UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN CIENCIAS

Aprendizaje profundo para la detección de árboles en imágenes aéreas de un cultivo de higo capturadas con un dron

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA:
Juan Luis Torres Olivares

DIRECTOR DE TESIS:
Dr. Juan Manuel Rendón Mancha

CO-DIRECTOR DE TESIS:
Dr. Jorge Alberto Fuentes Pacheco

CUERNAVACA, MORELOS NOVIEMBRE 2019

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN CIENCIAS

Aprendizaje profundo para la detección de árboles en imágenes aéreas de un cultivo de higo capturadas con un dron

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA:

Juan Luis Torres Olivares

DIRECTOR DE TESIS:
Dr. Juan Manuel Rendón Mancha

CO-DIRECTOR DE TESIS:
Dr. Jorge Alberto Fuentes Pacheco

SINODALES:
Dr. Juan Manuel Rendón Mancha
Dr. Jorge Alberto Fuentes Pacheco
Dr. Jorge Hermosillo Valadez
Dr. Porfirio Juárez López
Dr. Edgar Francisco Román Rangel

CUERNAVACA, MORELOS NOVIEMBRE 2019

Lista de jurado

- Presidente: Dr. Jesús Igor Heberto Barahona Torres
- Secretario: Dr. Jorge Hermosillo Valadez
- Vocal: Dr. Paúl Hernández Herrera
- Suplente: Dr. Juan Manuel Rendón Mancha
- Suplente: Dr. Jorge Alberto Fuentes Pacheco

Publicación relacionada con la tesis

Artículo:

- Jorge Fuentes-Pacheco, Juan Torres-Olivares, Edgar Roman-Rangel, Salvador Cervantes, Porfirio Juarez-Lopez, Jorge Hermosillo-Valadez y Juan Manuel Rendón-Mancha. Fig Plant Segmentation from Aerial Images Using a Deep Convolutional Encoder-Decoder Network. *Remote Sensing*, (revista indizada, cuartil Q1), 11(10), 2019

Agradecimientos

Agradecemos al Laboratorio Nacional de Supercómputo del Sureste de México perteneciente a la red de laboratorios nacionales CONACYT, por los recursos computacionales, el apoyo y la asistencia técnica.

Al CONACYT por ser beneficiado de una beca de manutención para mis estudios de maestría en el programa educativo Maestría en Ciencias inscrito en el Programa Nacional de Posgrados de Calidad. El programa educativo es de la Universidad Autónoma del Estado de Morelos y la convocatoria para la obtención de beca fue la *Convocatoria de Becas Nacionales 2017 Segundo Periodo*.

Resumen

En este trabajo de tesis se realiza una segmentación automática de árboles de higo en imágenes aéreas por medio de una red neuronal. Se capturaron fotografías de una hectárea de árboles de higo cultivados en campo abierto con una cámara montada en un dron. El dron usado es el *Phantom 4* con una cámara óptica integrada. De las imágenes capturadas se seleccionaron diez fotografías, se recortaron de tamaño 2000×1500 pixeles y se segmentaron manualmente. Estas imágenes se recortaron de tamaño 128×128 pixeles dejando un traslape del $70\,\%$ y se pasaron como entrada a un programa hecho en Python en el que se programó una red neuronal convolucional profunda (CNN por sus siglas en inglés). La CNN realiza operaciones de tipo: Convolución, *Maxpooling* y *Upsampling* con la finalidad de segmentar en una imagen las regiones que pertenecen a hojas de árbol de higo. Se realizaron pruebas de validación cruzada, reconstrucción de la imagen de tamaño original a partir de los recortes segmentados por la CNN, se probó el desempeño de la CNN con diferentes tamaños de recorte y se comparó el desempeño de la CNN propuesta contra SegNet-Basic. Finalmente, se muestran los resultados obtenidos de las pruebas realizadas.

Este trabajo proporciona un comienzo significativo para otras tareas de procesamiento de imágenes que se aplican a la agricultura de precisión como lo son: estimar el estado de salud de la planta, detección de plagas o estimación del área foliar.

Además de lo anterior hacemos público el código fuente de la CNN así como las imágenes capturadas con el dron y las imágenes recortadas y segmentadas manualmente para facilitar las comparaciones entre diferentes algoritmos.

Índice general

Capítulo 1

Introducción

1.1. Planteamiento del problema

El problema a tratar es la segmentación de regiones correspondientes a copas de árboles de higo mediante aprendizaje profundo (*deep learning*). En específico, usando una Red Neuronal Convolucional profunda (CNN por sus siglas en inglés). La entrada de la red son imágenes aéreas de gran resolución tomadas con una cámara montada en un dron. El dron ejecuta una misión de vuelo automática sobre una parcela de higo cultivada en campo abierto.

Las imágenes poseen características que hacen difícil segmentar las regiones correspondientes a árboles de higo. Entre estas características están los diferentes niveles de luminosidad debido a las cambiantes condiciones de la luz natural mientras se realizó la misión de vuelo para tomar las imágenes. Además de que los árboles proyectan una sombra sobre el suelo. Otro problema se presenta cuando la sombra de un árbol grande de higo se proyecta sobre un árbol menos alto. También el suelo presenta un problema para hacer segmentación en las imágenes ya que el suelo no está bien diferenciado del cultivo a causa de las sombras proyectadas. Si se pretendieran usar técnicas clásicas de procesamiento digital de imágenes para realizar una segmentación en una imagen con sombras sería muy complicado. Puede darse el caso en que las sombras se segmenten como el objeto de interés. Otro caso es cuando una sombra se proyecta sobre el cultivo cambiando su color típicamente verde a un color oscuro y el cultivo quede segmentado como objeto que no es de interés.

Otra característica que dificulta la segmentación en las imágenes capturadas son los elementos ajenos al cultivo y al suelo pero que son comunes en cultivos a campo abierto, como son rocas, maleza y objetos usados por los agricultores, además de colores cambiantes en la tierra causados por humedad y caminos de terracería.

1.2. Hipótesis

Una red neuronal convolucional profunda entrenada adecuadamente será capaz de segmentar las hojas de higo en las imágenes RGB capturadas con un dron. Se ha demostrado en el estado del arte que una red neuronal profunda tiene un buen desempeño en tareas de clasificación. Para el caso de la segmentación se quiere clasificar a nivel de pixel los pixeles que pertenecen a árbol de higo y los pixeles que no pertenecen a árbol de higo.

1.3. Justificación

Novedades en este trabajo:

- No se usarán imágenes satelitales, en cambio se usan imágenes aéreas obtenidas con una cámara RGB montada en un dron. Este tipo de imágenes contienen más información visual que la imagen satelital, ya que el dron puede volar a una corta distancia de los cultivos.
- Se usará una cámara óptica a diferencia de otros trabajos sobre segmentación de cultivos que suelen usar cámaras que capturan el infrarrojo cercano o múltiples espectros.
- La red neuronal convolucional profunda que se usará para segmentar no tiene los pesos pre-calculados, dichos parámetros se inicializan y se ajustan a partir de un entrenamiento con nuestro conjunto de datos de entrenamiento (*training set* en inglés). Normalmente se usan redes con pesos ajustados con una gran base de datos pública y se *terminan de entrenar* con las imágenes propias, a esto se le conoce como *transfer learning*.

Segmentar imágenes de cultivos brinda un comienzo muy importante para otras tareas de procesamiento de imágenes que se usan en la agricultura inteligente como lo son: detección de plagas, detección de escasez de nitrógeno, estrés hídrico, entre otros. Para estas tareas es conveniente saber qué es árbol y qué no lo es, por lo que segmentar la imagen a analizar es muy útil.

Por último, es necesario mencionar que en el trabajo de tesis se usarán imágenes propias de cultivos de árboles de higo que son inéditas en el estado del arte a diferencia de otros trabajos sobre segmentación de imágenes con redes neuronales que suelen usar un conjunto de datos (*dataset* en inglés) ampliamente conocido, como por ejemplo el *Pascal VOC dataset*.

1.4. Objetivo general

Segmentar imágenes aéreas de una parcela de higo cultivada a campo abierto utilizando aprendizaje profundo con la finalidad de separar las hojas de higo de otras regiones como suelo o maleza.

1.5. Objetivos particulares

1. Completar una base de datos de imágenes de árboles de higo capturadas con un dron.
2. Entrenar una red neuronal profunda para detectar hojas de árbol de higo en las imágenes obtenidas.

Capítulo 2

Marco teórico

2.1. Justificación del uso de redes neuronales convolucionales profundas

El aprendizaje profundo ha logrado avances importantes en la solución de problemas que anteriormente se consideraban muy difíciles como el reconocimiento de voz y el análisis de textos y ha superado a los enfoques clásicos de solución de problemas, reduciendo tiempos del modelado del problema, pre-procesamiento y post-procesamiento. Además los modelos de redes neuronales se desempeñan muy bien al optimizar el modelo respecto a su objetivo.

La razón de usar redes neuronales convolucionales profundas (CNNs) es que actualmente existen modelos de CNN diseñados para realizar tareas complejas obteniendo buenos resultados. Por ejemplo, las redes neuronales convolucionales para el reconocimiento en imágenes conocidas como AlexNet [11] y VGG16Net [22] han mostrado una exactitud muy alta en la tarea de reconocimiento de imágenes, esta exactitud ha superado a otros métodos como los métodos clásicos de aprendizaje de máquina (*machine learning*) [12].

Antes del uso de redes neuronales profundas, la exactitud del reconocimiento en imágenes había mejorado a un ritmo lento, pero después de la implementación de redes neuronales profundas, la tasa de error en el reconocimiento en imágenes disminuyó del 40 % en 2010 a aproximadamente 7 % en 2014 y este valor sigue disminuyendo. Para tener un punto de comparación, la tasa de reconocimiento del humano es de alrededor del 5 % [16]. Estas redes *aprenden* una representación de los datos, es decir, no solo aprenden a reconocer un objeto sino que también aprenden cuáles son las características importantes que definen la clase a la que pertenece el objeto de interés.

Las técnicas convencionales de aprendizaje de máquina tenían una capacidad limitada para procesar *datos naturales* en su forma original. Durante décadas la creación de un sistema de reconocimiento de patrones o de *machine learning* requirió de una considerable experiencia para diseñar un extractor de características que transformara los datos brutos (como los valores de píxel de una imagen) en una representación abstracta o vector de características adecuado con el cual el sistema de aprendizaje, a menudo un clasificador, pudiera detectar o clasificar patrones de la entrada [12].

El término *profundo* se refiere a aprender representaciones de manera jerárquica [6]. Esta jerarquía permite aprender representaciones complicadas, construyéndolas a partir de otras más simples [13]. A la cantidad de capas de un modelo de aprendizaje profundo se le llama profundidad del modelo [6].

Es muy frecuente el uso de redes neuronales profundas actualmente, por ejemplo, cuando Google introdujo el aprendizaje profundo para su sistema operativo *Android*, logró una reducción del 25 % en el error de reconocimiento de palabras [16].

2.2. *Backpropagation*

Cada capa en una CNN obtiene a su salida una suma ponderada de su entrada con un valor W y pasa esta suma a través de una función no lineal, por ejemplo, la función no lineal ReLU: $salida = ReLU(W \times entrada + b)$. En esta expresión W y b (a W se le conoce también como núcleo o *kernel* y b como sesgo o *bias*) son los pesos o parámetros entrenables para esa capa. Estos pesos contienen la información aprendida por la CNN [6].

Inicialmente los pesos tienen valores aleatorios, en este punto si se realizaran predicciones es muy probable que los resultados difieran mucho de los resultados esperados pero estos valores de peso son un punto de partida. Después de inicializar sigue ajustar los pesos de manera gradual, a esto se le llama entrenamiento (*training*). Este entrenamiento se realiza dentro de lo que se llama ciclo de entrenamiento (*training loop*) que funciona como a continuación se describe [6]:

Repetir estos pasos en un ciclo, tanto como sea necesario:

1. Construir un conjunto de datos de entrenamiento x y sus correspondientes etiquetas y.
2. Dar x como entrada a la CNN (este es un paso llamado *forward pass* o paso hacia adelante) para obtener las predicciones de la red $y_{predicciones}$.
3. Calcular la pérdida de la CNN en todo el conjunto de predicciones para medir la diferencia entre $y_{predicciones}$ y y.
4. Actualizar todos los pesos de la CNN de manera que se reduzca ligeramente la pérdida entre $y_{predicciones}$ y y.

La pérdida se calcula usando las predicciones de la CNN y las etiquetas y calculando un puntaje de la diferencia entre ellos, este puntaje da información de qué tan bien ha predicho la CNN.

Para el último paso sobre cómo actualizar los pesos en la CNN, una buena estrategia es tomar la ventaja de que todas las operaciones que se usan en la CNN son diferenciables para así calcular el gradiente de la pérdida respecto a los pesos y después mover estos pesos en la dirección opuesta del gradiente logrando disminuir la pérdida [6].

En la práctica una CNN consiste en varias funciones encadenadas donde cada función tiene una derivada conocida. Aplicando la regla de la cadena para el cálculo de los valores del gradiente en una CNN da lugar a un algoritmo llamado propagación hacia atrás (*backpropagation* o *Backward pass*) que inicia con el valor de pérdida y avanza hacia atrás aplicando la regla de la cadena para calcular la contribución de cada parámetro al valor de pérdida [6], en la imagen 2.1 se muestra un ejemplo de la propagación hacia atrás.

En la práctica la modificación de parámetros se realiza basada en el valor de pérdida de un lote aleatorio que se pasa en el *forward pass*. Después se calcula el gradiente en el *backward pass* para así actualizar los pesos como se menciona en el punto 4 del *training loop*. Si se actualizan los pesos en la dirección opuesta al gradiente, la pérdida será menor en cada iteración del *training loop*, los pasos para hacer esto se enlistan a continuación [6]:

1. Construir un lote de ejemplos de entrenamiento x y sus correspondientes etiquetas y.
2. Dar el lote x a la red para obtener las predicciones $y_{predicciones}$.
3. Calcular la pérdida de la red en el lote, es decir, medir la discrepancia entre $y_{predicciones}$ y y.
4. Calcular el gradiente de la pérdida respecto a los parámetros de la red, es decir, un *backward pass*.

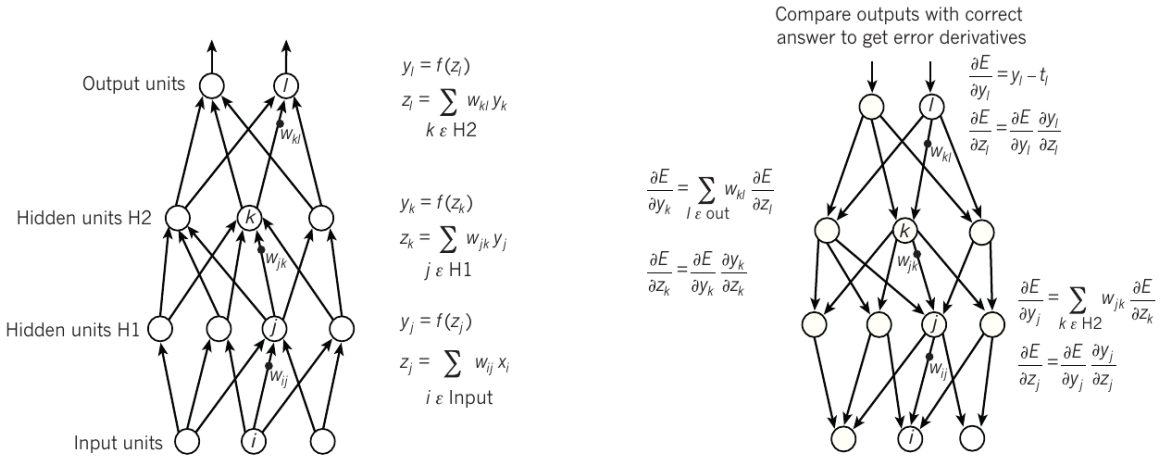


Figura 2.1: *Backpropagation* en redes neuronales profudas. **Izquierda**, *Forward pass*. En cada capa primero se calcula z que es la suma ponderada de los pesos de la capa con las salidas de su capa anterior. Luego se aplica una función no lineal $f(.)$ a z para obtener la salida de la capa. **Derecha**, *Backward pass*. En cada capa oculta se calcula la derivada del error respecto de la salida. Esta derivada es una suma ponderada de la derivada del error respecto a las entradas de las capas superiores (como se puede ver en la expresión $\frac{\partial E}{\partial y_k} = \sum_{l \varepsilon out} w_{kl} \frac{\partial E}{\partial z_l}$). Luego se convierte la derivada del error respecto a la salida en la derivada del error respecto a la entrada multiplicándola por el gradiente de $f(z)$ (como se puede ver en la expresión $\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$). En la capa de salida, la derivada del error se calcula al restar la función de pérdida entre y_l y t_l donde t_l es el resultado deseado. Una vez que se conoce $\frac{\partial E}{\partial z_k}$. La derivada del error de w_{jk} es $y_j \frac{\partial E}{\partial z_k}$. Imagen de [12].

5. Mover los parámetros en la dirección opuesta al gradiente ($W = W - step \times gradiente$). Esto reducirá un poco la pérdida sobre el lote.

Esta manera de modificar los parámetros se llama descenso de gradiente estocástico por mini lotes (*mini-batch stochastic gradient descent* mini-batchSGD). En este método es importante escoger un buen valor de *step*. Si es un valor pequeño, obtener buenos valores de peso en la red tomará muchas iteraciones, si es un valor grande, se podrían posicionar los valores de peso en lugares irrelevantes dentro del espacio de soluciones.

Además existen variantes de SGD, hay por ejemplo SGD con *momentum*, Adagrad, RMSProp, Adadelta y muchos otros, estas variantes se conocen como métodos de optimización u *optimizers* [6].

2.3. Convolución

Las CNNs realizan una convolución que es una operación de filtrado en el dominio espacial. La palabra *filtrado* viene del procesamiento en el dominio de frecuencias, donde filtrar es dejar pasar o rechazar frecuencias. [19]. De manera general la convolución se puede ver como una operación de dos funciones que produce una tercera función que expresa cómo se modifica una función con la otra [13].

La convolución es un proceso muy usado en el filtrado de imágenes, que consiste en: una vecindad, un filtro y una operación de filtrado definida sobre la vecindad, esto crea un pixel con las coordenadas del centro de la vecindad cuyo resultado depende de la operación de filtrado, de este modo una imagen filtrada es generada con el centro del filtro visitando cada pixel de la imagen de entrada [19].

En la tarea de filtrado de imágenes hay dos operaciones importantes, una se llama correlación, definida como:

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

La otra se llama convolución y esta definida como:

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

En ambos casos w es el filtro o *kernel* y f es la imagen [19].

A continuación se muestra un ejemplo de convolución con un filtro y una imagen de tamaño 3x3 con el objetivo de visualizar las posiciones de la imagen y el filtro que intreractúan en la sumatoria.

Operación de convolución con $a = 1$ y $b = 1$

$$w(x,y) * f(x,y) = \sum_{s=-1}^{1} \sum_{t=-1}^{1} w(s,t) f(x-s, y-t)$$

Elementos de una imagen de 3×3 (f)

(x-1, y-1)	(x-1, y)	(x-1, y+1)
(x, y-1)	(x, y)	(x, y+1)
(x+1, y+1)	(x+1, y)	(x+1, y+1)

Elementos del filtro o *kernel* de tamaño 3×3 (w)

(-1, -1)	(-1, 0)	(-1, 1)
(0, -1)	(0, 0)	(0, 1)
(1, -1)	(1, 0)	(1, 1)

Como se puede ver, en la convolución el *kernel* queda girado 180 grados mientras que en la correlación el *kernel* no se rota, para ambos casos después se multiplica punto a punto.

En aprendizaje profundo las convoluciones operan sobre mapas de características que tienen altura, anchura y profundidad. Para una imagen RGB la profundidad es de 3 porque la imagen tiene tres canales de color: rojo, verde y azul. Para el caso de una imagen en blanco y negro la profundidad es de 1 por el nivel de gris [6].

La operación de convolución extrae parches (*patches*) del mapa de características que se dé como entrada y aplica un filtro a todos estos parches, produciendo un mapa de características de salida. La profundidad del mapa de características de salida depende del número de filtros que se usen en la capa convolucional. Esta operación está definida por dos parámetros importantes: El tamaño de los parches y la profundidad. El tamaño de los parches que se extraen de la entrada que son típicamente de tamaño 3×3, 5×5 y 7×7. Usar un tamaño de filtro de 3×3 es la opción más común. La profundidad de la salida del mapa de características es el número de filtros en la capa convolucional [6].

En muchas librerías de redes neuronales es común que esté implementada una correlación pero la llaman convolución y esto se debe a que en el contexto del aprendizaje profundo se aprenden los valores del filtro en el lugar adecuado, además de que en redes neuronales que usan convolución se usan además otras funciones que no dependen de si se rota o no el filtro [13], como se ve en la figura 2.2.

Una peculiaridad de la convolución es que la altura y anchura del resultado de la convolución es menor a la altura y anchura de la entrada. Esto se debe a que no consideran los bordes de la

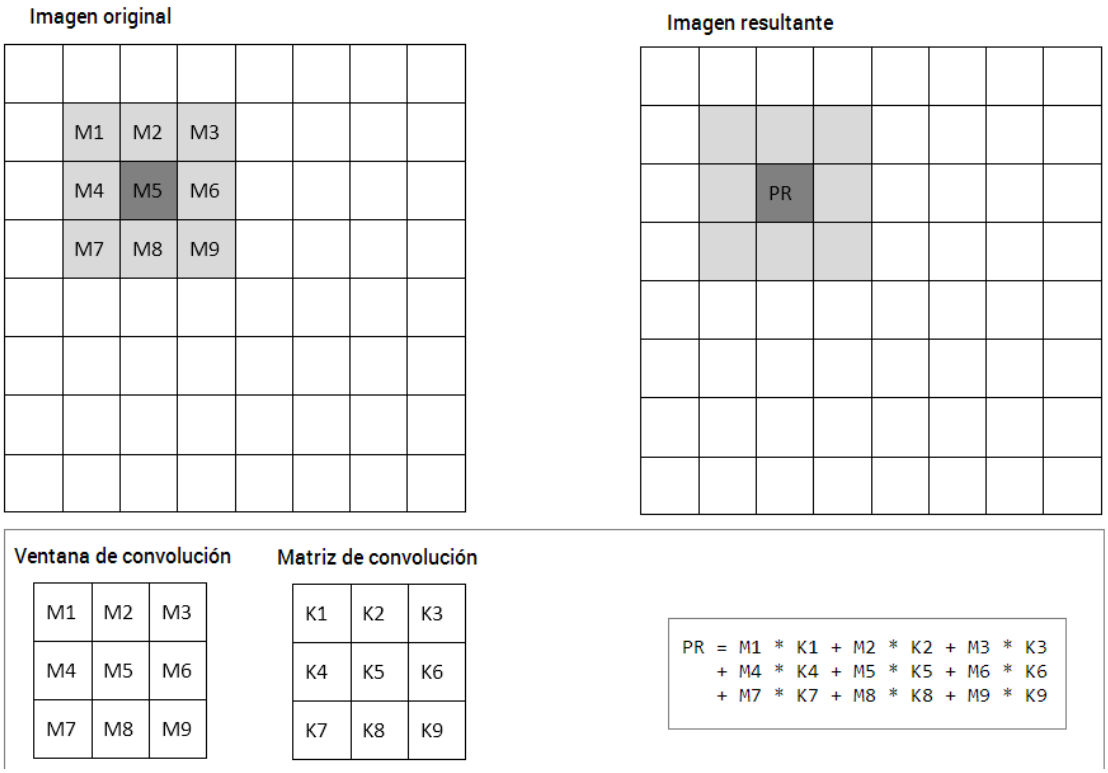


Figura 2.2: Filtro de correlación de 3×3.

imagen ya que el filtro se posiciona dentro de esta. Para resolver este posible inconveniente y controlar las dimensiones de la salida de las capas (*layers*) de convolución se usa *Zero-Padding*, que se describe en la sección 2.4. También se puede reducir la altura y anchura de la salida de la convolución con el uso de saltos (*strides*). En la convolución se asume que se extraen los parches posicionando el filtro en posiciones contiguas de la entrada pero la distancia entre dos extracciones de parches sucesivos es un parámetro llamado salto. Es posible hacer una convolución saltada (*strided convolution*) que es una convolución con un valor de salto mayor a 1 que resulta en un mapa de características reducido. Esta práctica es raramente utilizada ya que para para reducir los mapas de características se tiende a usar *Maxpooling* [13], que se describe en la sección 2.5.

2.4. *Zero-Padding*

Una característica de cualquier red neuronal convolucional es la colocación de manera implícita de *zero-padding* a la entrada de las capas convolucionales. Si no se implementa, el ancho y alto de la salida después de hacer la convolución queda reducido en cada capa convolucional y el tamaño de la imagen resultante en la salida podría no coincidir con el tamaño de la imagen de entrada [13]. Agregar *zero-padding* a la entrada nos permite controlar el ancho del *kernel* y el tamaño de la salida de la capa convolucional de forma independiente. En este caso la CNN puede contener tantas capas convolucionales como pueda soportar el *hardware* disponible. Cuando no se usa *zero-padding*, nos vemos obligados a elegir entre reducir el tamaño de la red neuronal convolucional y usar filtros pequeños, lo que limita de manera significativa el poder expresivo de la CNN [13]. A continuación se muestra un ejemplo de *zero-padding* usado en [6].

Considere un mapa de características de 5×5 (25 espacios en total). Solo hay 9 espacios alrededor de los cuales se puede centrar una ventana de 3×3 como se ve en la figura 2.3. Por lo tanto, el mapa de características de salida será de 3×3, se nota que el mapa de características

resultante es pequeño respecto a la entrada.

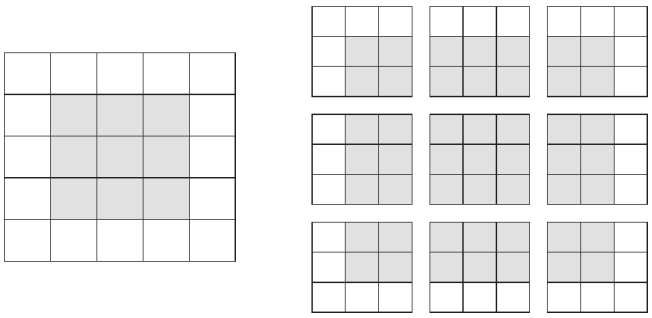


Figura 2.3: Posiciones válidas al pasar un filtro de 3×3 en un mapa de características de 5×5. Imagen de [6].

Si se desea un mapa de características de salida con las mismas dimensiones que la entrada se puede usar *zero-padding*, que consiste en agregar una cantidad adecuada de filas y columnas en cada lado del mapa de características de entrada para que las dimensiones del mapa de características de salida y de entrada tengan la misma dimensión. Para una ventana de 3×3 se agrega una columna a la derecha, izquierda, arriba y abajo, para una ventana de 5×5 se agregan dos filas en las mismas ubicaciones. La operación convolución de un *layer* junto con *zero-padding*, una función no lineal y *Maxpooling* son esenciales en CNNs.

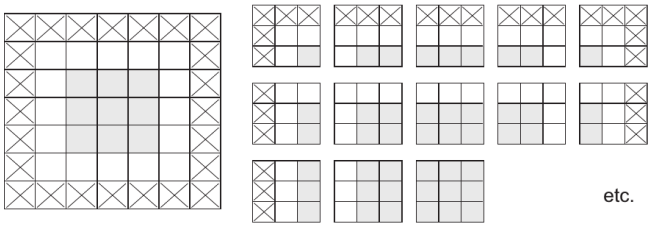


Figura 2.4: *Padding* de una entrada de 5×5 para poder extraer 25 parches de 3×3. Imagen obtenida de [6].

Lo mencionado anteriormente es importante en una CNN; también es importante contar con una función no lineal en cada capa convolucional ya que sin ella las capas convolucionales solo aprendería transformaciones lineales y el espacio de soluciones sería demasiado restringido. Además si son funciones lineales y tienen el mismo tamaño, serían redundantes porque una función lineal puede contener la composición de dos o más funciones lineales. Para acceder a un mejor espacio de soluciones se necesita una función no lineal o de activación. Actualmente la función no lineal más pópular es la unidad lineal rectificada (*rectified linear unit* en inlgés abreviado como ReLU) definida como $ReLU = max(z, 0)$ [12].

2.5. *Maxpooling*

La finalidad de usar *Maxpooling* en una red de convolución es filtrar activaciones ruidosas de una capa anterior mediante la extracción de activaciones en un campo receptivo con un solo valor representativo [10]. También sirve para agregar invarianza a la traslación y para lidiar con cambios de escala [13].

La arquitectura típica de una CNN consiste en una serie de niveles en la que los primeros niveles están compuestos por dos tipos de capas: de convolución y agrupamiento (*pooling*). El objetivo de los *layers* de convolucionales es detectar características del *layer* anterior. La finalidad de un *layer* que hace *pooling* es combinar semánticamente características similares en una sola [12], lo que reduce agresivamente los mapas de características. Hacer *pooling* permite que las representaciones varien poco cuando los elementos en la capa anterior varían en posición y apariencia [6].

El tipo más usual de *pooling* se llama *Maxpooling*, un *pooling layer* que calcula el máximo de una vecindad en un mapa de características (o muchos mapas de características). El ejemplo más común es crear una cuadrícula de 2×2 en cada mapa de características y seleccionar el valor máximo de activación en esa vecindad descartando el resto de valores y continuar recorriendo el mapa de características en *strides* (saltos de 2) [16].

Hay dos parámetros para cada capa de agrupación que son el tamaño del campo receptivo (a veces se le conoce como celda y en la librería Keras [7] se le llama *Pool size*) y el paso (*stride* en inglés). Una elección usual para estos parámetros es, como ya se ha mencionado antes, un tamaño del campo receptivo de 2 y un *stride* de 2, una consideración en el momento de elegir el tamaño de la celda es no escoger uno muy grande porque puede descartar mucha información al hacer la operación *pooling* y puede no ser útil.

Existe una fórmula para saber el resultado del *layer* de *pooling* [16]: con un tamaño de entrada I, tamaño del campo receptivo F, tamaño del salto S, se puede obtener en cada dimensión la salida O:

$$O_w = \frac{(I_w - F_w)}{S_w} + 1$$

$$O_h = \frac{(I_h - F_h)}{S_h} + 1$$

donde los subíndices w y h significan *width*, *height*: anchura, altura respectivamente.

Estas capas de *pooling* no cambian la profundidad del volumen de la entrada, manteniendo el mismo número de *rebanadas* (típicamente las rebanadas que resultan de una capa convolucional), ya que la operación de agrupamiento se realiza en cada rebanada de manera independiente.

De forma similar a las diferentes funciones de activación en las capas convolucionales, también se pueden usar diferentes operaciones de agrupamiento, obtener el valor máximo es lo más común (es decir, hacer *Maxpooling*), pero tomar el promedio de los valores en el campo receptivo (*Average pooling*) también es usado. En la práctica al usar *Maxpooling* se tiene un buen desempeño porque conserva las estructuras más relevantes en la imagen [6].

Si se quisiera usar una CNN para realizar una clasificación en imágenes se necesitan además de las capas de convolución y *pooling*, capas completamente conectadas y en caso de ser necesario, una capa de salida con activación *softmax*. En el caso de la segmentación que se realizó en este trabajo de tesis, se usa una capa que realiza la operación contraria al agrupamiento, generalmente se le conoce como *Unpooling* o *Upsampling* y se usa para hacer más grande el mapa de características con la finalidad de que después de una serie de *Unpooling*, el mapa de características tenga las mismas dimensiones que la imágen de entrada de la CNN [5].

2.6. *Upsampling*

Hacer *Maxpooling* en una CNN sirve para filtrar activaciones ruidosas en una capa anterior fusionando características semánticamente similares en una sola [12]. Aunque hacer esto ayuda a la clasificación al conservar solo activaciones robustas, la información espacial se pierde al hacer *Maxpooling*, lo que puede ser un problema para hacer una localización precisa, que se requiere para una segmentación mediante *deep learning* [10].

Para resolver este problema se usan capas de *upsampling* (también conocido como *Unpooling*) en la CNN que realizan la operación inversa a *Maxpooling* y reconstruyen el tamaño orginal de los mapas. Una manera de realizar esta operación es ir registrando las ubicaciones de las activaciones máximas seleccionadas por *Maxpooling* y después colocar cada activación de nuevo en su ubicación original. Esta estrategia es útil para reconstruir la estructura de los objetos.

En la figura 2.5 se observa la operación *Unpooling*, como solo se guarda la ubicación del valor máximo, el resto de ubicaciones se llenan con 0 [5].

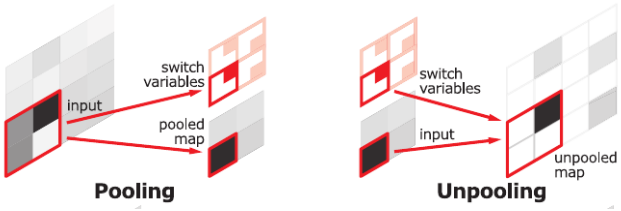


Figura 2.5: *Pooling* y *Unpooling*. En *switch variables* se guarda la ubicación de la activación máxima obtenida en *pooling*. Imagen de [10].

En Keras [7] la manera de hacer *Upsampling* es la siguiente: Repetir las filas y columnas de los datos en cantidad $size = (a, b)$ respectivamente.

Ejemplo: Se tiene el siguiente mapa de características para hacer *upsampling*

9	2
6	3

Después del $UpSampling$ con $size = (2, 2)$

9	9	2	2
9	9	2	2
6	6	3	3
6	6	3	3

2.7. Redes neuronales para la clasificación de imágenes

Existen cuatro ideas clave detrás de las CNNs que aprovechan las propiedades de las imágenes: interacciones dispersas, pesos compartidos, agrupamiento (*pooling*) y el uso de muchas capas [12].

En las redes neuronales convolucionales hay interacciones dispersas (también conocidas como conectividad dispersa o pesos dispersos). Esto se logra haciendo que el filtro convolucional (a veces se le conoce como *kernel*) sea más pequeño que la imagen de entrada. Por ejemplo, al procesar una imagen, esta puede tener miles o millones de pixeles pero, usando filtros que ocupan decenas de pixeles en la imagen, se pueden detectar características pequeñas y significativas como los bordes de los objetos en la imagen. Esto significa que se necesitan almacenar menos parámetros, reduciendo los requisitos de memoria del modelo de CNN [13].

Los pesos compartidos se refiere al uso del mismo valor de peso en varias partes en la entrada de la capa en un modelo de CNN. En una red neuronal tradicional con capas completamente conectadas, cada elemento de la matriz de pesos se multiplica por un elemento de la entrada y luego nunca se vuelve a usar ese peso. En una red neuronal convolucional, cada peso en el filtro se usa en cada posición de la entrada, el uso de pesos compartidos significa que en lugar de aprender un conjunto separado de parámetros para cada ubicación, solo se aprende un conjunto de filtros que contienen pesos [13]. Si una característica (o patrón) aparece en una parte de la imagen, podría aparecer en cualquier lugar, de ahí la idea de que los pesos en los filtros se usen en cada parte de la imagen [12]. Después de aprender a reconocer un patrón en una parte de la imagen, la CNN podrá reconocerlo en cualquier lugar, esto hace a las CNNs eficientes al procesar imágenes, ya que las características que aprenden durante el entrenamiento son invariantes a la traslación y el mundo visual es fundamentalmente invariante a la traslación [6].

Mientras que la función de los filtros de la capa convolucional es detectar los patrones (características) locales en la entrada, la función de la capa de *pooling* es fusionar las características semánticamente similares en una sola [12]. Debido a que las posiciones de las características importantes pueden variar un poco para su detección confiable, se puede realizar una agrupación (*pooling*) de la posición de cada característica para superar este problema. Un *pooling* típico obtiene el valor máximo en una pequeña zona de un mapa de características (*Maxpooling*), lo que reduce el tamaño del mapa de características y crea una invarianza a pequeños cambios y distorsiones [12]. Realizar *pooling* mejora la eficiencia computacional ya que al pasar a la siguiente capa, un mapa de características al que se le hizo *pooling* tiene una entrada más pequeña para procesar. También es eficiente cuando el número de pesos en la siguiente capa es una función del tamaño de la entrada, como en el caso de una capa completamente conectada. Reducir el tamaño mediante *pooling* resulta eficiente ya que requerirá menor memoria para almacenar los pesos de la capa completamente conectada [13].

El usar muchas capas en la CNN es para que tenga la propiedad de aprender jerarquías espaciales de patrones, porque el mundo visual es fundamentalmente jerárquico espacialmente [6]. De manera general se espera que la primera capa de convolución aprenda pequeños patrones locales como son los bordes de un objeto, la segunda capa aprenderá a encontrar patrones más elaborados a partir de los bordes, la siguiente capa encontrará partes de objetos, la siguiente objetos y así sucesivamente, lo que permite que las redes neuronales convolucionales aprendan conceptos visuales más complejos [12], [6].

2.8. *K-fold Cross-Validation*

Para evaluar una red neuronal mediante una validación cruzada (*cross-validation* también se le conoce como *k-fold cross-validation*), se puede dividir el *dataset* en un conjunto de entrenamiento y uno de validación. La mejor práctica es dividir los datos con los que se cuenta en K partes y luego dividir estos K conjuntos en los conjuntos de entrenamiento y validación. K suele ser 10 o 30 [2], pero como los conjuntos de datos frecuentemente no son los suficientemente grandes para hacer la división de ese tamaño, se pueden dividir en tamaños de 3, 4 o 5 [6]. Esta división se hace repetidamente pero de manera distinta. El *dataset* se divide en subconjuntos entrenando la red con una combinación diferente de estos subconjuntos y validando con los subconjuntos restantes, hasta validar con cada uno de los posibles subconjuntos. Un problema de esto es que los resultados obtenidos sean dependientes, ya que los conjuntos divididos comparten datos [2]. El valor de la validación cruzada para el modelo es el promedio de los K resultados obtenidos, esto es algo sencillo de hacer [6], en la figura 2.6 se muestra un instancia de una validación cruzada.

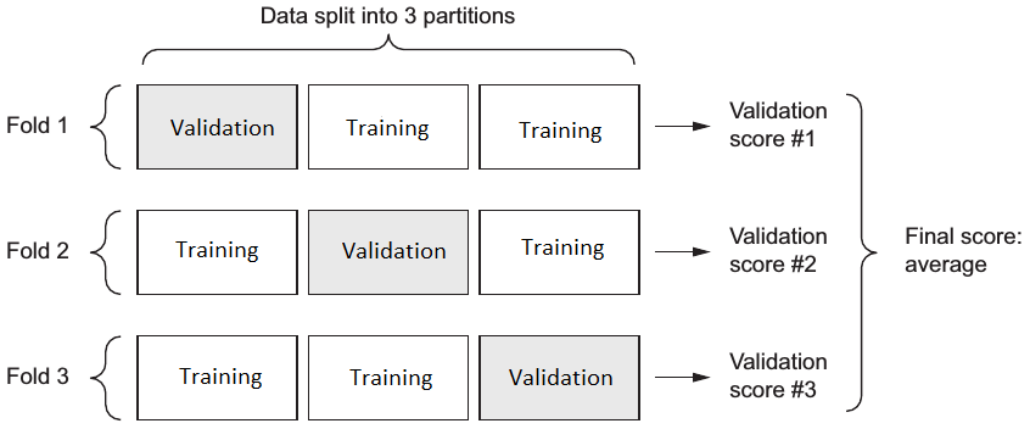


Figura 2.6: Una instancia de *K-Fold cross-validation* con $K = 3$.

Para cada conjunto de validación y entrenamiento es deseable que sea lo más grande y representativo posible, para que la estimación del error y otras métricas sea lo más robusta y al mismo tiempo mantener el traslape de los datos entre los diferentes conjuntos lo más pequeño posible. También se debe asegurar que las clases estén representadas en las proporciones correctas cuando sean creados los conjuntos de validación y entrenamiento para la validación cruzada, para no alterar los resultados que se obtengan, a esto se le llama estratificación. Por ejemplo, si una clase tiene 20 % de algún ejemplo en todo el *dataset*, entonces cada conjunto creado para el *cross-validation* debería tener aproximadamente 20 % de esos ejemplos [2].

Un caso de *K-fold cross-validation* es dejar-uno-afuera (*leave-one-out*), dado un *dataset* de N instancias, solo se deja una instancia como el conjunto de datos de validación y en el entrenamiento se usan las $N - 1$ instancias restantes, dejando una instancia distinta fuera en cada iteración. Dejar-uno-fuera no permite la estratificación [2] [6].

Capítulo 3

Antecedentes

Segmentar imágenes mediante aprendizaje profundo ha sido un problema de gran interés y bastante estudiado. Una de las propuestas de solución más conocidas se llama *SegNet* [8], una red neuronal convolucional inspirada en *VGG 16 Net* [22], que es una red neuronal diseñada para clasificar. SegNet se diseñó para segmentar sobre imágenes de interiores y en caminos de carretera. Esto último es muy útil para conducción autonóma de vehículos. Las modificaciones hechas sobre *VGG 16 Net* para crear *SegNet* son: eliminar las capas completamente conectadas para poner en su lugar un *decodificador* compuesto de capas convolucionales y de *Upsampling*. Al final los mapas de características de la salida del decodificador se envían a una capa con activación *softmax* donde a cada pixel de manera independiente se le asigna la probabilidad de pertenecer a una clase. SegNet fue programada usando la librería llamada *Caffe*. En la figura 3.1 se aprecia una ilustración general de SegNet.

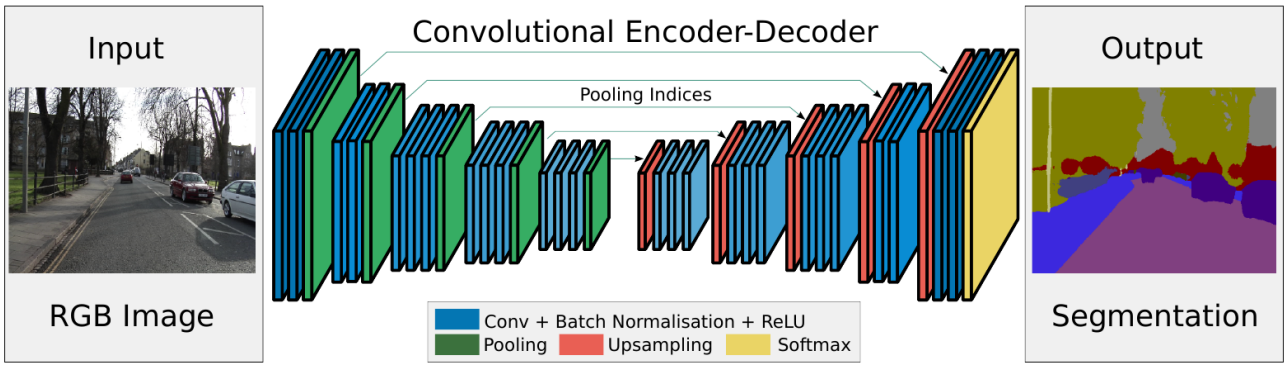


Figura 3.1: Ilustración de SegNet, contiene capas convolucionales, *pooling* y *upsampling*. Los mapas de características de la salida del decodificador se envían a una capa con activación *softmax* para la clasificación de pixeles. Imagen de [8].

El *codificador* de SegNet, que es la primera mitad de la red neuronal, consta de 13 capas convolucionales que corresponden a las primeras 13 capas convolucionales de VGG16Net con los pesos pre-entrenados, el número de parámetros es de 14.7M, mientras que el *decodificador* tiene una capa convolucional y de *upsampling* correspondiente por cada capa convolucional y de *Maxpooling* en el codificador y por lo tanto, la red de decodificación tiene 13 capas convolucionales. Cada *layer* convolucional realiza una convolución con filtros para producir un conjunto de mapas de características. Después se les aplica un *batch normalization* seguido de una función no lineal llamada $ReLU$ definida como $ReLU = max(0, x)$.

En el encoder se aplica *Maxpooling* con una ventana de 2×2 con saltos de 2.

En el decoder se aplica *Upsampling* donde se usan los índices de 2 bits de *Maxpooling* que se almacenan. En la figura 3.3 se aprecia la manera de hacer esta operación.

La salida del decodificador es la entrada a un clasificador *softmax* de múltiples clases para

producir probabilidades de clase para cada pixel de forma independiente, al final la segmentación predicha corresponde a la máxima probabilidad de clase en cada pixel.

Para entrenar SegNet usaron el conjunto de datos llamado *CamVid* que consiste en 367 imágenes de entrenamiento y 233 imágenes de prueba de escenarios de carreteras de ciudades con una resolución de 360×480. En la figura 3.2 se muestran imágenes de ejemplo de este conjunto de datos. Para otro experimento usaron 3433 imágenes para entrenar SegNet combinando los conjuntos de datos: *ICCV09DATA*, *Cambridge-driving Labeled Video Database (CamVid)*, *KITTI semantic segmentation benchmark* y *LabelMe*.

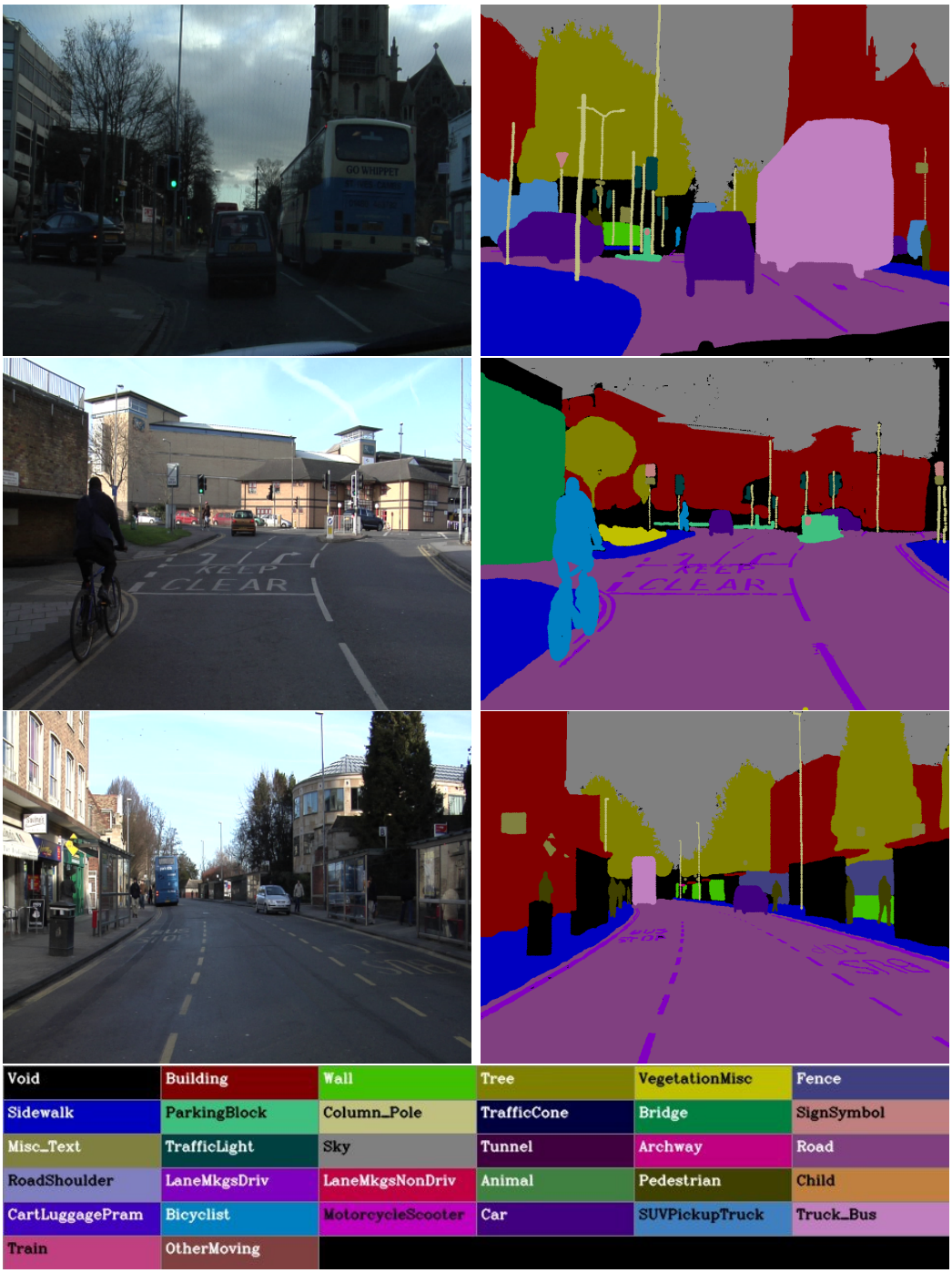


Figura 3.2: Imágenes de *CamVID dataset*. Columna izquierda, imágenes RGB. Columna derecha, *Ground Truth*. Fila de abajo, lista de colores de las clases asignadas.

Para comprobar se usó también el *dataset Sun RGB-D* que consiste en 5285 imágenes de entrenamiento y 5050 imágenes de prueba. Todas las imágenes de este conjunto de datos son

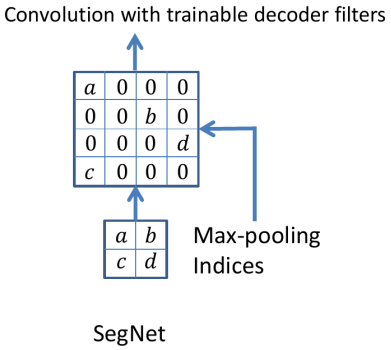


Figura 3.3: Ejemplo del *decoder* de SegNet. a, b, c, d corresponden a valores en el mapa de características, SegNet usa los índices de *Maxpooling* para hacer *Upsampling* en los mapas de características, después convoluciona con varios filtros entrenables. Imagen de [8].

de escenarios interiores, en la figura 3.4 se encuentra una imagen de muestra de este conjunto de datos.

Para comparar cuantitativamente el desempeño de SegNet, usaron las métricas: exactitud global (*global accuracy* G), que mide el porcentaje de pixeles correctamente clasificados en el conjunto de datos, exactitud promedio de clase (*class average accuracy* C), que es la media de la exactitud predictiva sobre todas las clases, la intersección de la media sobre la unión de todas las clases (mIoU), a esta útima métrica se le conoce también como índice de Jacard, y *boundary F1 measure* (BF), que consiste en calcular la medida-f para cada clase en una imagen de prueba y promediarla, luego se promedia sobre todas las imágenes en el *test set*.

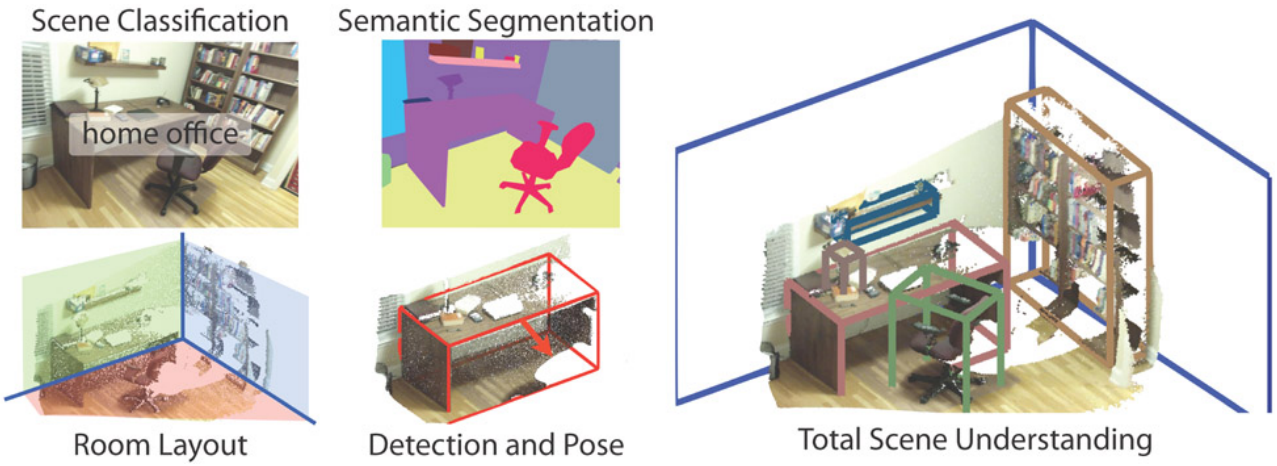


Figura 3.4: Imagen de muestra de *Sun RGB-D datset*.

Otros hiperparámetros que reportan los autores son:

- Como función de optimización usa *stochastic gradient descent* con un *learning rate* de 0.1 y *momentum* de 0.9,
- La función de pérdida usada es *cross-entropy*.
- Se entrenó por 50, 100 y mayor a 100 épocas.

Donde un hiperparámetro es un parámetro de un algoritmo de aprendizaje y no del modelo, como tal no se ve afectado por el algoritmo de aprendizaje en sí, debe establecerse antes de un entrenamiento del modelo y permanecer constante. Optimizar hiperparámetros es importante en la construcción de un sistema de aprendizaje profundo [3]. Los resultados publicados se muestran en la tabla 3.1, se compara SegNet contra otros métodos tradicionales usados para

segmentar. Algunos métodos usan CRF, que significa *conditional random fields*, es un método clásico de *machine learning* que puede ser usado para segmentar y con el que se pueden obtener buenas segmentaciones.

Method	Building	Tree	Sky	Car	Sign-Symbol	Road	Pedestrian	Fence	Column-Pole	Side-walk	Bicyclist	Class avg.	Global avg.	mIoU	BF
SfM+Appearance [28]	46.2	61.9	89.7	68.6	42.9	89.5	53.6	46.6	0.7	60.5	22.5	53.0	69.1	n/a*	
Boosting [29]	61.9	67.3	91.1	71.1	58.5	92.9	49.5	37.6	25.8	77.8	24.7	59.8	76.4	n/a*	
Dense Depth Maps [32]	85.3	57.3	95.4	69.2	46.5	**98.5**	23.8	44.3	22.0	38.1	28.7	55.4	82.1	n/a*	
Structured Random Forests [31]	n/a											51.4	72.5	n/a*	
Neural Decision Forests [64]	n/a											56.1	82.1	n/a*	
Local Label Descriptors [65]	80.7	61.5	88.8	16.4	n/a	98.0	1.09	0.05	4.13	12.4	0.07	36.3	73.6	n/a*	
Super Parsing [33]	87.0	67.1	96.9	62.7	30.1	95.9	14.7	17.9	1.7	70.0	19.4	51.2	83.3	n/a*	
SegNet (3.5K dataset training - 140K)	**89.6**	**83.4**	96.1	**87.7**	52.7	96.4	**62.2**	**53.45**	**32.1**	**93.3**	**36.5**	**71.20**	**90.40**	60.10	46.84
CRF based approaches															
Boosting + pairwise CRF [29]	70.7	70.8	94.7	74.4	55.9	94.1	45.7	37.2	13.0	79.3	23.1	59.9	79.8	n/a*	
Boosting+Higher order [29]	84.5	72.6	**97.5**	72.7	34.1	95.3	34.2	45.7	8.1	77.6	28.5	59.2	83.8	n/a*	
Boosting+Detectors+CRF [30]	81.5	76.6	96.2	78.7	40.2	93.9	43.0	47.6	14.3	81.5	33.9	62.5	83.8	n/a*	

Tabla 3.1: Comparaciones cuantitativas de SegNet con métodos tradicionales sobre CamVid.

En la tabla 3.2 se compara SegNet contra otras CNNs que realizan segmentación, por ejemplo DeconvNet. Los resultados se obtienen sobre el conjunto de prueba CamVid y se entrenó con las 3433 imágenes descritas anteriormente. Las iteraciones mostradas se pueden interpretar como épocas mediante la siguiente relación proporcionada por los autores: 40K, 80K y >80K iteraciones corresponden a 50, 100 y >100 épocas aproximadamente.

Network/Iterations	40K				80K				>80K				Max iter
	G	C	mIoU	BF	G	C	mIoU	BF	G	C	mIoU	BF	
SegNet	88.81	59.93	50.02	35.78	89.68	69.82	57.18	42.08	90.40	71.20	60.10	46.84	140K
DeepLab-LargeFOV [3]	85.95	60.41	50.18	26.25	87.76	62.57	53.34	32.04	88.20	62.53	53.88	32.77	140K
DeepLab-LargeFOV-denseCRF [3]	not computed								89.71	60.67	54.74	40.79	140K
FCN	81.97	54.38	46.59	22.86	82.71	56.22	47.95	24.76	83.27	59.56	49.83	27.99	200K
FCN (learnt deconv) [2]	83.21	56.05	48.68	27.40	83.71	59.64	50.80	31.01	83.14	64.21	51.96	33.18	160K
DeconvNet [4]	85.26	46.40	39.69	27.36	85.19	54.08	43.74	29.33	89.58	70.24	59.77	52.23	260K

Tabla 3.2: Resultado cuantitativo de CNNs sobre el conjunto de prueba de CamVid cuando se entrena sobre 3433 imágenes de carretera. Se observa que DeconvNet se acerca a los valores reportados por SegNet pero a un alto costo computacional.

En la tabla 3.3 se comprara SegNet con otras redes neuronales convolucionales profundas sobre *SUNRGB-D dataset* cuando son entrenadas usando 5250 imágenes de escenarios interiores. Concluyendo que aunque otros modelos de aprendizaje profundo reportan un buen desempeño, SegNet es una red más pequeña y más rápida de entrenar que otros modelos, lo que es muy importante cuando se tiene poco poder de cómputo. Otra característica de SegNet que la hace rápida de entrenar es que guarda los índices de *maxpooling* en 2 bits ocupando poco recursos. SegNet segmenta principalmente en escenarios de exteriores y es muy útil para tareas de robótica y conducción autónoma, los autores también hacen mención que las imágenes de escenas del exterior capturadas desde un automóvil en movimiento son más fáciles de segmentar, caso contrario a las imágenes de escenas de interiores.

Una obervación sobre este trabajo es que los autores mencionan que una razón por la que en general el desempeño de SegNet es bajo, es el alto número de clases en la tarea de segmentación en interiores, muchas de las clases ocupan una pequeña parte de la imagen y aparecen con poca frecuencia. Nuestro *dataset* cumple con esta descripción: en nuestras imágenes aparecen objetos que ocupan una pequeña parte de la imagen, por ejemplo, maleza, rocas, objetos usados por los agricultores, que aparecen con poca frecuencia. Tener en cuenta este detalle es importante

para realizar una buena segmentación.

Otra red neuronal para segmentar es propuesta por Hyeonwoo Noh, Seunghoon Hong y Bohyung Han y se conoce como *red de deconvolución* o *DeconvNet* [10] cuya arquitectura también está inspirada en *VGG 16 Net* [22], con la diferencia de que no tiene una capa *softmax*, solo *layers* de convolución *Maxpooling*, Deconvolución (*Deconvolution* en inglés) y *Unpooling*. En la figura 3.5 se muestra DeconvNet, que de manera general se compone de dos partes: red de convolución (*convolution network* en inglés) y red de deconvolución (*deconvolution network* en inglés).

Entre sus principales contribuciones, los autores mencionan que la red de deconvolución para hacer segmentación es muy útil y ellos son los primeros en realizar segmentación con este tipo de red.

La red de convolución es un extractor de características mientras que la red de deconvolución produce la segmentación de objetos a partir de las características extraídas de la red de convolución. La salida de la red es un mapa de probabilidades que indica la probabilidad de pertenencia de cada pixel a una clase, este mapa de probabilidades es del mismo tamaño que la imagen de entrada. La DeconvNet usa la red neuronal VGG16Net [22] para la red convolucional eliminando la parte de la red que clasifica, por lo tanto esta parte tiene 13 capas y la parte de deconvolución es una versión duplicada de la red de convolución.

La DeconvNet para realizar *maxpooling* recorre el mapa de características con una ventana de 2×2 en la que se queda con el valor máximo, esto se hace para filtrar activaciones ruidosas quedándose con un solo valor representativo. Una característica de *maxpooling* es que reduce el tamaño de los mapas de características perdiendo información espacial. Perder información espacial puede ser un problema para realizar la segmentación al no poder clasificar con precisión de pixeles. Para resolver este problema los autores usan *unpooling* en la parte de deconvolución para reconstruir el tamaño original de los mapas de activación. Para hacer esta operación primero se guardan las ubicaciones de las activaciones máximas de *maxpooling* en unas variables llamadas *switch variables*, después se usan estas variables para colocar cada activación en la posición registrada de su *maxpooling* correspondiente, típicamente en las posiciones restantes se coloca el valor 0, de esta manera se intenta conservar información espacial. En la figura 3.6 se muestra cómo se hacen estas operaciones.

La salida de *unpooling* es un mapa de características más grande pero disperso, provocado por los valores 0 que se agregan al realizar esta operación, los autores usan capas de deconvolución para eliminar esta dispersión. Esta operación se realiza mediante una convolución con varios filtros pero a diferencia de una capa convolucional en la que al pasar un filtro se va obteniendo una salida, las capas deconvolucionales tienen varias salidas, tantas como sea el tamaño del filtro. Como se puede ver en la figura 3.6 la deconvolución incrementa el tamaño de lo que se dé como entrada. Para resolver este problema los autores recortan los bordes de la salida.

Esta red se entrenó con el *Pascal VOC dataset* [15] con 12031 imágenes de entrenamiento y

Network/Iterations	80K				140K				>140K				Max iter
	G	C	mIoU	BF	G	C	mIoU	BF	G	C	mIoU	BF	
SegNet	70.73	30.82	22.52	9.16	71.66	37.60	27.46	11.33	72.63	44.76	31.84	12.66	240K
DeepLab-LargeFOV [3]	70.70	41.75	30.67	7.28	71.16	42.71	31.29	7.57	71.90	42.21	32.08	8.26	240K
DeepLab-LargeFOV-denseCRF [3]	not computed								66.96	33.06	24.13	9.41	240K
FCN (learnt deconv) [2]	67.31	34.32	24.05	7.88	68.04	37.2	26.33	9.0	68.18	38.41	27.39	9.68	200K
DeconvNet [4]	59.62	12.93	8.35	6.50	63.28	22.53	15.14	7.86	66.13	32.28	22.57	10.47	380K

Tabla 3.3: Resultado cuantitativo de comprar SegNet con otras CNNs sobre *Sun RGB-D datset* cuando se entrena sobre 5250 imágenes de escenarios interiores. Una razón por la que en general el desempeño de SegNet es bajo es el alto número de clases en la tarea de segmentación en interiores, muchas de las clases ocupan una pequeña parte de la imagen y aparecen con poca frecuencia.

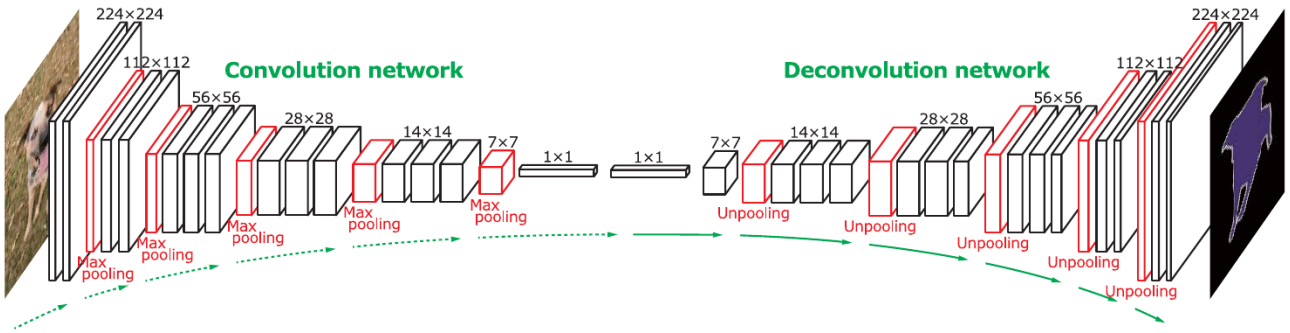


Figura 3.5: Ilustración de DeconvNet. La parte de *convolution network* está basada en VGG 16 net [22] y es seguida de una *deconvolution network*. Con una representación de características obtenidas en *convolution network* se construye un mapa de predicción de clases con precisión de pixel en *deconvolution network*. Imagen de [10].

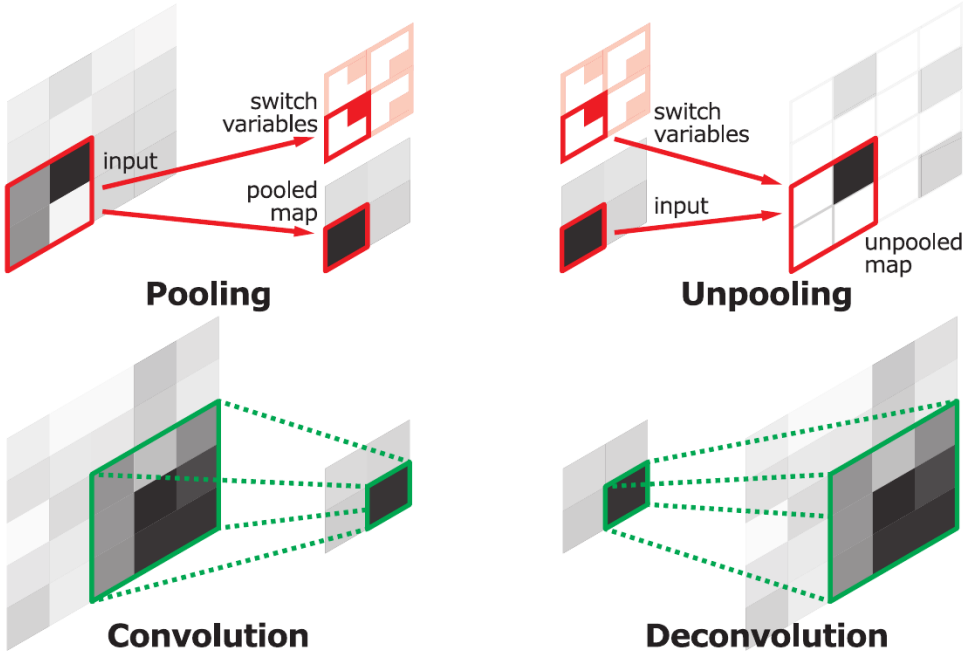


Figura 3.6: Ejemplo de las operaciones *Pooling* (superior izquierda), *Unpooling* (superior derecha), *Convolution* (inferior izquierda) y *Deconvolution* (inferior derecha). Imagen de [10].

validación. En la figura 3.7 se muestran imágenes pertenecientes a este *dataset*. Entrenar una red neuronal profunda que realice una muy buena segmentación con este número de ejemplos es muy difícil, pero los autores usan los siguientes métodos para lograr un buen desempeño: Método uno: *Batch normalization*, que típicamente normaliza la entrada de las capas de forma adaptativa. Mantiene la media cerca de 0 y la desviación estándar cerca de 1. Realizar esta normalización ayuda a la propagación del gradiente. *Batch normalization* se agrega después de una convolución, esto ayuda a evitar óptimos locales. Método dos: entrenar en dos etapas para que la red se entrene para segmentar imágenes complicadas, en la primera etapa entrenan la red con ejemplos sencillos recortando las imágenes originales para que el objeto de interés en el GT quede ubicado en el centro, es más fácil porque se reduce el espacio de búsqueda del objeto de interés y en la segunda etapa entrenan con ejemplos complicados seleccionando ejemplos donde los objetos de interés estén traslapados para enseñarle a la red a desagrupar los objetos de interés.

La red de deconvolución fue implementada usando el *framework* para aprendizaje profundo conocido como Caffe. Como función de optimización se usó *stochastic gradient descent with momentum* con los siguientes parametros: *learning rate*: 0.01, *momentum*: 0.9 y *weight decay*:

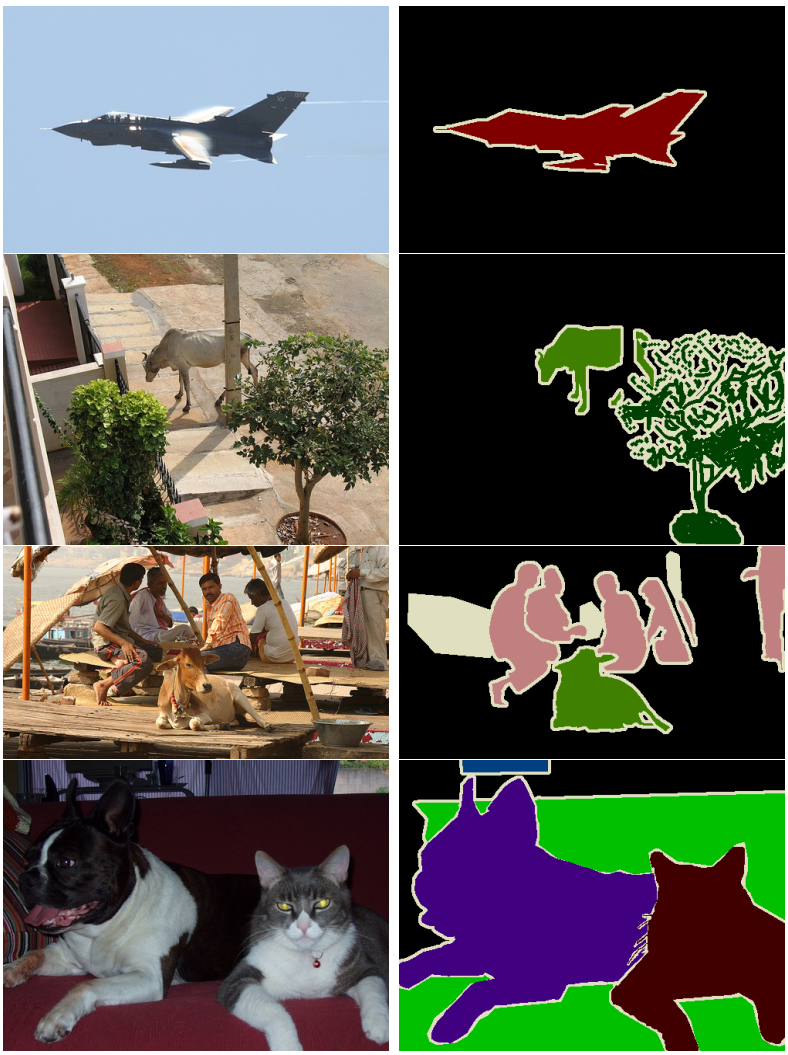


Figura 3.7: Algunas imágenes de *Pascal VOC dataset*. Columna izquierda, imágenes RGB. Columna derecha, *Ground Truth*.

0.0005. Se inicializaron los pesos de la red de convolución con la red VGG16Net entrenada sobre el conjunto de datos ImageNet mientras que los pesos en la parte de deconvolución se inicializaron usando números con distribución Gaussiana con media cero.

Para medir el desempeño de la red miden la intersección sobre la unión entre el *ground truth* y las segmentaciones predichas. Los resultados se aprecian en la tabla 3.4, la red de deconvolución se muestra con el nombre DeconvNet, CRF significa que a la salida de la red se le aplica un postprocesamiento usando campos aleatorios condicionales (*conditional random fields* en inglés) y la letra E al principio de DeconvNet significa que la segmentación se produjo calculando la media de los valores de salida de DeconvNet y una red llamada *Fully convolutional network for semantic segmentation* (FCN por sus siglas en inglés).

Concluyen los autores que la red de deconvolución que proponen tiene un rendimiento tan bueno como los métodos del estado del arte, por ejemplo, la red FCN, sobre el conjunto de datos Pascal Voc dataset 2012, sin usar otro conjunto de datos para entrenar la red de deconvolución. Además es posible combinar su red de deconvolución con la red FCN y con el método llamado *conditional random fields* para obtener mejores resultados.

La diferencia entre SegNet y DeconvNet son: DeconvNet usa una operación llamada deconvolución y SegNet no, SegNet se entrena principalmente sobre imagenes de carretera y DeconvNet sobre *Pascal Voc dataset*, DeconvNet tiene mas parámetros. La operación *Unpooling* entre ambas CNNs se realiza de la misma manera.

Method	bkg	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbk	person	plant	sheep	sofa	train	tv	mean
Hypercolumn [11]	88.9	68.4	27.2	68.2	47.6	61.7	76.9	72.1	71.1	24.3	59.3	44.8	62.7	59.4	73.5	70.6	52.0	63.0	38.1	60.0	54.1	59.2
MSRA-CFM [3]	87.7	75.7	26.7	69.5	48.8	65.6	81.0	69.2	73.3	30.0	68.7	51.5	69.1	68.1	71.7	67.5	50.4	66.5	44.4	58.9	53.5	61.8
FCN8s [19]	91.2	76.8	34.2	68.9	49.4	60.3	75.3	74.7	77.6	21.4	62.5	46.8	71.8	63.9	76.5	73.9	45.2	72.4	37.4	70.9	55.1	62.2
TTI-Zoomout-16 [20]	89.8	81.9	35.1	78.2	57.4	56.5	80.5	74.0	79.8	22.4	69.6	53.7	74.0	76.0	76.6	68.8	44.3	70.2	40.2	68.9	55.3	64.4
DeepLab-CRF [1]	93.1	84.4	**54.5**	81.5	63.6	65.9	85.1	79.1	83.4	30.7	74.1	59.8	79.0	76.1	83.2	80.8	**59.7**	82.2	50.4	73.1	63.7	71.6
DeconvNet	92.7	85.9	42.6	78.9	62.5	66.6	87.4	77.8	79.5	26.3	73.4	60.2	70.8	76.5	79.6	77.7	58.2	77.4	52.9	75.2	59.8	69.6
DeconvNet+CRF	92.9	87.8	41.9	80.6	63.9	67.3	**88.1**	78.4	81.3	25.9	73.7	61.2	72.0	77.0	79.9	78.7	59.5	78.3	**55.0**	75.2	61.5	70.5
EDeconvNet	92.9	88.4	39.7	79.0	63.0	67.7	87.1	**81.5**	84.4	27.8	76.1	61.2	78.0	79.3	83.1	79.3	58.0	82.5	52.3	80.1	64.0	71.7
EDeconvNet+CRF	93.1	**89.9**	39.3	79.7	63.9	**68.2**	87.4	81.2	**86.1**	28.5	**77.0**	62.0	79.0	**80.3**	**83.6**	80.2	58.8	**83.4**	54.3	**80.7**	65.0	**72.5**
* WSSL [21]	93.2	85.3	36.2	**84.8**	61.2	67.5	84.7	81.4	81.0	**30.8**	73.8	53.8	77.5	76.5	82.3	**81.6**	56.3	78.9	52.3	76.6	63.3	70.4
* BoxSup [2]	**93.6**	86.4	35.5	79.7	**65.2**	65.2	84.3	78.5	83.7	30.5	76.2	**62.6**	**79.3**	76.1	82.1	81.3	57.0	78.2	**55.0**	72.5	**68.1**	71.0

Tabla 3.4: Resultados cuantitativos de DeconvNet comparado con otros métodos del estado del arte.

Una observación es que hay ocasiones en que se obtienen imágenes con mucho ruido en su segmentación, por ejemplo hay pixeles segmentados que pertenecen a clases incorrectas, también hay ocasiones donde el fondo aparece segmentado. Esto presenta una desventaja para nuestro trabajo de segmentar un cultivo de higo, ya que puede pasar que se segmente mal marcando como cultivo la maleza o el suelo que al tener varios tonos pueden aparecer segmentados como cultivo, lo que provocaría un mal rendimiento en nuestra tarea.

Una propuesta de segmentar imágenes de cultivos mediante aprendizaje profundo (usando una CNN) fue hecha por Philipp Lottes, Andres Milioto y Cyrill Stachniss [18]. El cultivo sobre el que se trabajó es remolacha azucarera (*sugar beet* en inglés) y lo que se segmentó fueron plantas de remolacha, maleza y suelo obteniendo tres objetos de interés. La segmentación se realiza con base únicamente en datos obtenibles de imágenes RGB. Estos datos obtenibles son índices de vegetación ExR, ExG, CIVE y NDI, el espacio de color HSV, derivadas de Sobel, el Laplaciano y los contornos mediante el detector de Canny. Las derivadas y los contornos se obtienen a partir de ExG. De esta manera constituyen una entrada de 14 canales. Además abordan el problema de segmentar cultivos a aproximadamente la velocidad de cuadros por segundo de la cámara.

Su CNN (ver la figura 3.8) es capaz de realizar la segmentación en tiempo real, implementa y evalúa su CNN en un robot agrícola real. Los resultados que proporciona sugieren que su sistema de segmentación generaliza bien y es adecuado para operaciones en tiempo real sobre los campos.

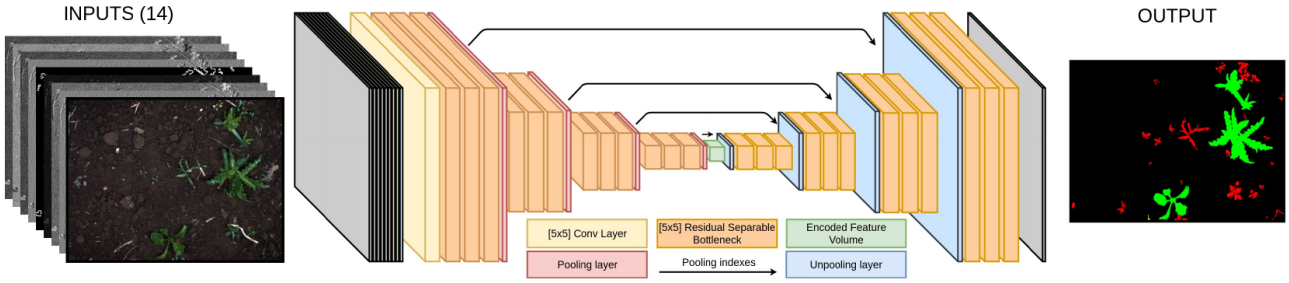


Figura 3.8: Ilustración del modelo *encoder-decoder* para la segmentación de cultivos usado por Philipp Lottes. Imagen de [18].

La contribución principal de su trabajo es un nuevo enfoque para la clasificación de malezas de cultivos utilizando datos provenientes de imágenes RGB. La clasificación depende de usar redes neuronales convolucionales (CNN). El objetivo es proporcionar a la CNN conocimientos de antecedentes relevantes para la tarea de segmentar a nivel de pixel con la finalidad de acelerar el entrenamiento y generalizar mejor a nuevos campos de cultivo de remolacha.

Los autores tienen tres objetivos que son: 1) Realizar con precisión de pixel la segmentación de cultivos, maleza y suelo tratando adecuadamente objetos superpuestos y a cultivos en varias

etapas de crecimiento. 2) Crear una CNN que actúe como un extractor de características robusto que generaliza bien las condiciones de iluminación, condiciones climáticas y de suelo. 3) Su CNN requerirá poca información para adaptarse a un nuevo entorno. La segmentación de cultivos trabaja en tiempo real en una GPU.

Su metodología está separada en dos partes, primero se calculan diferentes índices de vegetación y representaciones alternativas que son usadas comunmente en clasificación de plantas y ayudan a la CNN ya que cuentan con un conjunto de datos pequeño que no es muy común al aplicar *deep learning*. Como segundo paso emplean una CNN que los autores diseñaron para hecer segmentación.

Para calcular los 4 índices de vegetación que son Exceso de verde (EXG), Exceso de rojo (EXR), Índice de color de extracción de vegetación (CIVE) y Normalizado de índice de diferencias (NDI) se usan las siguientes fómulas:

$$I_{EXG} = 2I_G - I_R - I_B$$

$$I_{EXR} = 1.4I_R - I_G$$

$$I_{CIVE} = 0.881I_G - 0.441I_R - 0.385I_B - 18.78745$$

$$I_{NDI} = \frac{I_G - I_R}{I_G + I_R}$$

Estos índices comparten la propiedad de ser menos sensibles a los cambios de iluminación. Junto con ellos se usa el espacio de color HSV y las derivadas de Sobel, Laplaciano y el detector de Canny. Todas estas representaciones están *pegadas* a los tres canales de imagen de entrada RGB y constituyen una entrada de 14 canales que se da como entrada a la CNN, como se puede ver en la tabla 3.5. En la imagen 3.9 se observa la imagen de algunas de estas representaciones, para lograr este primer paso los autores usaron la librería llamada OpenCV [4] muy usada en procesamiento de imágenes.

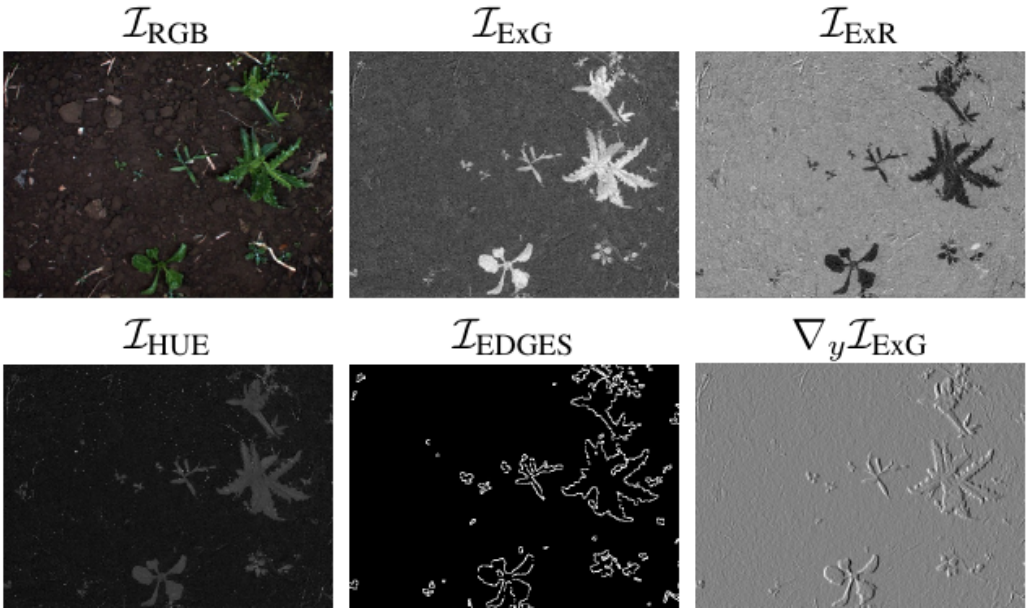


Figura 3.9: Ilustración de algunas de las representaciones alternativas usadas.

Número de canal	Contenido del canal
I_1	I_R
I_2	I_G
I_3	I_B
I_4	I_{EXG}
I_5	I_{EXR}
I_6	I_{CIVE}
I_7	I_{NDI}
I_8	I_{HUE} del espacio de color HSV
I_9	I_{SAT} del espacio de color HSV
I_{10}	I_{VAL} del espacio de color HSV
I_{11}	$\nabla_x I_{EXG}$ Sobel en la dirección x de I_{EXG}
I_{12}	$\nabla_y I_{EXG}$ Sobel en la dirección y de I_{EXG}
I_{13}	$\nabla^2_{I_{EXG}}$ Laplaciano en I_{EXG}
I_{14}	I_{EDGES} Detector de bordes de Canny en I_{EXG}

Tabla 3.5: Índices y representaciones usadas como entrada en la CNN.

Para el segundo paso, que es diseñar la CNN, los autores usaron Tensorflow [1]. Para implementar la CNN usaron las capas: convolucional, cuello de bottella de residuos separables (*residual separable bottleneck* en inglés), pooling, unpooling y capa de salida. Los detalles se describen a continuación:

- **Imagen de entrada**: Se redimensiona la imagen de entrada a 512×384.
- **Capa convolucional**: Contiene 16 filtros de 5×5 con activación ReLU con Zero-Padding.
- **Cuello de botella de residuos separables**: Esta capa es para lograr un procesamiento mas rápido, se basa en usar operaciones convolucionales separadas. La conexión residual ayuda con el problema de degradación, que es un aumento en el error durante el entrenamiento en CNNs muy profundas. En la figura 3.10 se observa como construyen esta capa.
- **Pooling**: Tamaño de ventana de 2×2 con un salto de dos, obteniendo el valor máximo cuando se desliza la ventana por el mapa de características.
- **Unpooling**: Comparte los índices usados en su operación pooling simétrica, con la intención de compartir las posiciones de máxima activación y obtener información espacial en la parte del encoder.
- **Salida**: Posee una activación *softmax* para obtener la probabilidad de que el pixel pertenezca al suelo, maleza o cultivo.

Más información que ofrecen los autores sobre la construcción de su CNN son:

- **Función de optimización**: Adam.
- **Tamaño del *batch***: 15.
- **Función de pérdida**: Cross-entropy.
- **Número de épocas de entrenamiento**: 200.

El *dataset* para entrenar la CNN se le conoce como el conjunto de datos de Bonn, estas imágenes fueron capturadas en Bonn Alemania y una parte de este *dataset* está disponible al público. En la figura 3.11 se muestra una imagen de este *dataset* con su segmentación. Este conjunto de datos consta de 10036 imágenes, fue tomado en el transcurso de un mes y cuenta con imágenes de cultivos de remolacha en varias etapas de crecimiento. Estas imágenes se separaron en 70 % para entrenar, 15 % para validar y 15 % para probar.

Para entrenar usan un *aumento de datos* que consiste en deformar un poco las imágenes de entrada realizando estiramiento, rotaciones y otras modificaciones de manera aleatoria con la finalidad de incrementar el número de ejemplos que se le dan a la CNN, esta práctica es muy común cuando se quiere aplicar *deep learning* con un conjunto de datos pequeño.

Para probar su modelo se usó además del 15 % de Bonn, 2577 imágenes de un campo en Zurich Suiza y 2584 imágenes tomadas en Stuttgart Alemania, estos tres conjuntos cuentan con su respectiva imagen capturada en el infrarrojo cercano (NIR por sus siglas en inglés). La manera en la que se muestra el rendimeinto de su CNN es entrenando tres CNNs: una solo con imágenes RGB, otra con RGB + NIR y la última con RGB + representaciones extra.

Como resultados presentan la precisión de la segmentación, el resultado de adaptar la CNN a nuevos cultivos y resultados de la segmentación en tiempo real.

Los resultados de la precisión de la segmentación se muestran en la tabla 3.6. Para 15 % de Bonn con los conjuntos Stuttgart y Zurich al 100 %, la precisión y el recuerdo son muy altos en la mayoría de los casos pero es muy seguido por la red que usa RGB + NIR, además en los conjuntos Stuttgart y Zurich la precisión y el recuerdo son bastante bajos, por ejemplo la precisión al segmentar las imágenes de Zurich es del 23.05 %, en el recuerdo (*recall*) del conjunto Stuttgart para maleza es del 46.9 %, en general las tres CNNs son muy buenas para clasificar el suelo siendo todos los resultados de alrededor del 99 %.

Para saber cómo se adapta su CNN a nuevos cultivos mostrando que su CNN es un buen extractor de características, se realizó otro experimento en el cual las imágenes de los cultivos Stuttgart y Zurich se separaron a la mitad, se extrajeron de una mitad conjuntos de 10, 20, 50 y 100 imágenes y solo se entrena el último *layer* que fue entrenado en Bonn separando estos pequeños conjuntos en 80 % y 20 % y entrenando hasta que el error en el *validation set* empiece a aumentar, con la finalidad de tener este proceso automatizado. En la tabla 3.7 se muestran los resultados de precisión y recuerdo al detectar los objetos de interés. Para el conjunto Zurich, la exactitud, precisión y recuerdo son buenos, tanto usando imágenes RGB como usando las 14 representaciones, pero la CNN que usa las 14 representaciones presenta el desempeño más alto. En el conjunto Stuttgart la exactitud es muy baja para RGB mientras que son muy altos la exactitud, la precisión y el recuerdo de la red que usa las 14 representaciones.

Para el resultado de imágenes procesadas en tiempo real, los autores reportan usando su robot para agricultura llamado *Bosch Deepfield Robotic BoniRob UGV* que cuenta con un CPU Intel i7 y una GPU NVIDIA GTX1080Ti. La CNN logra procesar 22 imágenes por segundo. Además se realizó un segundo experimento usando un Jetson TX2 que es un módulo embebido de NVIDIA para cómputo de inteligencia artificial, el cual realiza un procesamiento de casi 5 imágenes por segundo. Debido a que el Jetson TX2 es un módulo pequeño, la CNN se podría utilizar en un robot volador.

Concluyen los autores que su CNN es un buen enfoque para la segmentación de cultivos y malezas en tiempo real, que solo necesita imágenes RGB y datos obtenibles a partir de estas, con la intención de acelerar el entrenamiento y mejorar la generalización en nuevos cultivos.

Una de las observaciones sobre este trabajo es que los autores no dan muchos argumentos de por qué decidieron usar exactamente esas señales, cuando se habla de aprendizaje profundo normalmente se tiene la idea de usar los datos en bruto sin un preprocesemaiento (aunque no siempre es así), además cuando se habla de una CNN para segmentar o clasificar, se mencionan a los bordes (*edges* en inglés) de la siguiente manera. Un sistema de aprendizaje profundo

puede representar el concepto de una imagen de un objeto al combinar conceptos más simples como esquinas y bordes [13]. En la primera capa (en un sistema de aprendizaje profundo) se representa típicamente la presencia o ausencia de bordes en orientaciones y ubicaciones particulares en la imagen. La segunda capa típicamente detecta patrones mediante acomodos particulares de bordes idependientemente de las pequeñas variaciones de los bordes, las capas siguientes pueden detectar combinaciones de capas anteriores para detectar objetos [12].

Aunque existiera el caso de que una CNN no necesite detectar bordes, se entiende la idea de que una red neuronal profunda debe ser capaz de aprender a extraer de una imagen la información que necesita para clasificar o segmentar, aunque los autores mencionan que se hace con la intención de acelerar el entrenamiento y generalizar. Estas metas se pueden obtener de otras maneras, por ejemplo, para acelerar el entrenamiento se puede usar una CNN más chica, pero que que dicha CNN no comprometa el rendimiento o usar un *encoder* ya entrenado como AlexNet o VGG16Net y solo entrenar la parte del *decoder*, para generalizar mejor se pueden usar regularizadores o una técnica llamada *dropout* que mejora el desempeño de la CNN como consecuencia de una buena generalización.

En sus resultados los autores siempre comparan el desempeño obtenido con solo RGB vs RGB+NIR vs RGB + características extras pero no se compara al uso de sólo RGB vs RGB + índices de vegetación vs RGB + características que no sean índices de vegetación vs RGB + HSV vs HSV y más combinaciones de características, para mostrar si la mejora en el desempeño es consecuencia de los índices de vegetación, del espacio HSV, de los detectores de contorno, o de usar todas las características. Se tendrían que hacer más experimentos para obtener esa respuesta.

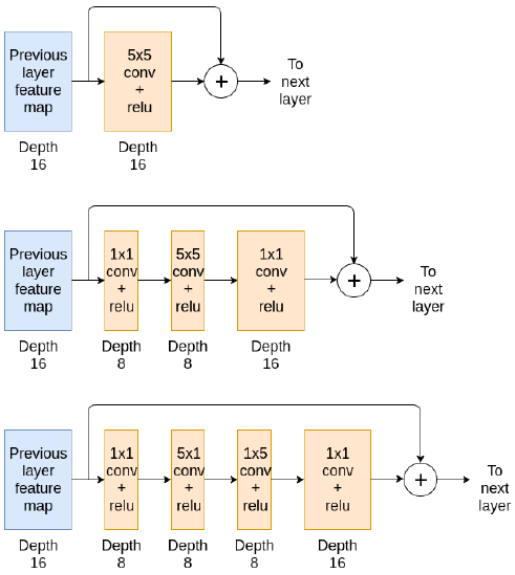
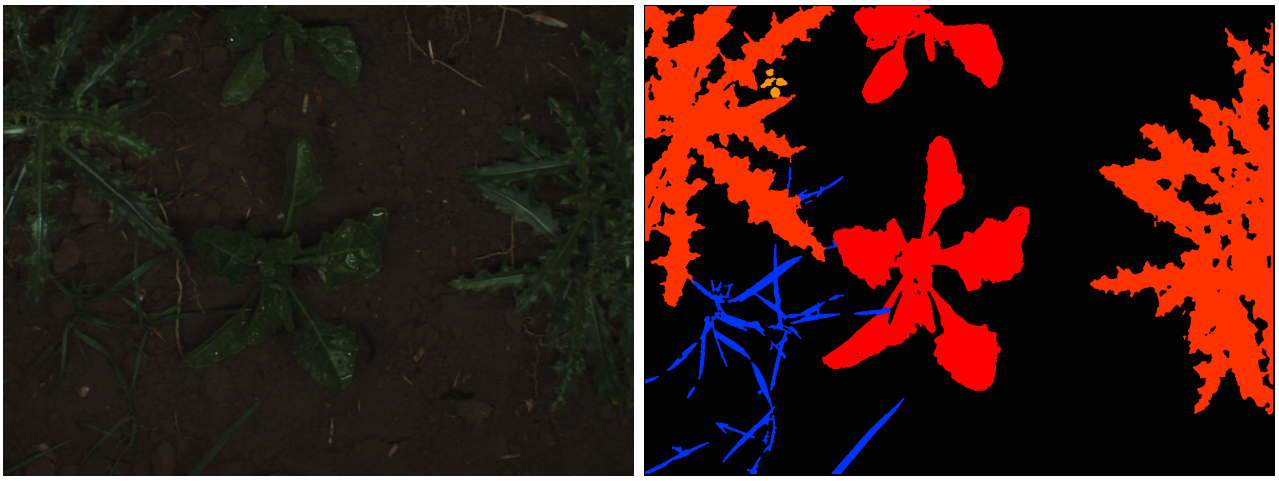


Figura 3.10: Construcción de un bloque residual para sustituir una capa convolucional de 5×5.



(a) Imagen a color. **(b)** *Ground Truth.*

Figura 3.11: Una imagen de Bonn *dataset* con su *Ground Truth*, el cultivo esta segmentado en rojo, los otros colores son diferentes tipos de maleza.

Dataset	Network	mIoU[%]	IoU[%]			Precision[%]			Recall[%]		
			Soil	Weeds	Crops	Soil	Weeds	Crops	Soil	Weeds	Crops
Bonn	$\mathcal{I}_{RGB}$	59.98	99.08	20.64	60.22	99.92	28.97	66.49	99.15	41.79	82.45
	$\mathcal{I}_{RGB}$+$\mathcal{I}_{NIR}$	76.92	99.29	49.42	82.06	99.88	52.90	84.19	99.33	**88.24**	97.01
	$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	**80.8**	**99.48**	**59.17**	**83.72**	**99.95**	**65.92**	**85.71**	**99.53**	85.25	**97.29**
Zurich	$\mathcal{I}_{RGB}$	38.25	96.84	14.26	3.62	96.95	14.96	3.78	96.88	35.35	45.86
	$\mathcal{I}_{RGB}$+$\mathcal{I}_{NIR}$	41.23	98.44	16.83	8.43	99.68	19.03	9.07	98.46	**51.27**	54.46
	$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	**48.36**	**99.27**	**23.40**	**22.39**	**99.90**	**31.43**	**23.05**	**99.36**	47.79	**88.74**
Stuttgart	$\mathcal{I}_{RGB}$	48.09	99.18	21.40	23.69	99.84	21.90	52.43	99.34	42.95	28.95
	$\mathcal{I}_{RGB}$+$\mathcal{I}_{NIR}$	55.82	98.54	23.13	45.80	99.85	25.28	68.76	98.69	**49.10**	57.84
	$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	**61.12**	**99.32**	**26.36**	**57.65**	**99.86**	**37.58**	**68.77**	**99.45**	46.90	**78.09**

Tabla 3.6: Resultados del desempeño de la segmentación a nivel de pixel.

Inputs	Nr. Images	mAcc[%]	Precision[%]		Recall[%]	
			Weeds	Crops	Weeds	Crops
$\mathcal{I}_{\text{RGB}}$	10	60.08	58.75	62.22	73.03	57.99
	20	71.38	61.38	81.42	76.72	64.58
	50	74.08	63.00	85.42	74.14	68.34
	100	82.26	65.50	85.59	74.97	69.91
$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	10	83.90	69.23	80.73	71.71	76.18
	20	85.31	75.85	79.12	67.67	84.01
	50	86.25	76.50	85.24	71.55	84.33
	100	**89.55**	**85.89**	**89.52**	**89.69**	**86.76**

(a) Resultados sobre imágenes de Zurich.

Inputs	Nr. Images	mAcc[%]	Precision[%]		Recall[%]	
			Weeds	Crops	Weeds	Crops
$\mathcal{I}_{\text{RGB}}$	10	71.76	93.88	52.59	56.57	81.10
	20	72.30	94.40	57.72	51.43	86.35
	50	72.97	94.33	59.70	54.11	87.69
	100	73.34	95.20	63.26	56.36	87.66
$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	10	81.40	89.48	64.42	78.74	79.69
	20	81.84	87.83	68.18	81.45	74.61
	50	86.75	94.68	71.34	83.22	90.23
	100	**91.88**	**95.45**	**86.58**	**89.08**	**91.43**

(b) Resultados sobre imágenes de Stuttgart.

Tabla 3.7: Resultados de adaptarse a nuevos cultivos.

Capítulo 4

Metodología

4.1. Obtener imágenes con el dron

El conjunto de datos se capturó en un terreno ubicado en el ejido de Xalostoc, municipio de Ayala, Morelos, México a principios de febrero de 2017 por la mañana (alrededor de las 9:30 a.m.) los árboles de higo tenían tres años de edad, una altura de dos metros aproximadamente y estaban en temporada de producción de fruto. La distancia promedio entre árboles es de 2.5 metros. Sobre el suelo había mucha maleza, rocas, residuos y algunos caminos de tierra.

Para capturar las imágenes se usó una cámara montada en el dron DJI Phantom 4. La cámara captura imágenes de 4000×3000 pixeles, además se usó la aplicación para sistemas iOS llamada *DroneDeploy* para realizar una misión de vuelo sobre el cultivo (aproximadamente 20 metros). Al final de esta misión de vuelo se obtuvieron 110 imágenes a color. En la figura 4.1 se muestra el dron usado y la misión de vuelo hecha, esta misión tomó imágenes de todo el cultivo que tiene un área aproximada de una hectárea.

Las imágenes tomadas con una cámara montada en un dron en una misión de vuelo a baja altura ofrecen información visual muy valiosa. Se observa la imagen con buena calidad, con un contraste en los árboles, la maleza, las sombras y el suelo. Las razones por las que se tomaron las fotografías en las condiciones que se muestran son las siguientes:

- La altura mínima permitida por DroneDeploy para hacer una misión de vuelo y tomar fotografías es de 20 metros.
- A esa altura con un solapamiento del 50 % entre las fotografías se obtienen 110 imágenes de tamaño 4000×3000 pixeles que es una gran cantidad de información.
- A esa altura se tienen entre 2 y 3 centímetros por pixel dependiendo de la altura, siendo una buena resolución. Una hoja de higo mide en promedio 25 centímetros de largo y 20 centímetros de ancho.

Por otro lado, usar cámaras fotográficas para trabajos de agricultura de precisión y procesamiento de imágenes es muy atractivo ya que las cámaras fotográficas son sensores relativamente baratos y requieren bajo consumo de energía en comparación con cámaras que capturan múltiples espectros que también se usan para estas tareas. Con la CNN ya entrenada, este método de segmentación de árboles de higo puede operar en equipo de cómputo y equipo de captura de imágenes relativamente barato.

Figura 4.1: Izquierda, el dron usado. Derecha, plan de vuelo sobre el cultivo.

4.2. Etiquetado manual de los objetos de interés en las imágenes con precisión de pixel para la generación del *ground truth* y creación de un conjunto de datos

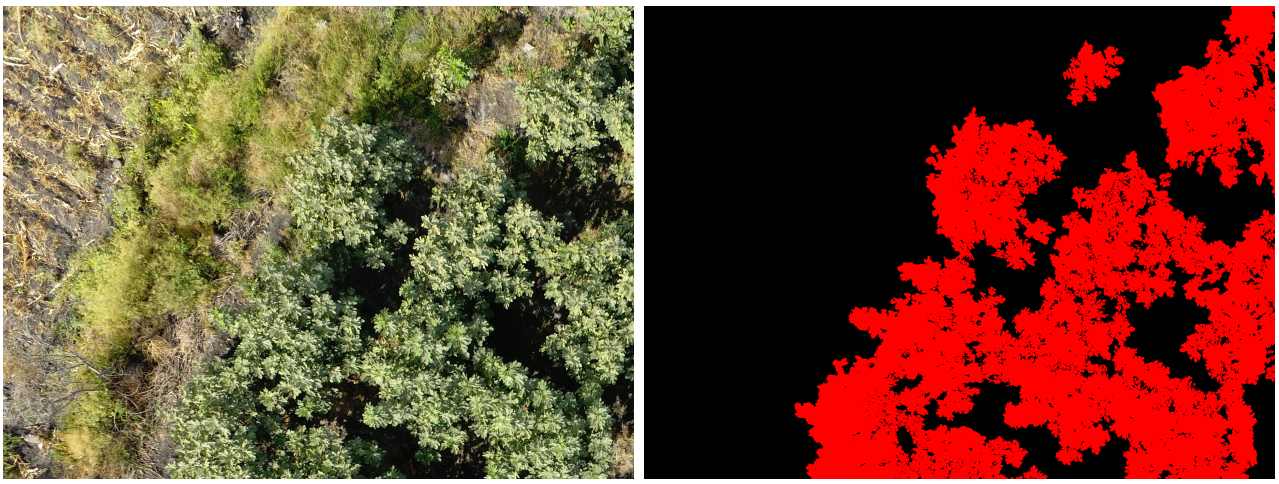
Se seleccionaron un total de 10 imágenes distribuidas por toda el área y se recortaron de tamaño 2000×1500 pixeles, la segmentación se realizó manualmente con precisión de pixel usando la herramienta de marcado *Image annotation tool with image masks* [14]. Este proceso tomó en promedio 8 horas por imagen, dicha segmentación fue hecha y verificada por dos expertos en visión por computadora y un experto en ciencias agropecuarias especializado en nutrición vegetal, pero aún con estas consideraciones podría haber errores en la segmentación por la complejidad de las imágenes. La imagen resultante de la segmentación sobre una imagen a color en el espacio de color RGB de tamaño 2000×1500 pixeles, es una imagen de 2000×1500 pixeles donde los pixeles pertenecientes a hojas de árbol de higo están coloreados en rojo y el resto de la imagen en color negro, el espacio de color de esta imagen es *8-bit color*, esta segmentación manual será considerada como la segmentación correcta y se le conoce como *ground truth* (GT).

La figura 4.2 muestra un ejemplo de una imagen recortada y su segmentación manual.

Para pasar las 10 imágenes de *8-bit color* con color rojo a *8-bit* en escala de grises donde el color rojo pasa a color blanco, se usó el *software* de procesamiento de imágenes conocido como *Fiji* [17]. Para pasar de color rojo a color blanco se usó la opción *Image* $\rightarrow$ *Adjust* $\rightarrow$ *Threshold* y para pasar la imagen a *8-bits* se usó la opción *Image* $\rightarrow$ *Type* $\rightarrow$ *8-bit*. En la figura 4.3 se muestra el resultado de pasar el etiquetado de color rojo en *8-bit color* a *8-bits*.

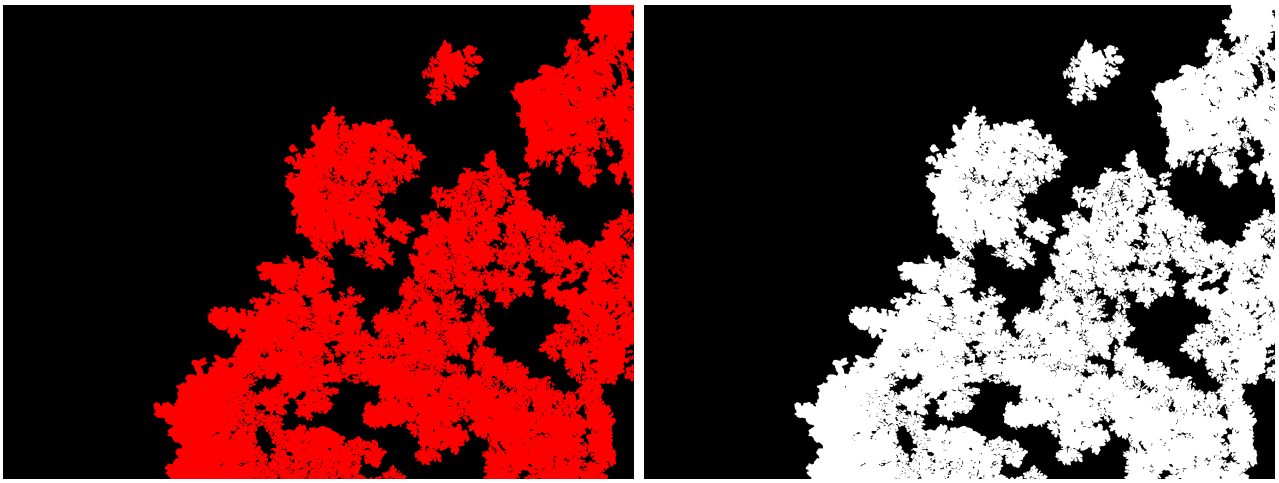
Como a la CNN diseñada con los hiperparámetros finales se le dan como entrada imágenes de 128×128 pixeles, se deben recortar las 10 imágenes de 2000×1500 pixeles. La manera de recortar es con un pequeño código hecho en C++ con la librería OpenCV [4], este programa recorta en tamaño 128×128 pixeles dejando un traslape del 70 %, este traslape se deja tanto en vertical como en horizontal, en la figura 4.4 se observa un ejemplo de recorte con traslape. Por imagen se obtienen 1938 recortes, por las 10 imágenes se obtienen 19380.

En la práctica, cuando se intenta aplicar aprendizaje profundo, puede darse el caso de tener un pequeño conjunto de datos para hacer el entrenamiento. Cuando los datos son imágenes se puede hacer un *aumento de datos* que consiste en deformar de distintas maneras las imágenes de entrenamiento para incrementar la cantidad de estas. En nuestro caso no deformamos las imágenes sólo las recortamos en tamaño 128×128 con traslape y al final juntamos estos recortes



(a) Recorte de 2000 × 1500 pixeles a color. **(b)** Segmentación manual.

Figura 4.2: Ejemplo de la imagen recortada con su segmentación manual, se aprecia el tipo de problema que se trata de resolver y sus dificultades.



(a) Segmentación manual usando [14]. **(b)** Resultado después de usar [17].

Figura 4.3: Ejemplo del procesamiento del GT (hecho con la herramienta *Image annotation tool with image masks*) para que quede en *8-bits* en blanco y negro (mediante la herramienta *Fiji*).

segmentados por la CNN en la imagen de tamaño original obteniendo la imagen segmentada y la evaluación con las métricas.

4.3. Diseñar y entrenar la CNN

El modelo de CNN está inspirado en los modelos *Deconvolution Network* [10] y *SegNet* [8] con la diferencia de tener menos filtros por capa convolucional, menos capas convolucionales, de agrupamiento y *UpSampling*. En el caso específico de *SegNet*, la CNN propuesta no cuenta con una activación *softmax* en la última capa (la capa de salida) que es una función de activación muy usada para clasificaciones y segmentaciones de varias clases, en cambio la última capa tiene una función de activación *sigmoid* que es muy usada para realizar clasificación binaria [6], lo cuál es nuestro caso ya que el objetivo es clasificar pixeles en una imagen como perteneciente o no a hoja de árbol de higo.

Lo que sí comparten estos modelos con el nuestro es que tienen capas convolucionales y de agrupamiento para ir reduciendo el tamaño del mapa de características obtenido con la imagen de entrada, seguido de capas de convolución y *UpSampling* para incrementar el tamaño del

Figura 4.4: Seis recortes consecutivos con su GT de tamaño 128 × 128 pixeles con traslape del 70 %.

mapa de características hasta ser del mismo tamaño que la imagen de entrada. En la figura 4.5 se muestra un esquema del modelo de red empleado. El código fuente de la CNN que se propone y el conjunto de datos de higo se encuentran disponibles en `https://github.com/jofuepa/fig-dataset`.

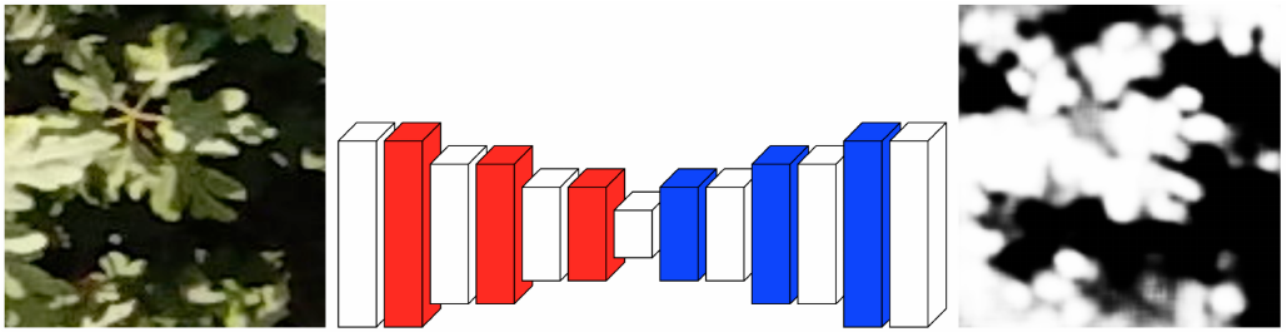


Figura 4.5: Esquema del modelo de CNN que se propone, la entrada es un recorte de imagen y la salida es su segmentación.

En el proceso de diseñar una red neuronal profunda es muy común empezar con una red pequeña para después ajustar hiperparámetros como el número de filtros, épocas de entrenamiento y número de capas. Las características de la CNN en un **estado inicial**, lista para ajustar hiperparámetros se enlistan a continuación y en la tabla 4.1.

Características de las capas convolucionales:

- 16 filtros de 3 × 3 en el primer *layer*.
- 16 filtros de 3 × 3 en el penúltimo *layer*.
- 3 filtros de 3 × 3 en el *layer* de salida.
- 8 filtros de 3 × 3 para el resto de los *layers*.
- El salto de todos los filtros es de 1.
- Se usa *Zero-Padding* en todas las capas convolucionales.
- El último *layer* tiene activación *sigmoide* y el resto activación *ReLU*.

Características de las capas *Maxpooling*:

- El tamaño de ventana (también conocido como *pool size*) es de 2×2.
- El salto de esta ventana es de 2.

Características de las capas *UpSampling*:

- Repite los datos de las filas y columnas en tamaño 2×2, para más información consulte 2.6 y [7].

Otras características de la CNN:

- Inicializa los parámetros de la red con la *distribución uniforme de glorot* [20].
- Como función de optimización tiene Adadelta [21].
- Como función de pérdida tiene *Binary crossentropy*, que es muy usado en clasificación binaria como es nuestro caso [6].
- Tamaño de batch de 32 imágenes.
- Las imágenes del *training set* se dan a la CNN en orden aleatorio.

	Entrada (recorte 224×224) RGB	
[16][3×3]		[224×224]
	MaxPooling	
[8][3×3]		[112×112]
	MaxPooling	
[8][3×3]		[56×56]
	MaxPooling	
[8][3×3]		[28×28]
	UpSampling	
[8][3×3]		[56×56]
	UpSampling	
[16][3×3]		[112×112]
	UpSampling	
[3][3×3]		[224×224]
	Salida (224×224) 3-canales	

Tabla 4.1: Descripción de la propuesta de CNN con los hiperparámetros iniciales. La columna izquierda indica [número de filtros][tamaño del filtro]. La columna derecha indica [tamaño de la salida de la convolución].

4.4. Experimentos hechos con la CNN

4.4.1. Entrenar y probar la CNN para modificar los hiperparámetros

Se entrenó la CNN para modificar los hiperparámetros sobre las capas convolucionales. La red se entrenó con el conjunto de datos dividido en 80 %, 15 %, 5 % y 80 %, 10 %, 10 % (entrenamiento, validación y prueba respectivamente) del *dataset*.

También se entrenó la CNN para determinar el tamaño ideal de los recortes en las imágenes de entrada y el número de épocas de entrenamiento (una época es una iteración sobre el conjunto de datos de entrenamiento). Para determinar los recortes de imagen se recortó el conjunto de datos en tamaños de 32×32, 64×64, 128×128 y 224×224 y para determinar el número de épocas de entrenamiento se entrenó la CNN entre un mínimo de 100 épocas y un máximo de 400 épocas.

4.4.2. Validar el modelo diseñado mediante k-fold cross-validation

Una buena práctica para validar un modelo de CNN con un conjunto de datos pequeño, como es nuestro caso, es usar *K-Fold validation* que consiste en dividir el conjunto de datos disponible en K partes iguales, típicamente se entrena la CNN con $k-1$ partes y se valida (prueba) con la parte restante, esto se hace K veces para obtener conjuntos de datos de entrenamiento y prueba diferentes, esta instancia en particular se conoce como dejar-uno-fuera [2]. Se usó dejar-uno-fuera que no permite la estratificación. El conjunto de datos que tenemos no permite balancear fácilmente la proporción de casos positivos (hojas de higo) y negativos (NO hojas de higo) cuando se seleccionan las imágenes de 2000×1500 para las instancias. Se puede obtener un puntaje de esta validación que es el promedio de cada uno de los puntajes (un puntaje puede ser la exactitud) de cada validación hecha. En la figura 2.6 se muestra una instancia de *K-Fold validation* con $K = 3$. Con este método se puede ver como se comporta el modelo sobre distintos datos de entrenamiento, si es un buen modelo entonces tendrá un desempeño muy bueno sobre cada *fold* de entrenamiento y prueba.

Para realizar la validación sobre el modelo diseñado se ha empleado una validación cruzada sobre el conjunto de las 10 imágenes antes mencionadas con su segmentación manual recortadas con traslape del 70 %. En esta validación cruzada se usa un $K = 10$ para dividir nuestro conjunto en las 10 imágenes originales dejando dos imágenes recortadas fuera del conjunto de imágenes de entrenamiento para usarlas como conjunto de imágenes de prueba. El total de conjuntos de entrenamiento y validación con 8 imágenes de entrenamiento y 2 de prueba son las combinaciones 8 en 10 ($C\binom{10}{8}$) siendo un total de 45 conjuntos de datos para realizar la *K-Fold validation*.

Para nuestra *K-Fold validation* se entrena por 120 épocas la CNN con el conjunto de datos de entrenamiento y se prueba con el conjuto de datos de prueba, después se calcula la exactitud sobre el conjunto de datos de prueba, por último se reinicia el entrenamiento, se entrena, prueba y calcula la exactitud nuevamente sobre otra combinación distinta y así sucesivamente hasta completar los 45 *folds*.

4.4.3. Escoger una instancia interesante de k-fold cross-validation y entrenar 120 épocas para calcular el desempeño

En este experimento se siguen los siguientes pasos:

- Escoger alguna combinación que tenga dos imágenes de prueba que se vean interesantes para segmentarlas.
- Entrenar la CNN 120 épocas.
- Obtener gráficas de exactitud y pérdida.
- Calcular los resultados usando las métricas que aparecen en la tabla 5.2.

Una imagen es interesante para segmentar si presenta elementos ajenos al cultivo como lo son objetos usados por los agricultores, animales y plantas ajenas al cultivo. En general, elementos que no son del cultivo de higo pero que no sería raro que estén presentes.

4.4.4. Reconstruir la imagen de tamaño original segmentada a partir de los recortes predichos por la CNN

Para reconstruir la imagen de tamaño original ya segmentada a partir de los recortes que segmenta la CNN se sigue la siguiente estrategia:

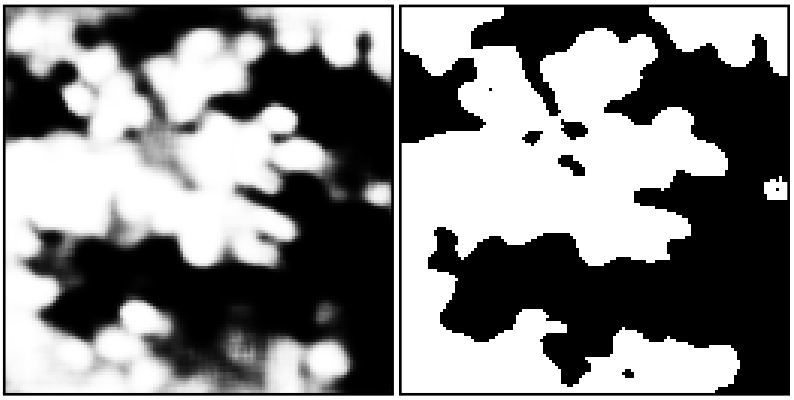
Para cada pixel que pertenece a la imagen de tamaño original segmentada:

1. Tomar todos los recortes segmentados con traslape donde aparece cada pixel de la imagen de tamaño original.
2. Promediar los valores donde aparezca cada pixel de la imagen original en los recortes traslapados.
3. El valor promediado de los recortes traslapados será el valor del pixel en la imagen original.

4.5. Postprocesamiento

Los niveles de intensidad en la imagen están entre 0 y 1 debido a la activación *sigmoide*, esto provoca que las imágenes tengan varios niveles de gris normalizados en el intervalo $[0, 1]$. Para obtener una imagen en dos niveles se realiza un postprocesamiento mediante la siguiente binarización sobre cada pixel clasificado por la CNN.

$$I_{binarizada}(i,j) = \begin{cases} 1 & \text{si } I_{predicha}(i,j) > 0.5 \\ 0 & \text{si } I_{predicha}(i,j) \leq 0.5 \end{cases}$$



(a) predicción de la CNN. **(b)** predicción postprocesada.

Figura 4.6: Ejemplo del postprocesamiento a las predicciones de la CNN para que queden binarizadas.

Capítulo 5

Resultados

5.1. Métricas de evaluación

Para medir el desempeño de la CNN se necesita conocer a nivel de pixel si fue clasificado como pixel perteneciente a hoja de higo o no. Teniendo el GT como referencia, el pixel puede caer en uno de los siguientes cuatro casos:

1. Verdadero positivo **(VP)**: Es aquel pixel clasificado como **hoja de higo** por la CNN y marcado como **hoja de higo** en el *Ground Truth* (GT).
2. Verdadero negativo **(VN)**: Es aquel pixel clasificado como **NO hoja de higo** por la CNN y marcado como **NO hoja de higo** en el GT.
3. Falso positivo **(FP)**: Es aquel pixel clasificado como **hoja de higo** por la CNN y marcado como **NO hoja de higo** en el GT.
4. Falso negativo **(FN)**: Es aquel pixel clasificado como **NO hoja de higo** por la CNN y marcado como **hoja de higo** en el GT.

En la siguiente tabla se muestra de manera más clara la relación de pixeles clasificados de la CNN respecto al GT.

Clasificado como		*Ground truth* (GT)
Hoja de higo	NO hoja de higo	
VP	FN	Hoja de higo
FP	VN	NO hoja de higo

Tabla 5.1: Casos posibles al clasificar los pixeles usando el *ground truth* para comparar. Es deseable que los valores de la diagonal compuesta por VP y VN sean altos y los valores de la diagonal compuesta por FN y FP sean bajos.

Una vez que se conocen los VP, VN, FP y FN se obtienen las métricas de la tabla 5.2 que son muy usadas en clasificadores binarios.

Nombre de la métrica	Fórmula
Tasa de éxito global o **Exactitud**	$\frac{VP+VN}{VP+VN+FP+FN}$
Tasa de verdaderos negativos o **Especificidad**	$\frac{VN}{VN+FP}$
Valor predictivo positivo o **Precisión**	$\frac{VP}{VP+FP}$
Tasa de verdaderos positivos, sensibilidad o **Recuerdo**	$\frac{VP}{VP+FN}$
Valor predictivo negativo (VPN)	$\frac{VN}{VN+FN}$
Medida F	$\frac{2\times VP}{2\times VP+FN+FP}$
Coeficiente de correlación de Matthews	$\frac{VP\times VN-FP\times FN}{\sqrt{(VP+FP)(VP+FN)(VN+FP)(VN+FN)}}$

Tabla 5.2: Métricas con su fórmula, usadas para medir el desempeño de la CNN. Un buen resultado tiene valores cercanos a 1 mientras que un mal resultado tiene valores cercanos a 0.

5.2. CNN con la selección final de hiperparámetros

Lo que a continuación se muestra son los hiperparámetros finales para la CNN. La diferencia con los hiperparámetros iniciales es que estos ofrecen mejores resultados cualitativos segmentando mejor el color verde del suelo como algo que no es de interés. En cuanto a los resultados cuantitativos mejora la exactitud en aproximadamente 2 %.

Los hiperparámetros cambiados se muestran en **negritas** y el cambio específico se muestra **subrayado**. Los cambios en los hiperparámetros se realizaron a prueba y error. En la tabla 5.3 se muestra la descripción de la CNN propuesta con la selección final de hiperparámetros.

Hiperparámetros de las capas convolucionales:

- **32 filtros de 7×7 en el primer *layer***.
- 16 filtros de 3×3 en el penúltimo *layer*.
- **1 filtro de 3×3 en el *layer* de salida**.
- 8 filtros de 3×3 para el resto de los *layers*.
- El salto de todos los filtros es de 1.
- Se usa *Zero-Padding* en todas las capas convolucionales.
- El último *layer* tiene activación *sigmoide* y el resto activación $ReLU$.

Hiperparámetros de las capas Maxpooling: (sin cambios)

- El tamaño de ventana (también conocido como *pool size*) es de 2×2.
- El salto de esta ventana es de 2.

Hiperparámetros de las capas UpSampling: (sin cambios)

- Repite los datos de las filas y columnas en tamaño 2×2. Para más información consulte 2.6 y [7].

Otros hiperparámetros de la CNN: (sin cambios)

- Inicializa los parámetros de la red con la *distribución uniforme de glorot* [20].

- Como función de optimización tiene Adadelta [21].
- Como función de pérdida tiene *Binary crossentropy* que es muy usada en clasificación binaria como es nuestro caso [6].
- Tamaño de batch de 32 imágenes.
- Las imágenes del *training set* se dan a la CNN en orden aleatorio.

	Entrada (recorte 128×128 RGB)	
$[32][7 \times 7]$		$[128 \times 128]$
	MaxPooling	
$[8][3 \times 3]$		$[64 \times 64]$
	MaxPooling	
$[8][3 \times 3]$		$[32 \times 32]$
	MaxPooling	
$[8][3 \times 3]$		$[16 \times 16]$
	UpSampling	
$[8][3 \times 3]$		$[32 \times 32]$
	UpSampling	
$[8][3 \times 3]$		$[64 \times 64]$
	UpSampling	
$[1][3 \times 3]$		$[128 \times 128]$
	Salida (128×128 1 canal)	

Tabla 5.3: Descripción de la propuesta de CNN con la selección final de hiperparámetros. La columna izquierda indica [número de filtros][tamaño del filtro]. La columna derecha indica [tamaño de la salida de la convolución].

5.3. Resultados de la validación cruzada

Para la validación cruzada se obtuvo la exactitud, precisión, recuerdo, especificidad, valor predictivo negativo, medida F y coeficiente de correlación de Matthews sobre el *test set* para cada una de las 45 combinaciones después de entrenar por 120 épocas. En la figura 5.1 se muestran las 45 exactitudes, ambas representan los mismos valores, lo que cambia es el intervalo del eje y.

A los 45 valores de cada una de las 7 métricas se les calculó el promedio y la desviación estándar que se pueden ver en la tabla 5.4.

Nombre de la métrica	**Promedio**	**Desviación estándar**
Tasa de éxito global o **Exactitud**	0.9385	0.0127
Tasa de verdaderos negativos o **Especificidad**	0.9176	0.0308
Valor predictivo positivo o **Precisión**	0.9394	0.0323
Tasa de verdaderos positivos, sensibilidad o **Recuerdo**	0.9499	0.0288
Valor predictivo negativo (VPN)	0.9309	0.0314
Medida F	0.9434	0.0174
Coeficiente de correlación de Matthews	0.8688	0.0211

Tabla 5.4: Promedio y desviación estándar de los 45 valores de cada métrica obtenidos de la validación cruzada.

5.4. Resultado de escoger una instancia interesante de k-fold cross-validation

Las imágenes usadas en este experimento son las mostradas en la figura 5.2. Éstas poseen elementos interesantes para segmentar. La imagen DJI_0010_A tiene un cultivo ajeno al de higos en la parte izquierda de la imagen y maleza alta. Por otro lado, la imagen DJI_0051_A tiene objetos ajenos al cultivo como unas cubetas usadas por los agricultores, sombras tenues sobre las hojas de los árboles y el suelo es ligeramente verde. En la figura 5.3 se pueden apreciar los elementos descritos anteriormente en los recortes de ambas imágenes con su respectiva predicción por la CNN postprocesada y GT. Los resultados númericos se muestran en la tabla 5.5.

Nombre de la métrica	**Promedio sobre DJI_0010_A y DJI_0051_A**
Exactitud	0.9384
Especificidad	0.9279
Precisión	0.9200
Recuerdo	0.9505
Valor predictivo negativo	0.9555
Medida F	0.9350
Coeficiente de correlación de Matthews	0.8770

Tabla 5.5: Resultados obtenidos sobre el *test set* con su valor numérico.

5.5. Las imágenes de tamaño original segmentadas reconstruidas

Se reconstruyeron las imágenes de tamaño original de cada una de las 45 instancias de la validación cruzada. Sobre las reconstrucciones se obtuvieron los valores de cada métrica, después se obtuvo el promedio y desviación estándar por cada métrica. En la tabla 5.6 se muestran los resultados. En la figuras 5.4, 5.5, 5.6 y 5.7 se muestran algunas imágenes de 2000×1500 reconstruidas, estas imágenes se obtuvieron de instancias de la validación cruzada.

Nombre de la métrica	**Promedio**	**Desviación estándar**
Tasa de éxito global o **Exactitud**	0.9409	0.0190
Tasa de verdaderos negativos o **Especificidad**	0.9198	0.0388
Valor predictivo positivo o **Precisión**	0.9404	0.0391
Tasa de verdaderos positivos, sensibilidad o **Recuerdo**	0.9527	0.0393
Valor predictivo negativo (VPN)	0.9353	0.0429
Medida F	0.9453	0.0241

Tabla 5.6: Promedio y desviación estándar de los 45 valores correspondientes a cada métrica obtenidos al reconstruir las imágenes de prueba de cada instancia de la validación cruzada.

Para el caso de la red neuronal diseñada se han presentado los resultados para la segmentación de árboles de higo. Este modelo falla en predecir como maleza pequeños árboles y maleza acumulada que no pertenecen al árbol de higo. También falla en predecir elementos que no son de interés como es el caso de unas cubetas usadas por los agricultores y plantas secas de una zona de cultivo vecina que están presentes en una de las imágenes. Para tratar ambas deficiencias podría incrementarse el número de estos ejemplos para el conjunto de datos de entrenamiento o la profundidad de la red.

5.6. Desempeño con diferentes tamaños de recorte

Se probó la CNN modelada con diferentes tamaños de recorte con la finalidad de demostrar que el desempeño de la CNN se mantiene. Los tamaños de recorte fueron 32×32 sin traslape obteniedo 29610 imágenes recortadas, 64×64 con traslape de 25 pixeles obteniendo 19380 imágenes recortadas y 256×256 con traslape de 220 pixeles obteniendo 1800 imágenes recortadas. Los diferentes traslapes son para mantener un aproximado al número de recortes de 128×128 (1938). En la tabla 5.7 se muestran los resultados, en todos los casos el desempeño es mayor a 89 % aunque la CNN modelada fue entrenada con recortes de 128×128.

	32×32	64×64	128×128	256×256
Exactitud	0.8955	0.9025	0.9384	0.9092
Especificidad	0.8304	0.8300	0.9279	0.8473
Precisión	0.8199	0.8300	0.9200	0.8425
Recuerdo	0.9769	0.9789	0.9505	0.9812
Valor predictivo negativo	0.9729	0.9752	0.9555	0.9779
Medida F	0.8893	0.8970	0.9350	0.9057

Tabla 5.7: Desempeño de la CNN modelada con diferentes tamaños de recorte.

5.7. Desempeño de la CNN propuesta contra SegNet-Basic

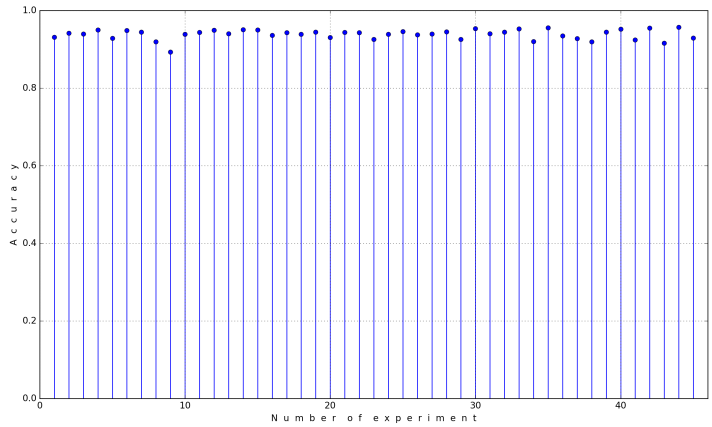
Fue entrenada con las imágenes propias la CNN llamada SegNet-Basic [9], que es una versión más pequeña que SegNet. Se midió el desempeño de SegNet-Basic para compararlo con la red modelada. En la tabla 5.8 se muestran las características de ambas CNNs. En la tabla 5.9 se muestra el desempeño de ambas CNNs, se logra un desempeño comparable con SegNet-Basic con la diferencia que la red modelada tiene menos parámetros y se entrena en menos tiempo.

	CNN modelada	**SegNet-Basic**
Capas convolucionales	7	8
Tamaño de los filtros	7×7 y 3×3	7×7
No. de filtros por capa	$32 - 8 - 8 - 8 - 8 - 16 - 1$	64 en todas las capas
MaxPooling	2×2 saltos de 2	2×2 saltos de 2
Upsampling	Repite los datos de filas y columnas en tamaño 2×2	Posiciones guardadas de *MaxPooling*
Número parámetros	$\sim 10,000$	~ 1.425 Millones
Épocas de entrenamiento (hasta converger)	128	62
Tiempo de entrenamiento (hasta converger)	45 minutos	3 horas

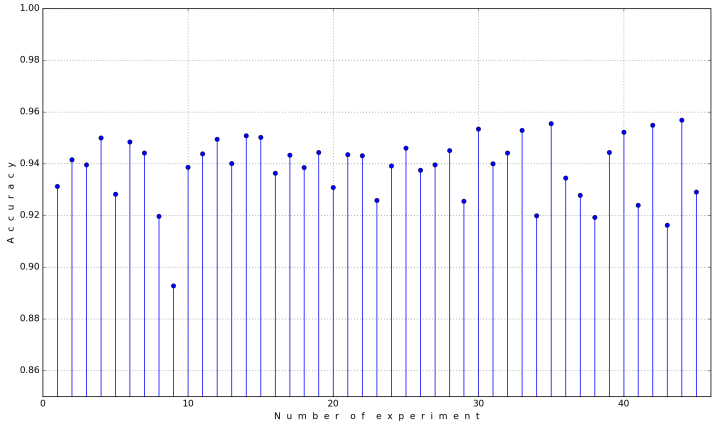
Tabla 5.8: Características de la CNN modelada y SegNet-Basic.

	CNN modelada	**SegNet-Basic**
Exactitud	0.9384	0.9382
Especificidad	0.9279	0.9433
Precisión	0.9200	0.9349
Recuerdo	0.9505	0.9322
Valor predictivo negativo	0.9555	0.9410
Medida F	0.9350	0.9335

Tabla 5.9: Desempeño de la CNN modelada y SegNet-Basic.



(a) Gráfica de 45 exactitudes con el eje y entre $[0, 1]$.



(b) Gráfica de 45 exactitudes con el eje y entre $[0.85, 1]$.

Figura 5.1: Gráfica de las 45 exactitudes obtenidas en la validación cruzada.

(a) DJI_0010_A.

(b) DJI_0051_A.

Figura 5.2: Imágenes seleccionadas para esta instancia.

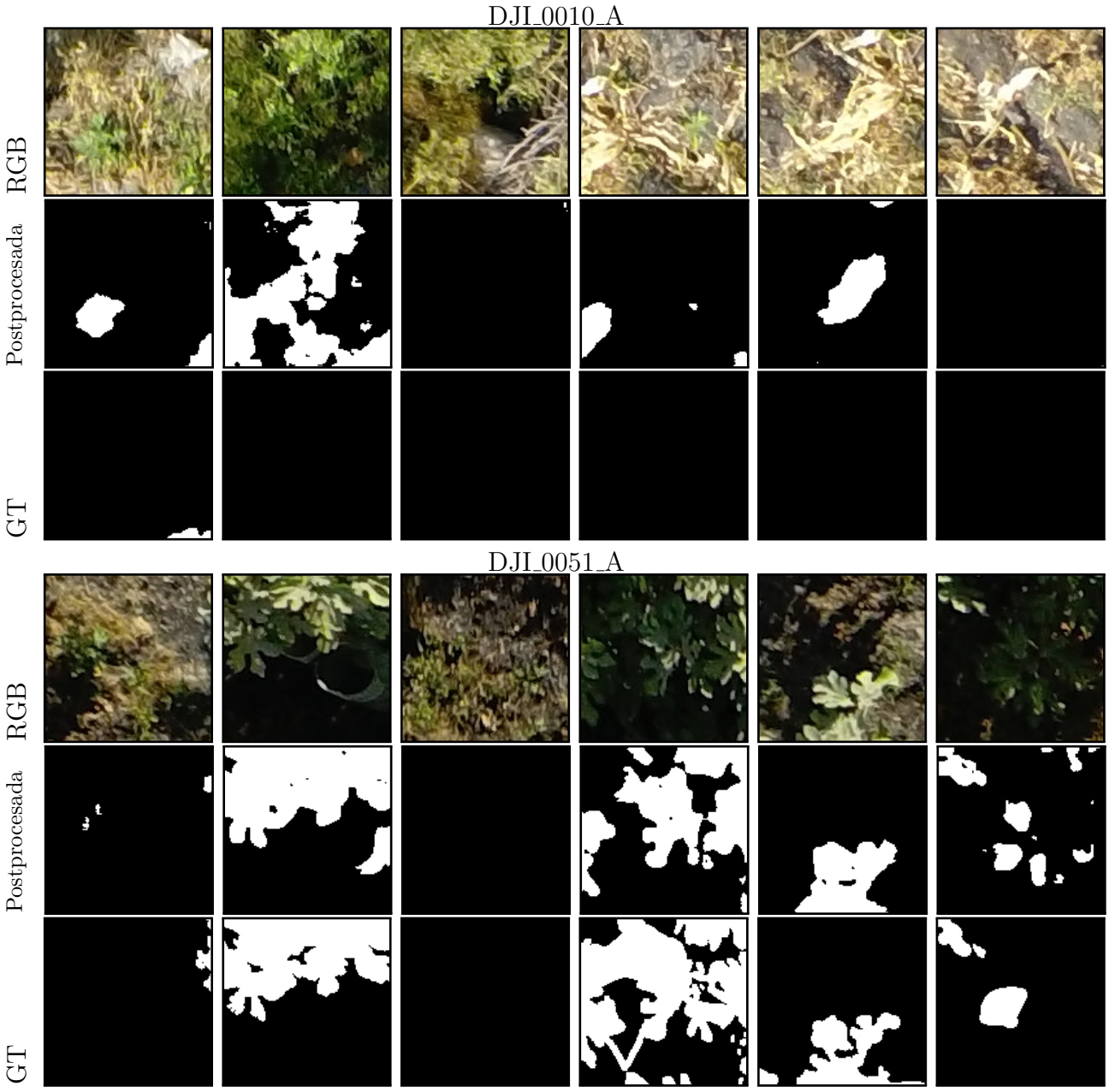
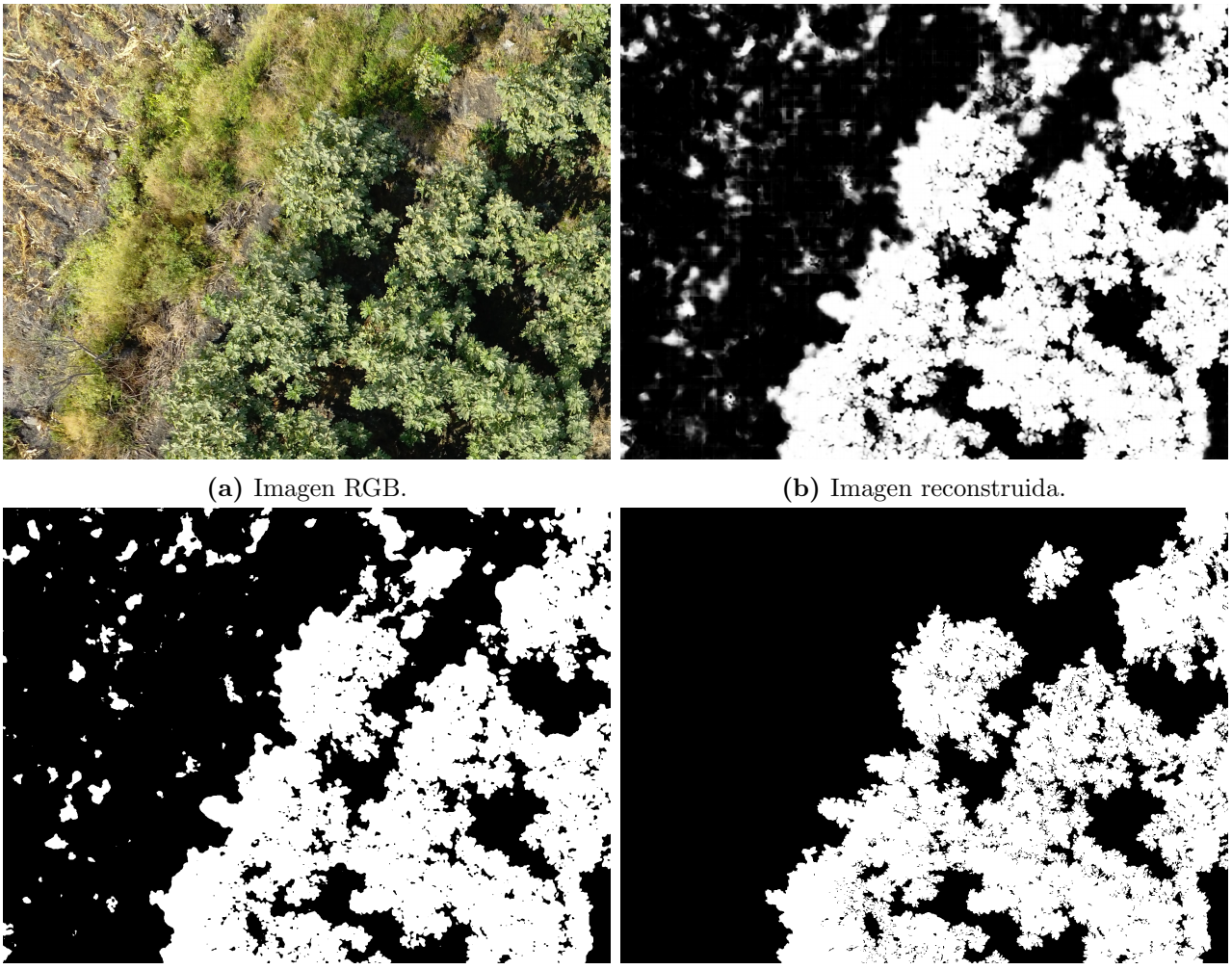


Figura 5.3: Seis recortes en DJI_0010_A y de DJI_0051_A, interesantes de segmentar, de tamaño 128×128 pixeles con sus predicciones hechas por la CNN postprocesadas y GT.



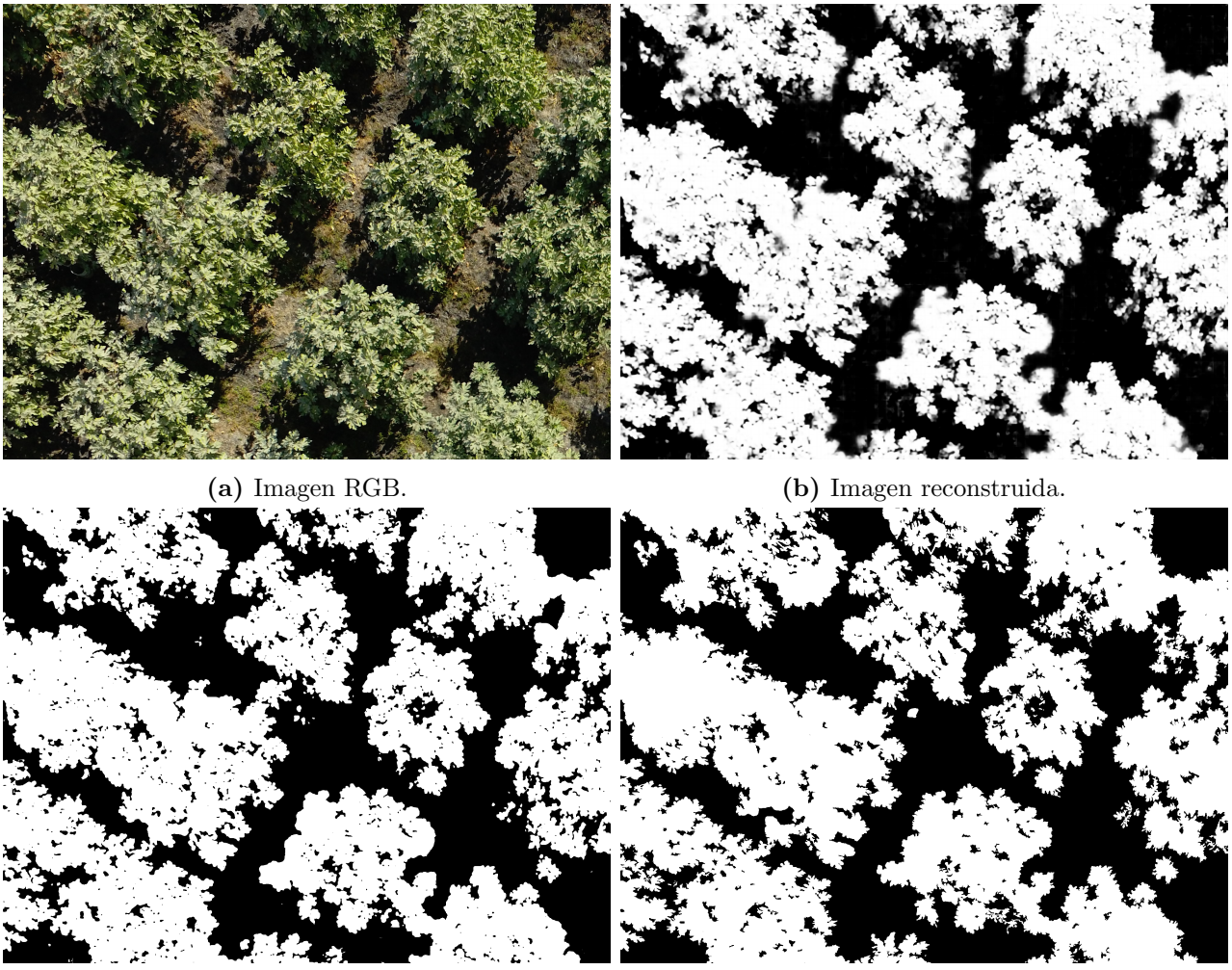
(a) Imagen RGB.

(b) Imagen reconstruida.

(c) Reconstrucción binarizada.

(d) *Ground Truth.*

Figura 5.4: Imagen a color, imagen reconstruida a partir de recortes de 128 × 128 segmentados por la CNN, reconstrucción binarizada y GT de DJI_0010_A obtenida de una instancia de la validación cruzada.



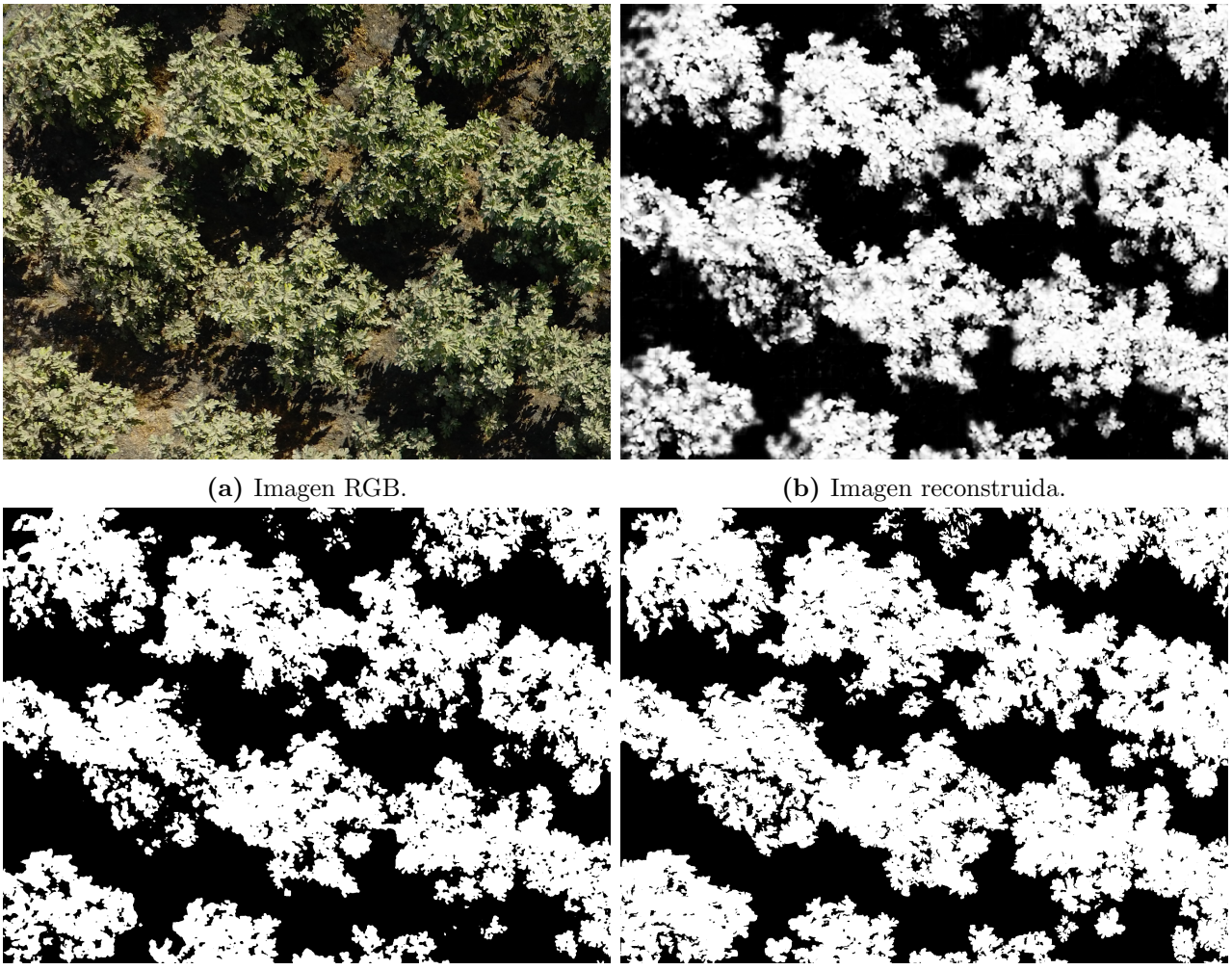
(a) Imagen RGB.

(b) Imagen reconstruida.

(c) Reconstrucción binarizada.

(d) *Ground Truth.*

Figura 5.5: Imagen a color, imagen reconstruida a partir de recortes de 128 × 128 segmentados por la CNN, reconstrucción binarizada y GT de DJI_0051_A obtenida de una instancia de la validación cruzada.



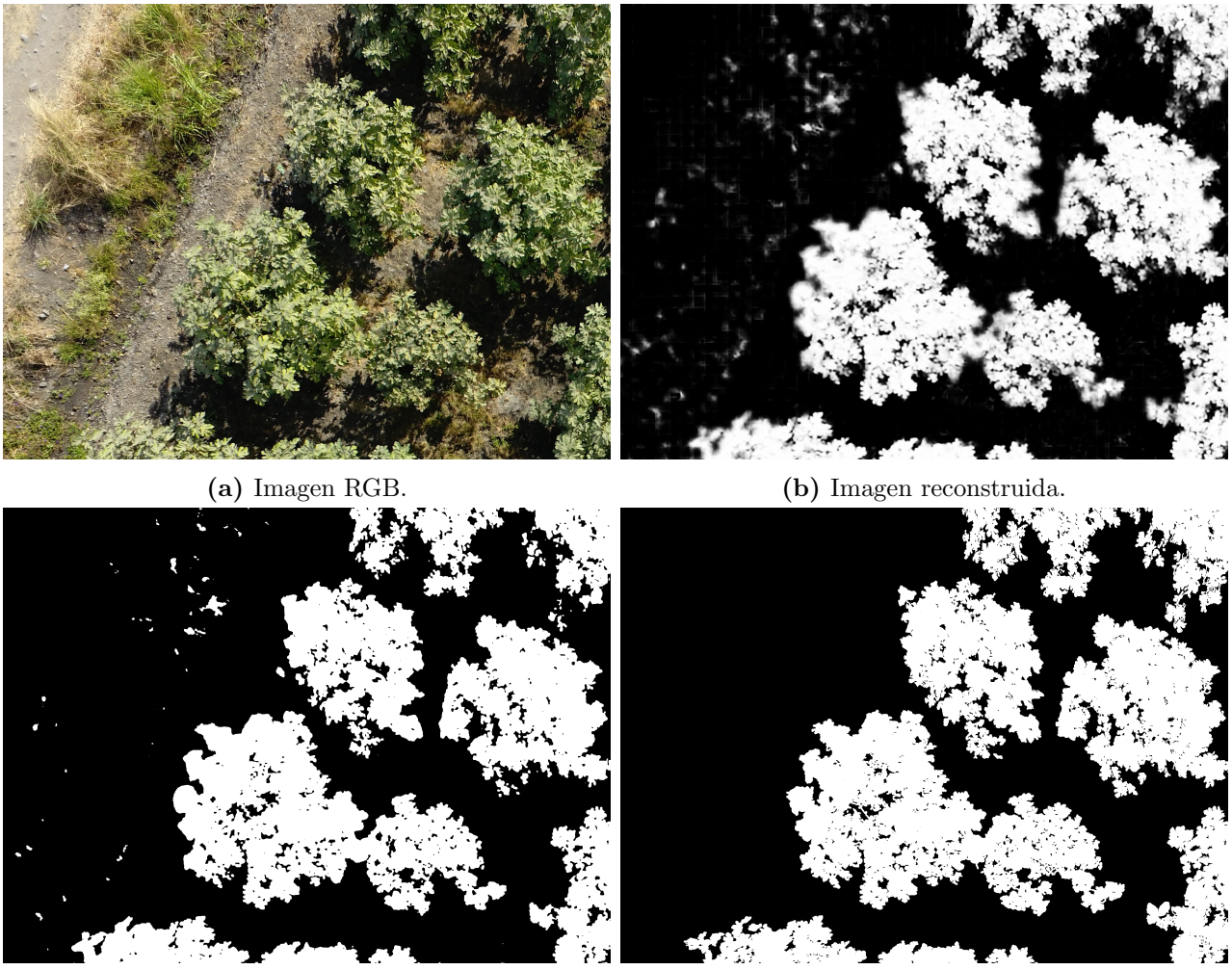
(a) Imagen RGB.

(b) Imagen reconstruida.

(c) Reconstrucción binarizada.

(d) *Ground Truth.*

Figura 5.6: Imagen a color, imagen reconstruida a partir de recortes de 128 × 128 segmentados por la CNN, reconstrucción binarizada y GT de DJI_0098_A obtenida de una instancia de la validación cruzada.



(a) Imagen RGB.

(b) Imagen reconstruida.

(c) Reconstrucción binarizada.

(d) *Ground Truth.*

Figura 5.7: Imagen a color, imagen reconstruida a partir de recortes de 128 × 128 segmentados por la CNN, reconstrucción binarizada y GT de DJI_0101_A obtenida de una instancia de la validación cruzada.

Capítulo 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

En este trabajo de tesis se implentó una red neuronal convolucional profunda con la finalidad de segmentar de manera *offline* (que no segmenta en tiempo real durante la misión de vuelo) árboles de higo en imágenes aéreas. Las imágenes fueron tomadas por un dron en una misión de vuelo semiautomática. Estas imágenes presentan características que dificultan su segmentación como son diferentes niveles de iluminación, suelo de diferentes colores, maleza y objetos ajenos al cultivo de higo. Usar redes neuronales para tratar este tipo de problemas es lo más adecuado, dado que han hecho grandes avances en la resolución de problemas que han resistido otros intentos de resolución por parte de la comunidad científica durante los últimos años. Además que las redes neuronales usan los datos en su forma original sin necesitar un preprocesamiento que puede provocar la pérdida de información relevante.

En general el desempeño fue por arriba del 90 % para las métricas de evaluación que se usaron cumpliendo con los objetivos y es comparable con el estado del arte en desempeño y tiempo de entrenamiento. Además se comparó el desempeño con SegNet-Basic teniendo un desempeño similar, con la diferencia de que la propuesta de CNN tiene menos parámetros y se entrena en menos tiempo. También se mostró que el modelo de CNN es robusto mediante la validación cruzada (teniendo una exactitud promedio de 93 %) y, usando diferentes tamaños de recorte como entrada de la CNN y midiendo su desempeño (con un desempeño promedio por cada métrica por arriba de 89 %). El trabajo realizado presenta un avance para el desarrollo de herramientas de agricultura de precisión que en un futuro serán accesibles a los agricultores. También este trabajo pone el conjunto de datos y CNN al alcance de la comunidad científica.

Lo mencionado anteriormente muestra los resultados de usar el aprendizaje profundo para la segmentación de cultivos en campo abierto en condiciones de iluminación no controladas con un pequeño conjunto de datos de entrenamiento. La CNN modelada tiene una cantidad menor de parámetros (en orden de miles) en comparación con las estado del arte (en orden de millones), se entrena en menos tiempo y los resultados son comparables. El problema de segmentar las imágenes de árbol de higo usadas en este trabajo es inédito en el estado del arte.

6.2. Trabajo a futuro

Una red neuronal convolucional profunda que realiza segmentación sobre cultivos en campo abierto puede ser usada para lidiar con problemas de interés para la comunidad científica del área de agricultura de precisión, como detección de enfermedades en los cultivos, estrés hídrico, calidad del suelo entre otros. Para los problemas anteriores, segmentar el cultivo constituye una etapa inicial necesaria.

La segmentación de cultivos es un paso importante para otras tareas de agricultura de precisión, como es la clasificación de hojas en dos clases: hoja sana y hoja enferma. Esta clasificación permitiría estimar el estado de salud del cultivo. También se podrían detectar anomalías en los cultivos como deficiencia de nitrógeno o estrés hídrico sólo con información obtenida de imágenes del cultivo. Para realizar estas tareas es muy útil saber qué es hoja de cultivo y qué no lo es por lo que segmentar las imágenes sería un paso importante. Con la experiencia adquirida en este trabajo de tesis se podrían modelar diferentes redes neuronales para atacar los problemas antes mencionados.

Sobre lo realizado en este trabajo de tesis, el trabajo a futuro podría ser incrementar las imágenes de higo con imágenes de diferentes cultivos de higo, en diferentes estaciones del año y en diferentes etapas de crecimiento. También podría incrementarse el número de imágenes segmentadas manualmente para tener un conjunto de datos más grande. La red neuronal convolucional se podría modificar agregando por ejemplo, más capas convolucionales y regularizadores como *batch normalization* y *dropout*. Además se puede usar *transfer learning* para extender el trabajo a otros cultivos. Hacer *transfer learning* consiste en tomar una red neuronal que tenga los pesos ajustados y que realice una tarea similar a la deseada, después se *termina de entrenar* con el conjunto de datos que se tenga disponible. Todo lo anterior con la finalidad de mejorar el desempeño de la CNN e incrementar su poder de generalización.

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, Dandelion 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] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2010.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Alexey Dosovitskiy, Jost Tobias Springenberg y Thomas Brox. Learning to generate chairs with convolutional neural networks. *CoRR*, abs/1411.5928, 2014.

[6] François Chollet. *Deep learning with Python*. Manning Publications, 2017.

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

[8] Vijay Badrinarayanan, Alex Kendall y Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[9] Vijay Badrinarayanan, Ankur Handa y Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *CoRR*, abs/1505.07293, 2015.

[10] Hyeonwoo Noh, Seunghoon Hong y Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[11] Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[12] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

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

[14] Alexander Kläser. Image annotation tool with image mask. `https://lear.inrialpes.fr/people/klaeser/software_image_annotation`, 2010.

[15] Mark Everingham, S.M. Ali Eslami, Luc Van Gool, Christopher K.I. Williams, John Winn y Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, jan 2015.

[16] Valentino Zocca, Gianmario Spacagna, Daniel Slater y Peter Roelants. *Python Deep Learning*. Packt Publishing, 2017.

[17] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676, 2012.

[18] Andres Milioto, Philipp Lottes y Cyrill Stachniss. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs. *CoRR*, abs/1709.06764, 2017.

[19] Rafael C. Gonzales y Richard E. Woods. *Digital Image Processing*. Pearson, 2008.

[20] Xavier Glorot y Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[21] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[22] Karen Simonyan y Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 09 2014.

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS
Instituto de Investigación en Ciencias Básicas y Aplicadas

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

Coordinación de Programas Educativos

Posgrado en Ciencias

DR. VICTOR BARBA LÓPEZ
COORDINADOR DEL POSGRADO EN CIENCIAS
PRESENTE

Atendiendo a la solicitud para emitir DICTAMEN sobre la revisión de la TESIS titulada **"Aprendizaje profundo para la detección de árboles en imágenes aéreas de un cultivo de higo capturadas con un dron"** que presenta el alumno **Juan Luis Torres Olivares (10009570)** para obtener el título de **Maestro en Ciencias.**

Nos permitimos informarle que nuestro voto es:

NOMBRE	DICTAMEN	FIRMA
Dr. Jesús Igor Heberto Barahona Torres IM-UNAM	Aprobado	
Dr. Jorge Hermosillo Valadez CINC-UAEM	Aprobado	
Dr. Paúl Hernández Herrera IBT-UNAM	Aprobado	
Dr. Juan Manuel Rendón Mancha CINC-UAEM	Aprobado	
Dr. Jorge Alberto Fuentes Pacheco CINC-UAEM	Aprobado	