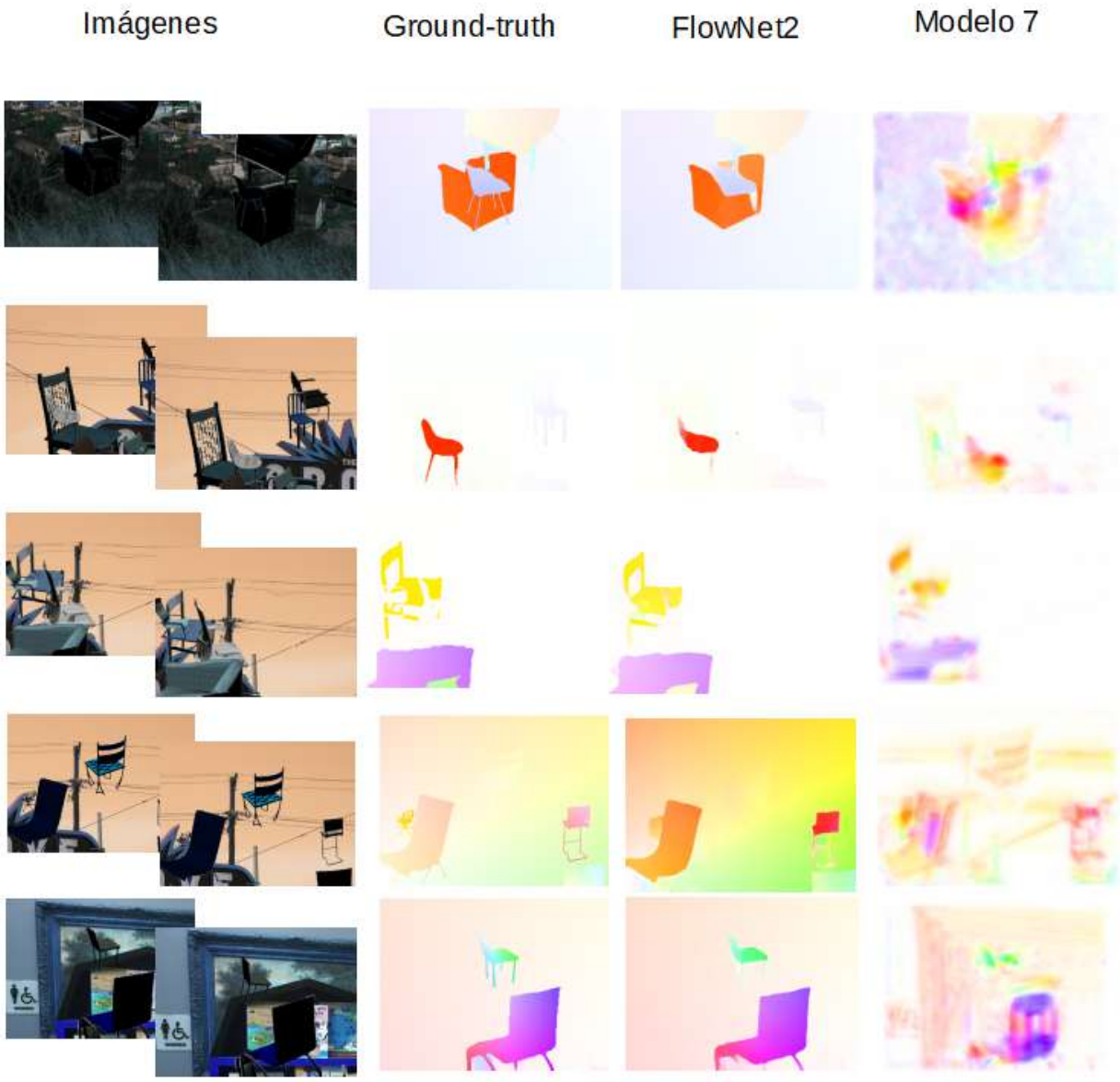


Figura 4.2.6: Campos de flujo resultantes de cada modelo. En la primeras 2 columnas se presenta el par de imágenes y el *ground_truth*. En la tercera columna el campo de flujo producido por FlowNet2 y en la última columna el campo de flujo producido por el modelo 7.

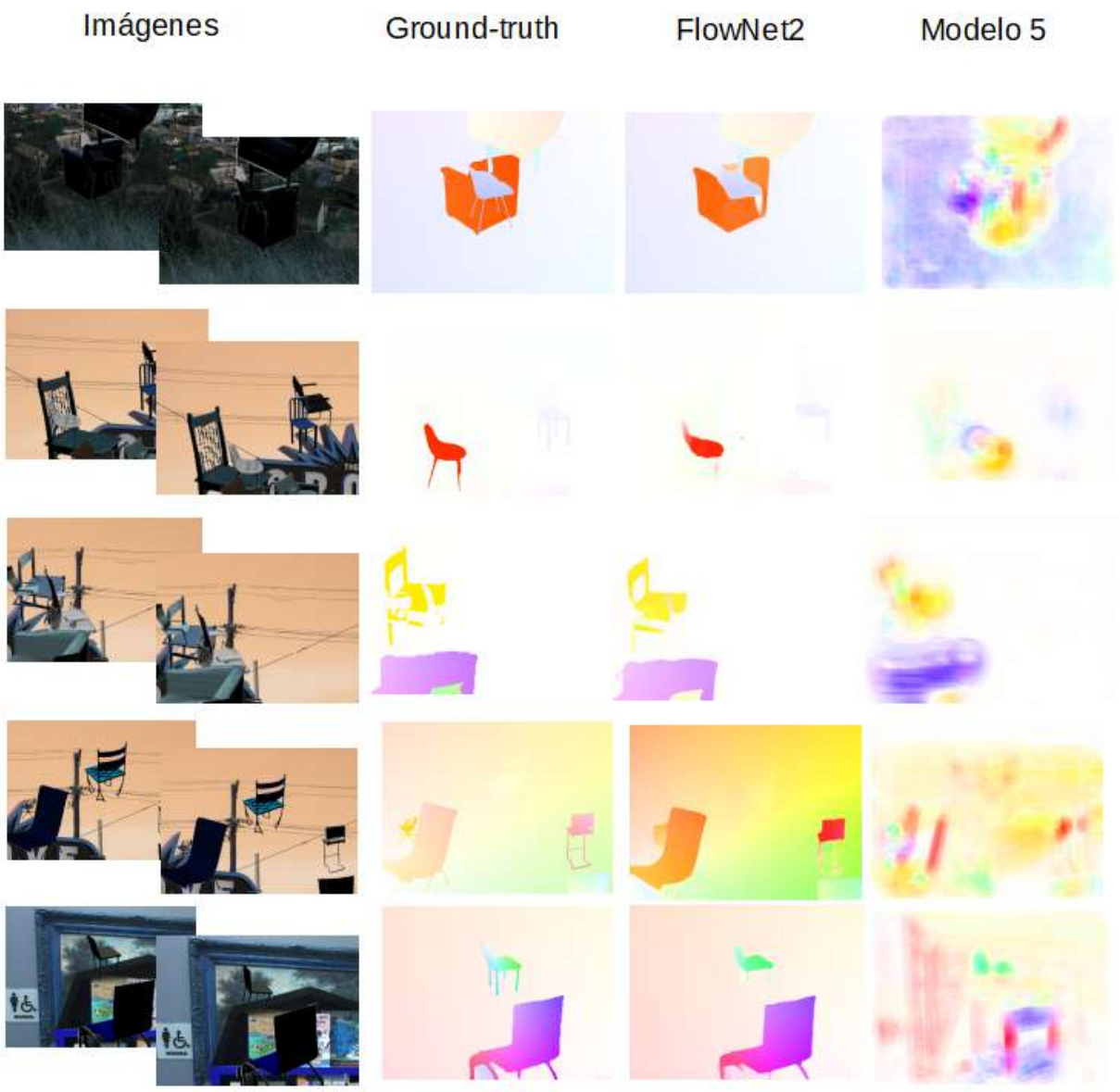


Figura 4.2.7: Campos de flujo resultantes de cada modelo. En la primeras 2 columnas se presenta el par de imágenes y el *ground_truth*. En la tercera columna el campo de flujo producido por FlowNet2 y en la última columna el campo de flujo producido por el modelo 5.

4.3. Experimentación (tercera etapa)

Con el propósito de observar más a fondo los comportamientos de aprendizaje de los modelos 7 y 5, se realizó otra serie de experimentos más sencillos, ya que debido a la complejidad de FlyingChairs es difícil analizar los resultados de manera objetiva. La siguiente serie de experimentos consiste en la creación de diversos conjuntos de datos (o ”dataset” por su nombre en inglés) sencillos. Estos dataset contienen un rectángulo colocado aleatoriamente con diversos desplazamientos aleatorios, con los que se varia la complejidad. Todas las imágenes generadas son de 150x150 de ancho y alto (en pixeles). En el Algoritmo 3 se observa de manera general los pasos para generar los ejemplos. En la Tabla 4.3.1 se describe cada uno de los dataset generados.

Generar un conjunto Θ de ángulos aleatorios;
for $\theta \in \Theta$ **do**
    Generar imagen de $W \times H$ ancho y alto;
    Generar un rectangulo de $w \times h$ ancho y alto;
    Posicionar el rectangulo en una posición (x, y) aleatoria de la imagen;
    Generar un dezplazamiento aleatorio d;
    Generar el campo de flujo f de acuerdo al desplazamiento d y ángulo θ;
    Generar la nueva imagen con respecto al flujo f;
end

Algoritmo 3: Generación de datasets de rectángulos.

Para realizar la experimentación en esta sección se utilizaron 3000 ejemplos de entrenamiento y 600 de prueba. Se entrenó cada modelo (solo el 5 y 7) con los dataset por separado, por lo que se generaron 18 variaciones de resultados. Todos los entrenamientos fueron hechos con el optimizador Adam con una tasa de aprendizaje de $0{,}0001$ y tamaño de *batch* de 20, durante 10 épocas. Los resultados del promedio y desviación estándar del AEPE con el conjunto de prueba se observan en la Figura 4.3.1, y de manera numérica en la Tabla 4.3.2. También se presentan algunos de los resultados visuales con cada dataset en las tablas 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.7, 4.3.8, 4.3.9, 4.3.10, 4.3.11. Es de observarse que la experimentación con los DS7 y DS8 tienen un fallo en su generalización, por lo que las predicciones son erróneas. La característica que tienen estos dataset es que los fondos varían en color de manera aleatoria. Mientras que en la experimentación con el DS9 (Tabla 4.3.11), cuya característica es el desplazamiento más grande, solo el modelo 5 tiene problemas en la generalización.

Dataset	Descripción
Dataset 1 (DS1)	Consiste en imágenes de fondo blanco y un rectángulo de 15x15 color negro, los desplazamientos varían de $[-20,20]$. Véase el ejemplo en la Figura 4.3.2.
Dataset 2 (DS2)	Consiste en imágenes de fondo blanco y un rectángulo de 60x60 color negro, los desplazamientos varían de $[-20,20]$. Véase el ejemplo de la Figura 4.3.3.
Dataset 3 (DS3)	Consiste en imágenes de fondo blanco y un rectángulo color negro de tamaño aleatorio cuyo ancho y alto varia de $[5,40]$, los desplazamientos varían de $[-20,20]$. Véase el ejemplo de la Figura 4.3.4.
Dataset 4 (DS4)	Consiste en imágenes con las mismas características que el DS1 pero el color del rectángulo es aleatorio. Véase el ejemplo de la Figura 4.3.5.
Dataset 5 (DS5)	Consiste en imágenes con las mismas características que el DS2 pero el color del rectángulo es aleatorio. Véase el ejemplo de la Figura 4.3.6.
Dataset 6 (DS6)	Consiste en imágenes con las mismas características que el DS3 pero el color del rectángulo es aleatorio. Véase el ejemplo de la Figura 4.3.7.
Dataset 7 (DS7)	Consiste en imágenes con las mismas características que el DS1 pero el color del rectángulo y fondo de la imagen es aleatorio. Véase el ejemplo de la Figura 4.3.8.
Dataset 8 (DS8)	Consiste en imágenes con las mismas características que el DS2 pero el color del rectángulo y fondo de la imagen es aleatorio. Véase el ejemplo de la Figura 4.3.9.
Dataset 9 (DS9)	Consiste en imágenes con las mismas características que el DS3 pero los desplazamientos están en el rango $[-40,-20] \cup [20,40]$.

Tabla 4.3.1: Tabla de descripciones de los dataset generados.

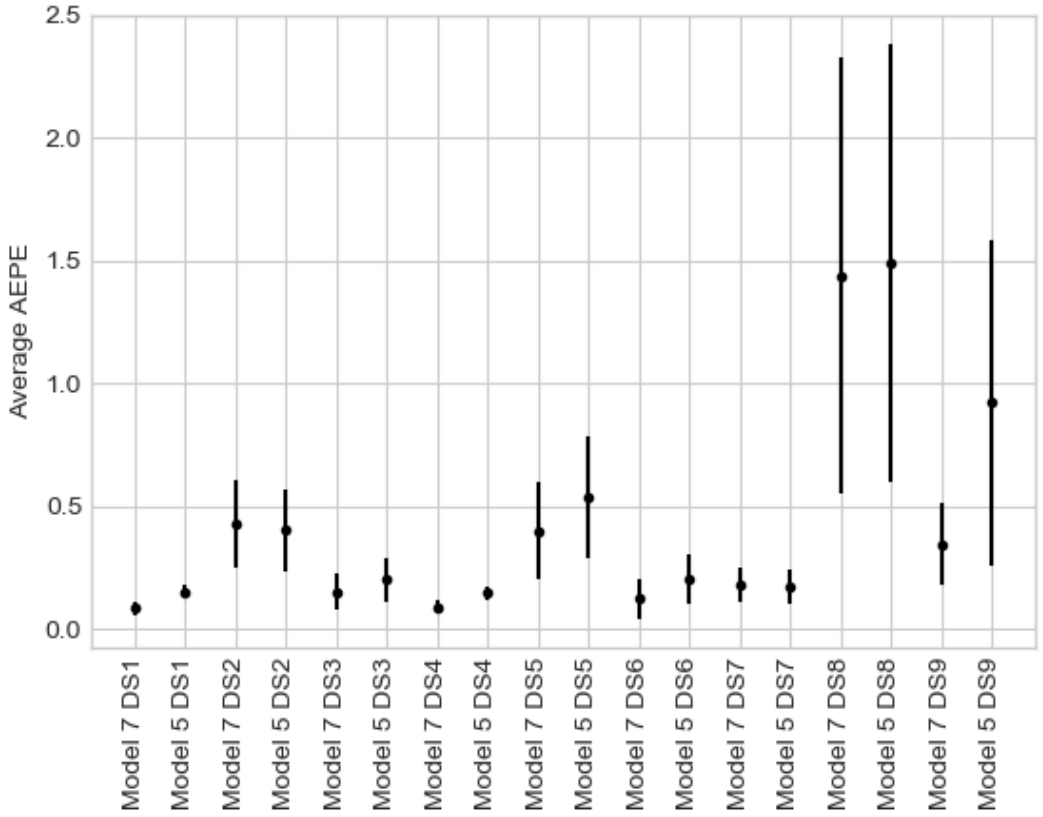
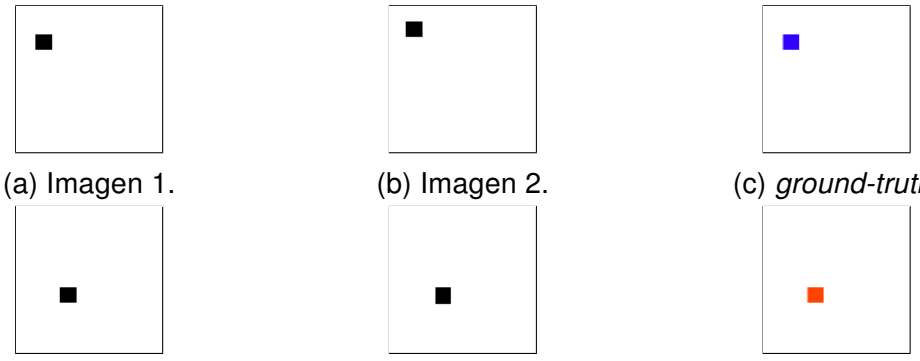


Figura 4.3.1: Resultados con el conjunto de prueba de cada uno de los modelos entrenados con los dataset de rectángulos.



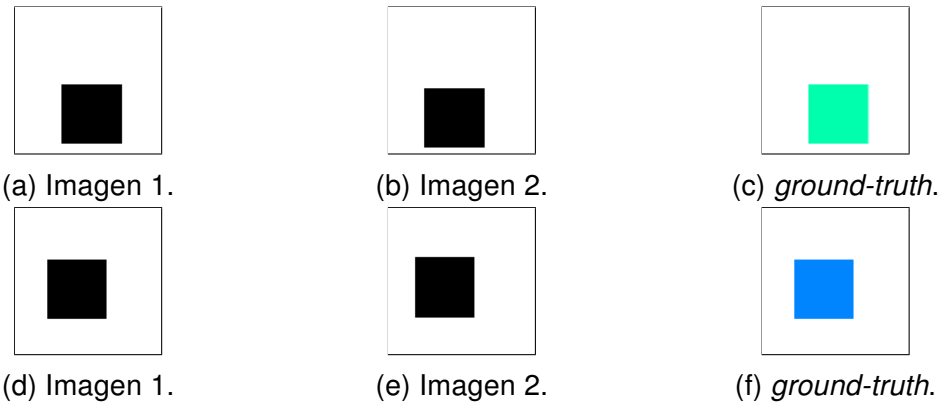
(a) Imagen 1. (b) Imagen 2. (c) *ground-truth*.

(d) Imagen 1. (e) Imagen 2. (f) *ground-truth*.

Figura 4.3.2: Ejemplos del dataset de rectángulos 1.

Modelo	Media	Desviación estándar
Model 7 (DS1)	0.08304	0.02930
Model 5 (DS1)	0.15174	0.02969
Model 7 (DS2)	0.42705	0.17534
Model 5 (DS2)	0.40336	0.16650
Model 7 (DS3)	0.15182	0.07266
Model 5 (DS3)	0.20010	0.09069
Model 7 (DS4)	0.08752	0.02839
Model 5 (DS4)	0.14638	0.02781
Model 7 (DS5)	0.39685	0.19750
Model 5 (DS5)	0.53430	0.24904
Model 7 (DS6)	0.12117	0.07863
Model 5 (DS6)	0.20224	0.10196
Model 7 (DS7)	0.17874	0.06847
Model 5 (DS7)	0.17185	0.06735
Model 7 (DS8)	1.43850	0.89077
Model 5 (DS8)	1.49058	0.89256
Model 7 (DS9)	0.34576	0.16721
Model 5 (DS9)	0.92021	0.66015

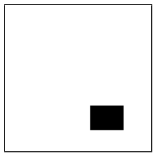
Tabla 4.3.2: Tabla de resultados de los modelos 5 y 7 con todas las variaciones de dataset de rectángulos.



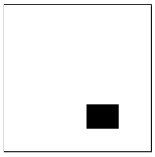
(a) Imagen 1. (b) Imagen 2. (c) *ground-truth*.

(d) Imagen 1. (e) Imagen 2. (f) *ground-truth*.

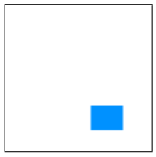
Figura 4.3.3: Ejemplos del dataset de rectángulos 2.



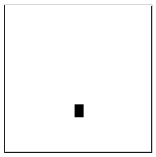
(a) Imagen 1.



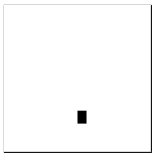
(b) Imagen 2.



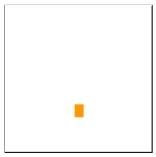
(c) *ground-truth*.



(d) Imagen 1.

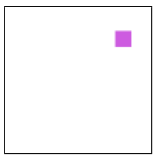


(e) Imagen 2.

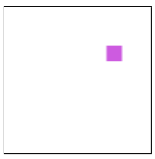


(f) *ground-truth*.

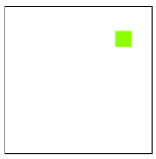
Figura 4.3.4: Ejemplos del dataset de rectángulos 3.



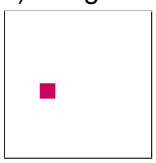
(a) Imagen 1.



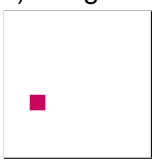
(b) Imagen 2.



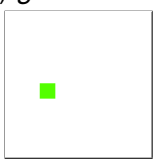
(c) *ground-truth*.



(d) Imagen 1.

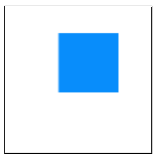


(e) Imagen 2.

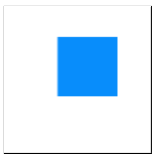


(f) *ground-truth*.

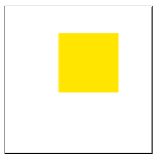
Figura 4.3.5: Ejemplos del dataset de rectángulos 4.



(a) Imagen 1.

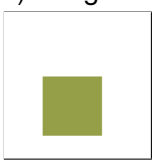


(b) Imagen 2.

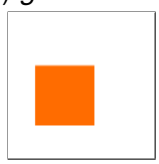


(c) *ground-truth*.

(d) Imagen 1.

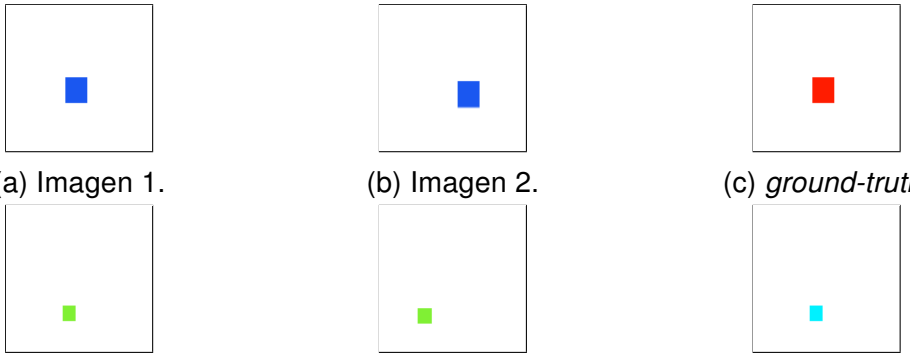


(e) Imagen 2.



(f) *ground-truth*.

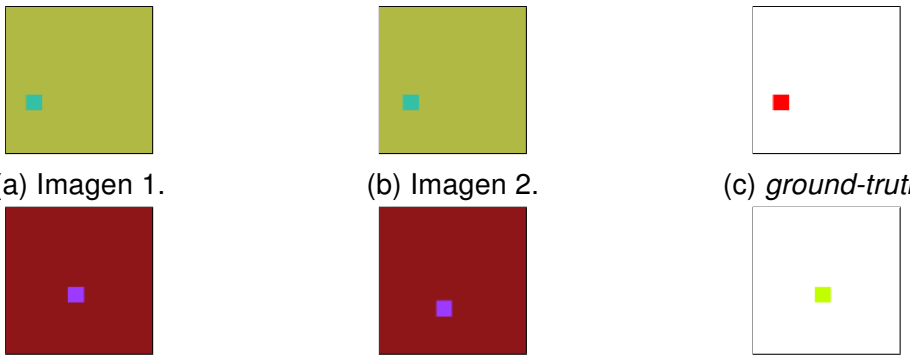
Figura 4.3.6: Ejemplos del dataset de rectángulos 5.



(a) Imagen 1. (b) Imagen 2. (c) *ground-truth*.

(d) Imagen 1. (e) Imagen 2. (f) *ground-truth*.

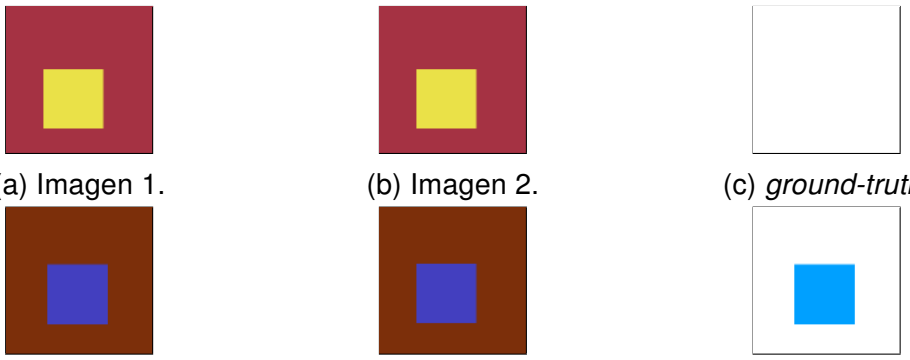
Figura 4.3.7: Ejemplos del dataset de rectángulos 6.



(a) Imagen 1. (b) Imagen 2. (c) *ground-truth*.

(d) Imagen 1. (e) Imagen 2. (f) *ground-truth*.

Figura 4.3.8: Ejemplos del dataset de rectángulos 7.



(a) Imagen 1. (b) Imagen 2. (c) *ground-truth*.

(d) Imagen 1. (e) Imagen 2. (f) *ground-truth*.

Figura 4.3.9: Ejemplos del dataset de rectángulos 8.

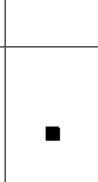
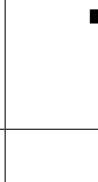
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.12|0.20
3051						0.11|0.17
3052						0.07|0.14
3053						0.06|0.12
3054						0.13|0.20
3055						0.08|0.15

Tabla 4.3.3: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS1.

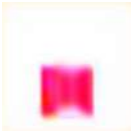
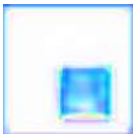
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.51|0.50
3051						0.22 | 0.24
3052						0.70|0.72
3053						0.47|0.47
3054						0.42|0.35
3055						0.18|0.15

Tabla 4.3.4: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS2.

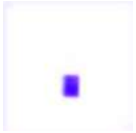
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.11|0.15
3051						0.15|0.18
3052						0.18|0.25
3053						0.36|0.39
3054						0.45|0.47
3055						0.15|0.21

Tabla 4.3.5: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS3.

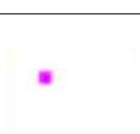
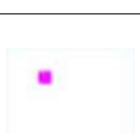
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.10|0.15
3051						0.12|0.18
3052						0.10|0.15
3053						0.09|0.14
3054						0.07|0.13
3055						0.10|0.14

Tabla 4.3.6: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS4.

No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.37|0.60
3051						0.31|0.37
3052						0.43|0.65
3053						0.19|0.24
3054						0.60|0.82
3055						0.69|0.79

Tabla 4.3.7: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS5.

No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.12|0.21
3051						0.06|0.13
3052						0.09|0.15
3053						0.47|0.71
3054						0.14|0.23
3055						0.09|0.15

Tabla 4.3.8: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS6.

No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.22|0.22
3051						0.11|0.10
3052						0.06|0.06
3053						0.24|0.23
3054						0.19|0.17
3055						0.10|0.09

Tabla 4.3.9: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS7.

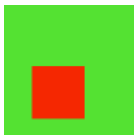
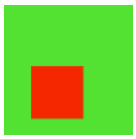
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.21|0.20
3051						3.19|3.44
3052						2.13|2.61
3053						2.91|3.15
3054						2.53|2.82
3055						2.30|2.40

Tabla 4.3.10: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS8.

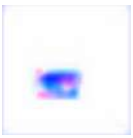
No.	Imagen 1	Imagen 2	*Ground-truth*	Modelo 7	Modelo 5	AEPE (M7|M5)
3050						0.38|0.84
3051						0.26|0.62
3052						0.56|0.76
3053						0.25|0.40
3054						0.14|0.34
3055						0.47|1.86

Tabla 4.3.11: Resultados de estimación de flujo de ambos modelos con algunos ejemplos del conjunto de prueba de DS9.

Capítulo 5

Conclusiones y trabajo futuro

El presente trabajo tuvo como objetivo investigar y proponer un modelo de red neuronal convolucional, la cual cuenta con la característica de tener convolución dilatada, y cuya tarea era dar una inferencia del flujo óptico. Durante el proceso de investigación se propusieron diversos modelos que puedan ser de utilidad y mediante un proceso de experimentación se fueron descartando y modificando modelos de CNNs. Durante este proceso de experimentación notamos diversos factores que son pieza clave en la manera en que estos modelos deben ser propuestos. Como hipótesis, se propuso que el campo receptivo de la red convolucional es importante para generar resultados más aproximados y de cierto modo esto es cierto. Si observamos los resultados del modelo 7 4.2.6 y el modelo 5 4.2.7 en el último ejemplo podemos observar que los gradientes predichos por el modelo 7 tienen más precisión que los resultados del modelo 5, y esto es porque su campo receptivo es aún mayor debido a que cuenta con convoluciones dilatadas, véase la figura 5.0.1. Sin embargo, a pesar de contar con un campo receptivo amplio puede que sea insuficiente aún para contrastar los modelos del estado del arte. Con respecto a ese aspecto se puede concluir que la convolución dilatada tiene un mejor desempeño para distribuir el campo receptivo de las neuronas y dar una predicción más certera.

Haciendo una observación más de cerca sobre las Figuras 4.1.4 y 4.1.6 se observan unos comportamientos muy ruidosos sobre las gráficas de costo tanto en prueba como en validación, que es más observable en la primera Figura. Estos comportamientos pueden deberse a la cantidad de ejemplos usados para entrenar. Puesto que en la segunda Figura, dicha anteriormente, se nota menos debido a que en esa experimentación el número de ejemplos de entrenamiento se aumento a 200. Con lo anterior, además de hacer notar la importancia de número de ejemplos es de notar que importa mucho ciertos ejemplos en particular. Es decir, algunos ejemplos de entrenamiento

son clave para que los pesos se acoplen mejor y tengan una mejor generalización, esto puede observarse en la Figura 4.1.4 a), puesto que en usa partición en especifico el comportamiento es menos errático. Una pregunta que puede surgir de esto puede ser ¿que ejemplos y en que orden se debe entrenar la red para que tengan un rendimiento mejor y un tiempo de entrenamiento menor?

A pesar de los múltiples experimentos hechos con diversas configuraciones de hyper-parámetros no se cuenta con la capacidad de generalización de predicción suficiente para que pueda compararse con el estado del arte. Durante estos experimentos con los hyper-parámetros se llegó a concluir que un tamaño de batch suficiente es de 20, mientras que una tasa de aprendizaje óptima es de $0{,}0001$. En cuanto al número de ejemplos notamos que usar más de 5 mil ejemplos no tenía un beneficio en la predicción. Lo que nos llevaría a que debe haber un factor mucho más importante en la generalización de los modelos. Dicho factor que hasta el momento se ha concluido que es importante, y los modelos que se propusieron no cuentan, es la reducción de dimensiones en una pirámide de resolución. Esto se puede realizar en dos estrategias que se han visto en el estado del arte, la primera son CNNs independientes en cada resolución como en spynet [30], y la segunda es una estructura de red que implícitamente la genere como en [12] donde usan el *stride* de 2 para reducir la dimensión en factores aproximados de 2 y es por eso que la dimensión reduce drásticamente, por lo que los mapas de características se enfocan en detalles de más baja resolución. Otro aspecto importante encontrado durante la experimentación es la elección de la función de costo. En nuestros experimentos, notamos que usar la función de costo L1 (2.3.4) es más estable que usar la función EPE (2.5.4). Durante esta investigación se logró diseñar un modelo de red cuyo número de parámetros es mucho menor a FlowNet ya que tiene cerca de 3 millones de parámetros mientras que cada modulo de FlowNet cuenta con cerca de 30 millones. Por lo que la diferencia en pesos es muy significativa, y por tanto el cómputo requerido para realizar una predicción es menor.

En la última etapa de experimentación se descubrió de manera objetiva la aportación de incluir la convolución con dilatación de manera distribuida entre las capas de la red. Es de notar que en la mayor parte de los resultados (en la Figura 4.3.1) el modelo 7 mantiene un costo menor a pesar de tener menos de la mitad de pesos que el modelo 5, además mantiene una mejor estabilidad para desplazamientos más largos (véase la Tabla 4.3.11). El único inconveniente visto en los resultados es cuando se tiene fondos aleatorios en los datasets de rectángulos. No se tiene una explicación objetiva con respecto a este problema, sin embargo una posible explicación es que al variar el color de los fondos el modelo no podría decir si el rectángulo esta en movimiento o todo el

fondo se movió, o por otra parte no se pudo generalizar los colores en la red.

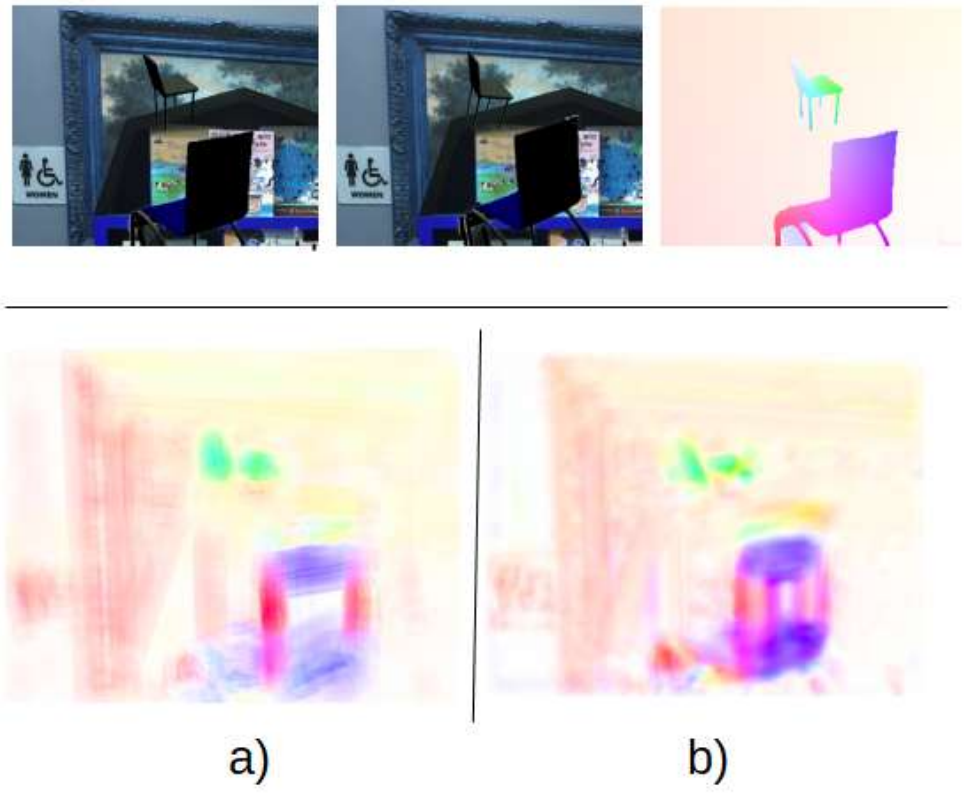


Figura 5.0.1: Observación de diferencias entre el modelo 5 y modelo 7, en la parte de arriba de la imagen observamos el ejemplo junto a su *ground-truth*. En a) es el resultado del modelo 5, en b) el resultado del modelo 7. Podemos observar que el modelo 7 tiene mejor predicción del movimiento debido a su campo receptivo, aún cuando la silla tiene regiones homogéneas.

5.1. Trabajo futuro

Como se mencionó anteriormente, un aspecto que probablemente es fundamental para mejorar la inferencia del flujo óptico es implementar una estrategia ***coarse-to-fine*** en el modelo de CNN. Y esta puede ser implementada de dos maneras, como ya se mencionó. Una posible manera directa de implementar es usar *stride* de dos en ciertas capas de convolución para generar un *downsampling* más efectivo y cuyas dimensiones desciende en factores aproximados de 2. Sin embargo, como se argumentará posteriormente, hacer esto llevaría a ciertos artefactos en las dimensiones de los ejemplos, por lo que los radios de aspectos en distintas salidas de etapas de convolución no

concordarán. Dicho lo anterior, es importante pensar en un diseño que pueda manejar estas problemáticas, de manera que pueda mezclarse el modelo general propuesto con la estrategia ***coarse-to-fine***. En este trabajo solo se utilizaron pocos ejemplos de entrenamiento, además que durante la experimentación se encontró que usar más de 5000 ejemplos no tenía una mejoría en la predicción. Sin embargo, puede que ser que re-estructurando el modelo se pueda usar *data augmentation* para incrementar el número de ejemplos mediante transformaciones afín y cambios en los brillos del ejemplo original. Con respecto a las funciones de costo, como se mencionó anteriormente, L1 tuvo mejor estabilidad. Sin embargo, existen otras funciones que podrían ayudar. Así como EPE trata de medir la diferencia de magnitud del vector de error, también se podría medir la diferencia del ángulo del vector conocido como *Angular Point Error* (o APE). Más aún, podría generarse una combinación lineal de dichas funciones de error con el objetivo de centrarse en minimizar ciertas características individuales para una mejor inferencia.

5.2. Discusión

En el trabajo presente se propusieron distintos modelos de redes neuronales convolucionales para inferir el flujo óptico. Sin embargo, dado los resultados vistos durante toda la experimentación, aún se tiene una falta de generalización en los modelos con capas de convolución dilatada para que puedan ser comparables con los resultados del estado del arte. Durante todo el proceso de investigación y experimentación existen ciertas observaciones que pueden tomarse en cuenta para continuar investigando este tema. Un factor importante es la etapa de **coarse-to-fine**, con respecto al tema de flujo óptico, nos referimos a este concepto a que es más fácil hacer inferencia de un campo de flujo en una resolución menor. Lógicamente es fácil de pensar que si queremos calcular el campo de flujo de una imagen muy pequeña (o baja resolución) el resultado será más preciso. Este método de **coarse-to-fine** es bien argumentado analíticamente en [26]. Una vez hecho inferencia de un campo de flujo en resoluciones pequeñas se propaga dicha estimación hacia una resolución mayor para mejorar la precisión de la inferencia en la resolución actual. En distintos trabajos de inferencia de flujo óptico como [17], se fundamentan generalmente de esta metodología *coarse-to-fine*. Sin embargo, escalar las imágenes para ir a una resolución distinta puede hacerse de diversas maneras. Una manera comúnmente usada para re-dimensionar las imágenes en un factor k proporcional, es dividir la resolución en un factor de k donde generalmente se usa $k = 2$. Con estas múltiples operaciones de re-dimensionado se

forma algo conocido como **pirámide multi-resolución**. Cuando trabajamos con redes neuronales convolucionales, este método se puede hacer explicito como en el trabajo de [30], donde existe una única CNN especializada para cada resolución. Y las resoluciones son perfectamente divisibles en cada etapa de la pirámide. Cuando pensamos en una estructura de CNN que integre *coarse-to-fine* directamente en el modelo (como en FlowNet [12]), no es directamente aplicable debido a que **la convolución no es una operación preservadora del radio de aspecto** en una imagen. Dicho lo anterior, cuando queremos hacer un re-dimensionado del campo de flujo para que concuerde con cierta profundidad de la red, notaremos que el radio de aspecto de dicho campo de flujo no concuerda con el original. En el trabajo de [12] se observó que hacer operar con los ejemplos de entrenamiento modificando su dimensión no trae consecuencias graves. Sin embargo, hacer una modificación de este tipo en el campo de flujo original implica cambiar por completo los vectores de flujo ya que las dimensiones de la nueva imagen re-dimensionada son diferentes. Es debido a esto que no se pensó en implementar una etapa *coarse-to-fine* directamente en los modelos propuestos, a excepción del modelo 3 3.2.3, sin embargo, durante la experimentación de este trabajo observamos que hacer esto llevó a peores resultados (4.2.1). Con respecto a la reducción de dimensión en factores proporcionales, en FlowNet es de observarse que el componente fundamental para hacer esto es aumentar el *stride*. Puesto que la formula para calcular la dimensión de salida de una convolución para cuando se tiene un *stride* de 2 (véase la relación 6), tiene el efecto de reducir las dimensiones de entrada en aproximadamente menos de la mitad (dependiendo del tamaño del filtro). Por tanto, el principal componente para generar una pirámide cuya resolución descienda en factores aproximados de la mitad es usar un *stride* de 2 cada cierta capa de convolución. Con respecto a los tiempos para realizar los experimentos se encontró que debido al tipo de problema y los tamaños de los ejemplos se necesitaba demasiado tiempo para realizar un entrenamiento. Mientras más parámetros se tenga en un modelo de CNN entonces es más tardado el entrenamiento. Un entrenamiento con 400 ejemplos durante 10 épocas, en general, tardaba cerca de 3 días en realizarse, bajo las condiciones de hardware con las que se contaron. Estás circunstancias deben ser tomadas importantes cuando se realiza experimentación para modelos de CNN que resuelvan el problema de inferir el flujo óptico, puesto que una metodología más adecuada podría ahorrar tiempo de entrenamiento y experimentación. Este trabajo de investigación se desarrolló con la idea de usar modelos de CNNs apilados como en los trabajos de [21] y [39]. En este tipo de metodología se tienen un modelo inicial que posteriormente pasa su predicción a un modelo siguiente cuyo objetivo es refinar la predicción. Dicho méto-

do puede tener dos variantes: La primera es entrenar la primera CNN, fijar los pesos y luego entrenar la siguiente CNN, que recibe la predicción anterior. La segunda variante consiste en que no se fijan los pesos de las CNNs pre-entrenadas, sino que al momento de entrenar la última CNN se entrena en conjunto con las redes previas y todas las CNN en serie actualizan sus pesos. Es evidente que la segunda variante tardará más en entrenarse. Además, de acuerdo al trabajo de FlowNet2 [21] se argumenta que es mejor fijar los pesos de las CNNs que han sido entrenadas previamente en el modelo en serie. Sin embargo, no se contó con el tiempo suficiente para experimentar con estas variantes a pesar de que el código para hacer pruebas fue preparado para esto. Acerca de la plataforma de machine learning elegida, en este caso TensorFlow, se encontraron diversas problemáticas. Con dichas problemáticas, tal vez sea conveniente cambiar a una plataforma más estable y bien documentada como Pytorch, ya que cuenta con más herramientas y funcionalidades para facilitar la definición de estructura e implementación de los modelos.

Apéndice A

Repositorio de código

https://bitbucket.org/IMart302/flowdcnn_thesis/src/master/

Bibliografía

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[3] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 4015–4023, 2015.

[4] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[5] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[6] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European*

Conf. on Computer Vision (ECCV), Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

[7] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, volume 1, page 7, 2017.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[9] Suduan Chen. Detection of fraudulent financial statements using the hybrid data mining approach. *SpringerPlus*, 5(1):89, 2016.

[10] Zhuoyuan Chen, Hailin Jin, Zhe Lin, Scott Cohen, and Ying Wu. Large displacement optical flow from nearest neighbor fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2443–2450, 2013.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[13] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[17] Kaiming He and Jian Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 111–118. IEEE, 2012.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8981–8989, 2018.

[20] Tae Hyun Kim, Hee Seok Lee, and Kyoung Mu Lee. Optical flow via locally adaptive fusion of complementary data costs. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[21] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[22] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 614–630, 2018.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[28] Santanu Pattanayak, Pattanayak, and Suresh John. *Pro Deep Learning with TensorFlow*. Springer, 2017.

[29] S. Pendleton, T. Uthaicharoenpong, Z. J. Chong, Guo Ming James Fu, B. Qin, Wei Liu, Xiaotong Shen, Zhiyong Weng, C. Kamin, M. A. Ang, L. T. Kuwae, K. A. Marczuk, H. Andersen, Mengdan Feng, G. Butron, Z. Z. Chong, M. H. Ang, E. Frazzoli, and D. Rus. Autonomous golf cars for public trial of mobility-on-demand service. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1164–1171, Sep. 2015.

[30] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4161–4170, 2017.

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[32] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1164–1172, 2015.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[36] Deqing Sun, Erik B. Sudderth, and Michael J. Black. Layered image motion with explicit occlusions, temporal consistency, and depth ordering. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2226–2234. Curran Associates, Inc., 2010.

[37] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[38] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *The International Journal of Robotics Research*, 26(9):889–916, 2007.

[39] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[40] Li Xu, Jiaya Jia, and Yasuyuki Matsushita. Motion detail preserving optical flow estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1744–1757, 2011.

[41] S. Z. Yong, M. Q. Foo, and E. Frazzoli. Robust and resilient estimation for cyber-physical systems under adversarial attacks. In *2016 American Control Conference (ACC)*, pages 308–315, July 2016.

[42] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.