UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

DETECCIÓN DE GRIETAS MEDIANTE DEEP LEARNING BASADO EN IMÁGENES EN CONCRETO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

MARCELO IGNACIO ORELLANA ESPINOZA

PROFESOR GUÍA:
ENRIQUE LÓPEZ DROGUETT

MIEMBROS DE LA COMISIÓN:
JUAN TAPIA FARIAS
VIVIANA MERUANE NARANJO

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: MARCELO IGNACIO ORELLANA ESPINOZA
FECHA: 2019
PROF. GUÍA: ENRIQUE LÓPEZ DROGUETT

DETECCIÓN DE GRIETAS MEDIANTE DEEP LEARNING BASADO EN IMÁGENES EN CONCRETO

En el presente informe de tesis se presenta el tema "Detección de grietas mediante Deep Learning basado en imágenes en concreto". La segmentación de grietas en concreto es una tarea importante en temas de estructuras, debido a que estas pueden provocar problemas graves sin la debida supervisión. Gran parte de trabajos previos del mismo tema tienen enfoque en detección de grietas de manera manual y no automática, por lo que se plantea como solución utilizar una red neuronal de aprendizaje automático, de manera de realizar una clasificación y segmentación de grietas con imágenes recibidas desde la industria.

Como principal motivación se tiene que la detección de grietas es una parte importante del trabajo de mantenimiento de puentes, de manera de poder evaluar su estado de salud y garantizar seguridad. Normalmente este trabajo se realiza principalmente por personal de ingeniería mediante inspección visual, un trabajo de gran dificultad y que requiere mucho tiempo, y agregando a esto la subjetividad que puede tener el personal humano [14]. Es por esto que se propone realizar esta inspección de manera automatizada, mediante la segmentación de grietas a través de una red neuronal convolucional computalizada.

Es por esto que se tiene como objetivo principal la detección automática de grietas en estructuras a través de una red neuronal, por lo que para lograr esto se tienen objetivos secundarios importantes, tales como el aprendizaje de programación en base a Deep Learning y el entrenamiento de redes de segmentación de grietas de acuerdo a las imágenes recibidas. El alcance del tema propuesto se acota a la identificación de grietas de manera automática y la comparación de modelos de segmentación para el mismo objetivo, para lo cual se propone realizar un modelo computacional en base a Deep Learning con imágenes obtenidas desde el monitoreo de grietas en concreto.

La metodología a seguir consiste en primera etapa en el entendimiento de datos disponibles, luego sigue la prueba de modelos de segmentación y la realización del modelo computacional necesario para lograr los objetivos. Luego prosigue aplicar el modelo a los datos disponibles (imágenes) y la comprobación de los datos obtenidos, en estas fases se produce un loop o seguimiento de las mismas, ya que el modelo computacional se debe entrenar continuamente de manera de alcanzar óptimos necesarios para lograr un porcentaje aceptable en la identificación y cuantificación de las grietas.

Luego, al lograr el desarrollo del modelo computacional en base Deep Learning de manera óptima y/o aceptable se procede a generar conclusiones y tasas de falla de mediciones. Por último se generarán protocolos de gravedad y límites del programa computacional, finalizando el informe requerido. Los recursos principales para realizar el tema de memoria son una GPU (Graphics Processing Unit) y la utilización del software Python y Tensor Flow.

Para mi familia, por su paciencia, apoyo y por siempre creer en mi.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

La identificación de imágenes de grietas en concreto es una tarea sumamente importante para la mantención y prevención de las estructuras y posibles fracturas, esto tomando en cuenta si las grietas son de ciertas medidas, profundidad, solo superficiales, entre otros. Las razónes que afectan a las estructuras suelen ser muy variadas, ya que pueden tener distinto origen y naturaleza. La identificación de las grietas se hace una tarea compleja ya que en ciertos casos por falta de supervisión solo se hace observable cuando ya hay daños perjudiciales para la estructura.

La principal motivación para el desarrollo de este tema es la detección de grietas en fase temprana, ya que se debe evitar el alto grado de peligro de fractura en la estructura, ya que se pueden provocar accidentes graves por peligro de desplome y la inhabilitación de la estructura, lo que provoca con esto grandes pérdidas económicas e incluso pérdidas humanas en los casos más graves.

Para evitar este problema, se tiene como principal objetivo realizar la detección automática de grietas a través de una red neuronal convolucional de manera temprana mediante la programación de un modelo computacional en base a Deep Learning, de manera de cumplir a su vez objetivos secundarios tales como:

1. Aprendizaje de programación en base a Deep Learning.

2. Programación satisfactoria y aplicación de un modelo computacional que permita la detección de grietas en concreto.

3. Cuantificación de grietas en las imágenes recibidas, así como principales conclusiones generadas a partir del modelo computacional.

El tema propuesto tiene un alcance acotado a la detección automática de grietas en concreto, pero no se descarta la expansión del msimo para otro tipo de materiales, aunque debe ser probado. Por otro lado se trata de una programación de redes de segmentación y cuantificación, por lo que también se permite, a priori, obtener la medida de la grieta identificada.

Cómo primera etapa crítica del tema a realizar se tiene el entendimiento y preparación de

los datos recibidos para la aplicación del modelo, puesto que se debe proceder con cautela de manera de no caer en malas interpretaciones de los mismos, para posteriormente no afectar la preparación de los datos y que estos no pierdan sintonía con el modelo a aplicar y los objetivos de la memoria.

Como segunda etapa crítica se tiene la realización del modelo de manera satisfactoria, ya que éste tiene que ser sometido a pruebas antes de llegar a una versión final, puesto que se debe procesar de manera clara y arrojar resultados coherentes a través del análisis de datos a la industria.

Por último la tercera etapa crítica es la aplicación del modelo, puesto que los datos generados por el mismo deben ser coherentes de manera de generar protocolos y conclusiones claras, ya que se deben evitar errores que puedan causar discordancias con la realidad en la aplicación de los protocolos.

1.1. Antecedentes generales

Ya nombramos la importancia de la detección temprana de las grietas en estructuras, por lo que es importante señalar las principales razones de la aparición de las mismas. Algunos de las causas comunes de grietas estructurales son:

- Daños como consecuencia de erosión, humedad, impactos, desgaste, entre otros.
- Fractura generada por tensiones térmicas, retracción, sobrecargas.
- Deterioro causado por condiciones climáticas, exposición a reacciones químicas, desastres naturales.
- Fallas arquitectónicas generadas por construcción deficiente, materiales inadecuados y/o falta de supervisión.

1.1.1. Identificación de grietas

La segmentación y clasificación de grietas de manera automática, permite entre otras cosas el ahorro de tiempo y personal, debido a que con un software capaz de realizar esta tarea no se necesita necesariamente un inspector o similar que se traslade a terreno para identificar las mismas grietas, por lo que adicionalmente esto permite que la inspección se estandarice, es decir, que no hayan dobles impresiones por ejemplo en el caso de inspecciones de distintos inspectores/personal especializado. Por otro lado la automatización en la identificación de grietas permite la observación de grietas con acceso restringido o en lugares de difícil acceso, habiendo también indirectamente un ahorro económico.

Capítulo 2

Marco Teórico

2.1. Inteligencia artificial y aprendizaje automático

El programa computacional para realizar el tema de memoria "Detección de grietas mediante Deep Learning basado en imágenes en concreto"se realiza empleando dos técnicas principales, redes neuronales recurrentes (RNN) y redes neuronales de segmentación.

La inteligencia artificial en como tal nace en la década de 1950, cuando unos pocos pioneros del campo de la informática empieza a preguntarse si las computadoras podrían hacerse para "pensar", una pregunta cuyas ramificaciones aún estamos explorando hoy. El campo de la inteligencia artificial se podría definir de manera concisa como .[el] esfuerzo por automatizar las tareas intelectuales que normalmente realizan los humanos". Como tal, la inteligencia artificial es un campo general que abarca el aprendizaje automático y aprendizaje profundo, pero eso también incluye muchos más enfoques que no involucran precisamente el aprendizaje. Los primeros programas de ajedrez, por ejemplo, solo involucraban reglas codificadas elaboradas por programadores, y no calificaron como aprendizaje automático. Durante bastante tiempo, muchos de los expertos creían que la inteligencia artificial a nivel humano podría lograrse haciendo que los programadores elaboraran un conjunto suficientemente grande de reglas explícitas para manipular el conocimiento deseado. Este enfoque se conoce como inteligencia artificial simbólica (IA simbólica), y fue el paradigma dominante en IA desde la década de 1950 hasta finales de la década de 1980. La IA simbólica alcanzó su máxima popularidad durante la década de 1980, pero aunque la IA simbólica es adecuada para resolver problemas lógicos bien definidos (como jugar ajedrez), resultó no ser suficiente para resolver problemas más complejos y difusos, como por ejemplo la clasificación de imágenes, reconocimiento de voz o traducción de idiomas. Es por esto que ante la necesidad surge un nuevo enfoque para tomar el lugar simbólico de la IA: el aprendizaje automático ([13] Pág. 4)

La inteligencia artificial y el aprendizaje automático junto a la evolución computacional actualmente permiten la solución de problemas en diferentes disciplinas mediante redes neuronales, que sin este recurso se efectuarían en tiempos muchísimo mayores. Este recurso pertenece en su mayoría al área de la informática o computación, pero también está ligado a la toma de decisiones y simulación de problemas en muchas disciplinas, sólo por mencionar problemas tales como diseño de maquinaria, vehículos autónomos, compras en línea,

entre otros, pueden ser solucionados mediante la programación y aprendizaje de inteligencia artificial.

2.2. Machine Learning y Deep Learning (Aprendizaje de máquina y aprendizaje profundo)

Machine Learning se define como la ciencia de programar sistemas computacionales para que puedan aprender de manera autónoma (aprendizaje automático) desde información entregada. Citando a Arthur Samuel (1959), "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed", es decir, el campo del estudio en el cual se da a los computadores la habilidad de aprender sin ser explicitamente programados para esto ([5] Pág 4).

Machine Learning en general, puede ser utilizado para automatizar tareas que sin este recurso requerirían una constante supervisión y actualización de parte del programador, haciendo que la tarea sea una actualización y supervisión continuas. Este recurso en cambio hace que ese tipo de problemas sea solucionado mediante el aprendizaje autónomo del programa mediante data o información que va llegando al mismo, un ejemplo de funcionamiento del sistema puede ser observado en la siguiente figura.

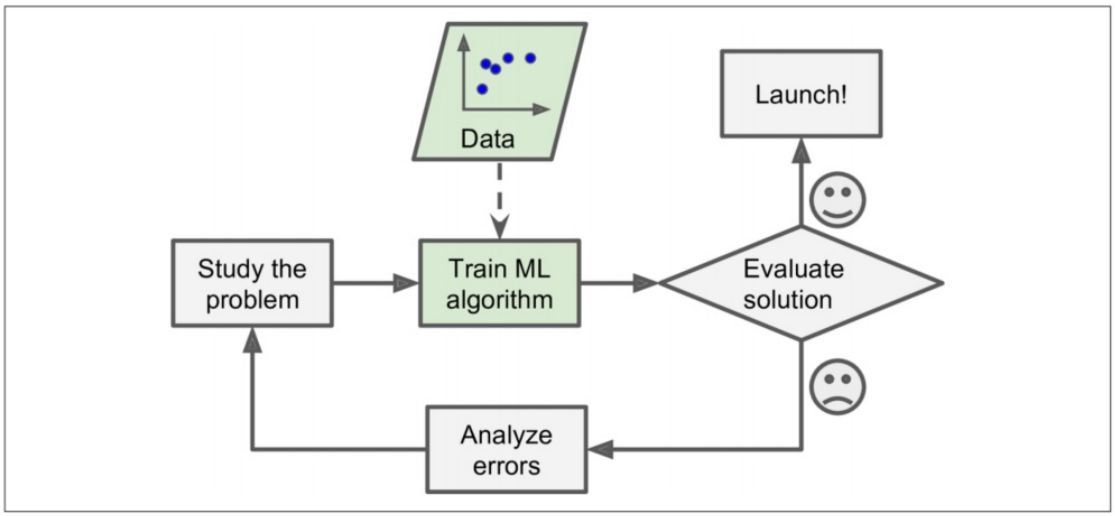

Figura 2.1: Concepto de Machine Learning - [5] Aurélien Géron (Hands-on Machine Learning with Scikit-Learn and TensorFlow) Figure 1-2.Machine Learning approach, page 5, 2017.

Por otro lado, Machine Learning (o ML en su abreviatura), puede ser utilizado para profundizar en grandes cantidades de data o información, de manera de ayudar a descubrir patrones que no son fácilmente observables de manera manual, tarea también llamada "data mining"([5] Pág 6). En la siguiente imagen se puede observar un esquema de cómo el concepto puede ser llevado también al aprendizaje de nosotros mismos hacia el problema en particular.

El aprendizaje profundo o Deep Learning es un subcampo específico del aprendizaje automático o ML, una nueva versión de las representaciones de aprendizaje a partir de datos que pone énfasis en el aprendizaje de capas sucesivas cada vez más profundas. Otros nombres

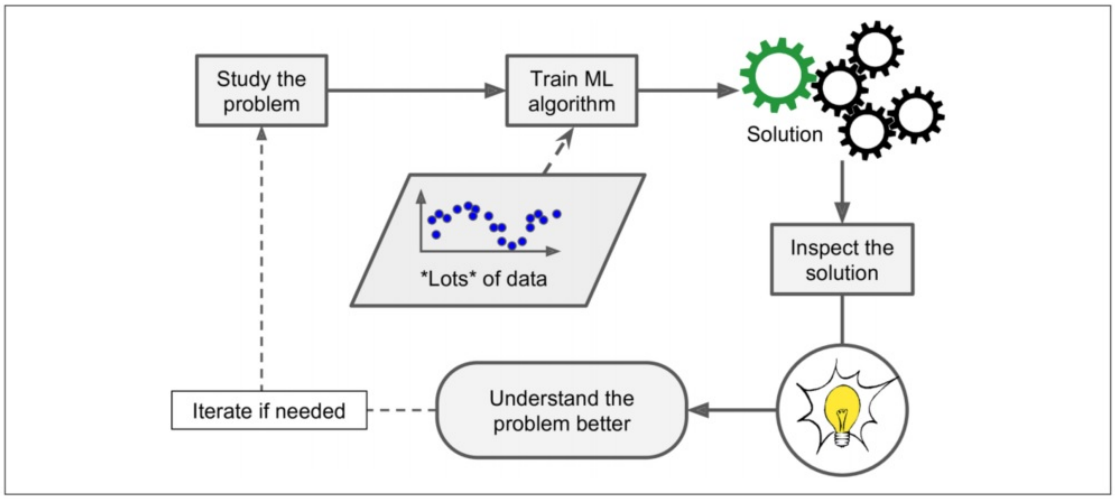

Figura 2.2: Concepto de aprendizaje desde Machine Learning - [5] Aurélien Géron (Hands-on Machine Learning with Scikit-Learn and TensorFlow) Figure 1-4. Machine Learning can help humans learn, page 7, 2017.

apropiados para el campo podrían haber sido aprendizaje de representaciones en capas y aprendizaje de representaciones jerárquicas. El aprendizaje en Deep Learning a menudo implica decenas o incluso cientos de capas sucesivas de representaciones.([5] Pág 8). En general, Deep Learning es el nombre que se le da a arquitecturas (o forma específica de conexión) de redes neuronales y algoritmos con un gran cantidad de capas no lineales, que usa la inteligencia artificial para aprender utilizando estas arquitecturas, haciendo posible lograr el aprendizaje de patrones.

Aunque el aprendizaje profundo es un subcampo bastante antiguo del aprendizaje automático, solo se popularizo a principios de la década de 2010. En los pocos años transcurridos desde entonces, ha revolucionado el campo con grandes resultados en problemas tales como la percepción de la vista y el oído, o incluso problemas que involucran habilidades que parecen naturales e intuitivas para los humanos, pero que se veían muy lejanas de alcanzar para las máquinas. En particular, el aprendizaje profundo ha logrado avances tales como:

- Clasificación de imágenes a nivel casi humano.
- Reconocimiento de voz a nivel casi humano.
- Transcripción de escritura a mano de nivel casi humano.
- Traducción automática mejorada.
- Conversión de texto a voz mejorada.
- Conducción autónoma casi humana.
- Resultados de búsqueda mejorados en la web.
- Capacidad para responder preguntas en lenguaje natural.

- Entre otros ([5] Pág 11).

2.3. Redes neuronales o Perceptrones multicapa

Una red neuronal artificial o perceptrones multicapa puede ser entendida de mejor manera desde comparar la unidad básica de la misma como una neurona en un cerebro biológico ([5] Pág. 254), donde se traspasa información junto a su conexión en redes y la forma en que se adapta el mismo para el aprendizaje. Dependiendo de la necesidad que se presente, una fila de neuronas se llama capa y una red puede tener múltiples capas (caso Deep Learning). Básicamente se compone de una capa de entrada, una o más capas llamadas capas ocultas, y una capa final llamada capa de salida. Cada capa, excepto la capa de salida, incluye una neurona sesgada completamente conectada a la siguiente capa. Cuando una red tiene dos o más capas ocultas, se hace llamar red neuronal profunda ([5] Pág. 261). La gran ventaja de las redes neuronales es su capacidad de aprendizaje o de reconocimiento de representaciones desde datos de entrenamiento, y como realizar relaciones con la variable de salida o objetivo que se desea obtener. Esta capacidad de predictiva proviene de la multicapa de las redes neuronales, ya que ésta puede aprender a representar características a dimensiones distintas y combinarlas en características de dimensiones o resoluciones superiores. Por ejemplo ir desde líneas a representar formas con esas mismas líneas.

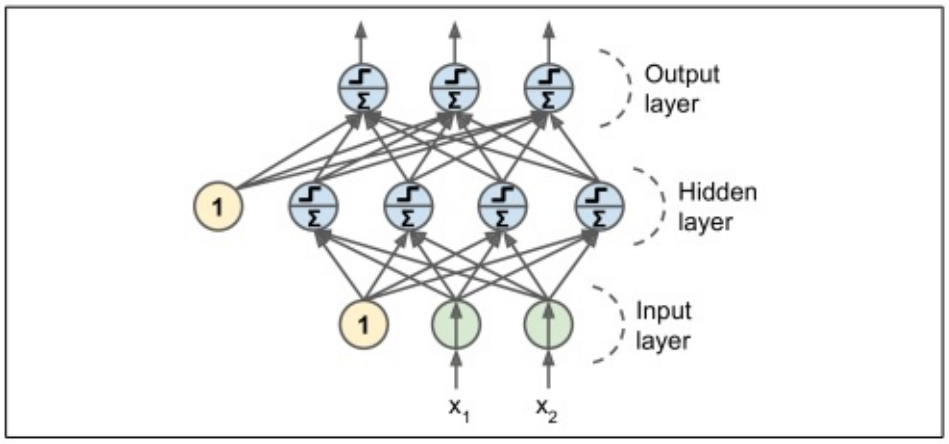

Figura 2.3: Ejemplo de perceptron multicapa - [5] Aurélien Géron. Hands-on Machine Learning with Scikit-Learn and TensorFlow, concepts, tools and techniques to build intelligent systems, 2017 (Fig. 10-7 Multilayer Perceptron)

En resumen, entrenar una red neuronal gira en torno a los siguientes conceptos:

- Capas combinadas en una red (o modelo de red neuronal).
- Datos de entrada y objetivo de la red.
- Función de pérdida, la cual define la señal de retroalimentación utilizada para el aprendizaje de la red.
- Optimizador, que determina como se va modificando el aprendizaje de la red.

La red, compuesta de capas, asigna los datos de entrada a las predicciones. La función de pérdida luego compara estas predicciones con los objetivos, produciendo un valor de pérdida, la cual es una medida de qué tan bien las predicciones de la red coinciden con lo esperado. El optimizador usa este valor de pérdida para actualizar los pesos de la red ([13] Pág. 58).

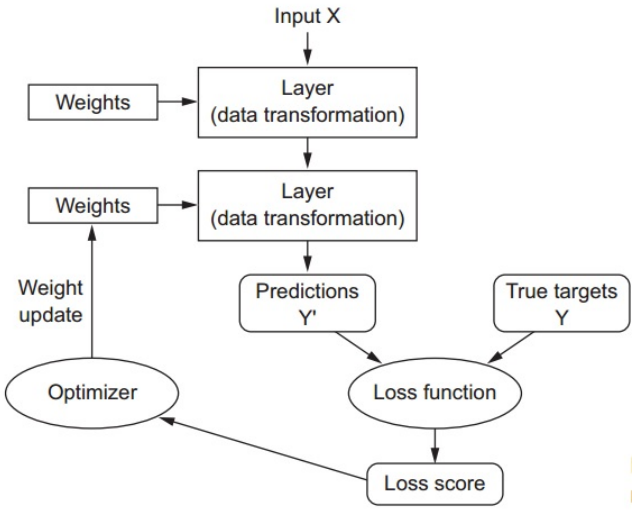

Figura 2.4: Relación entre la red neuronal, las capas, función de pérdida y optimizador - [13] Francois Chollet (Deep Learning with Python) Fig 3.1. Relationship between network, layers, loss function, and optimizer.

2.3.1. Neuronas artificiales

Una neurona artificial es la unidad básica de las redes neuronales, en otras palabras es una unidad computacional la cual pondera señales de entrada y producen señales de salida, estas últimas mediante funciones de activación, formando redes neuronales o modelos que varían de acuerdo a la experiencia y entrenamiento de la misma red con datos de entrada, utilizando procedimientos iterativos.

Las redes neuronales convolucionales (siglas en inglés CCN), son redes útiles principalmente para el reconocimiento de imágenes, donde las neuronas de las redes tienen un comportamiento muy parecido a las neuronas ubicadas en la corteza visual primera de un cerebro biológico. La red va desde lo más general a lo más particular, donde en el último tramo se encuentran neuronas que tienen la principal tarea de clasificar.

Las redes neuronales recurrentes, también conocidas como redes dinámicas (siglas en inglés RNN), son redes útiles principalmente para analizar secuencias de tamaño variable, reproducción de secuencias, reconocimiento de secuencias y asociación temporal. Las RNNs se caracterizan por ser sistemas dinámicos, donde lo presente en el paso de una entrada, depende del paso anterior o del paso futuro, según sea el caso. Estas redes tienen la capacidad de realizar tareas tales como predicción no lineal, modelación de sistemas dinámicos, tratamiento de secuencias, entre otros.

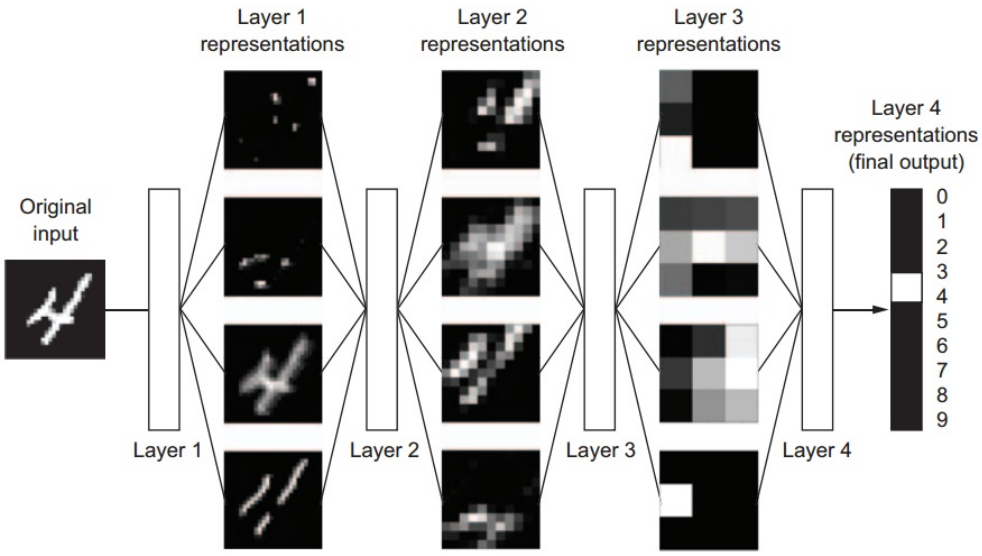

Figura 2.5: Funcionamiento de ejemplo de una red neuronal para el reconocimiento de imágenes) - [13] Francois Chollet (Deep Learning with Python) Fig 1.6. Deep representations learned by a digit-classification model.

2.3.2. Pesos Neuronales o sinápticos

Cada neurona de las que se explica anteriormente, tiene un sesgo o valor que puede ser considerado su valor de entrada, el cual siempre tiene el valor 1.0 y el que como puede inferirse, debe ser ponderado. Por ejemplo, digamos que tenemos una neurona con dos entradas, es decir, tiene dos entradas y un sesgo, por lo que necesita 3 pesos neuronales o sinápticos. Usualmente a los pesos se les aplica valores aleatorios relativamente bajos (rango de 0 a 0,3) aunque esto puede variar dependiendo de la complejidad de la red. A medida que los valores de los pesos van aumentando, también se va obteniendo una mayor complejidad del modelo, por lo que hay que tener en cuenta esto para la estabilidad del mismo.

2.3.3. Función de activación o Función de transferencia

Las neuronas artificiales que sirven de entradas ponderadas (mencionadas anteriormente) se suman, para luego pasar a través de la función de activación (o transferencia), que básicamente modifican o activan la neurona que pase realizando una asignación simple a la misma, dependiendo del valor que tenga ponderada la misma. Sin una función de activación la capa solo podría aprender transformaciones lineales (transformaciones afines) de datos de entrada. Para obtener acceso a un espacio de hipótesis mucho más rico que se beneficiaría de representaciones profundas, necesita una no linealidad o función de activación. Relu es la función de activación más popular en el aprendizaje profundo, pero hay muchos otros candidatos, todos los cuales vienen con nombres similares, prelu, elu, etc ([13] Pág. 72).

En otras palabras, la función de activación básicamente se encarga de devolver una salida a partir de un valor de entrada, siendo normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1). Usualmente las funciones de activación son no lineales debido a que permite que la red pueda combinar entradas de maneras más complejas,

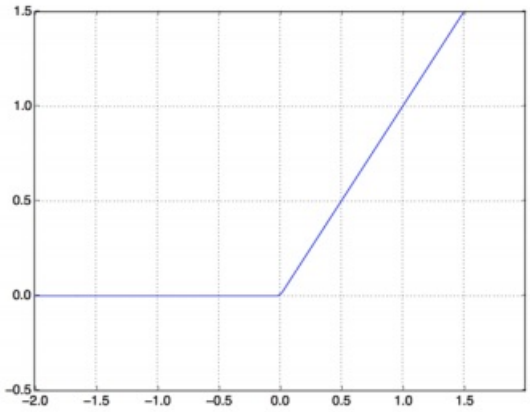

Figura 2.6: Función de activación relu (rectified linear function) - [13] Francois Chollet (Deep Learning with Python) Fig 3.4. rectified linear function.

proporcionando mayor capacidad en las funciones a modelar. Por otro lado además de relu hay otras funciones de activación usadas tales como la función logística (que emite un valor entre 0 y 1 (distribución en forma de s)), la función tangente hiperbólica ((Tahn) que emite la misma distribución en el rango de -1 a +1), o la función sigmoide (que transforma los valores introducidos a una escala (0,1), donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0).

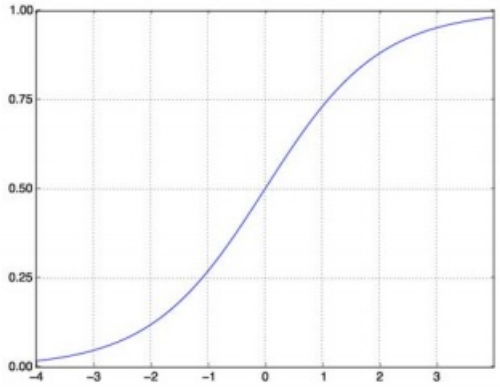

Figura 2.7: Función de activación sigmoud - [13] Francois Chollet (Deep Learning with Python) Fig 3.5. The sigmoud function.

2.3.4. Entrada, capas ocultas y salida

Las entradas de la red neuronal se refiere a las capas visibles de la misma en el inicio de la red, de manera de que se puede decir que es la parte expuesta. Cabe decir que esta no es una neurona como tal, ya que pasa el valor de su entrada a la capa inmediatamente después.

Las capas ocultas no están expuestas en la red neuronal, donde lo más simple es incluir solo una neurona en esta capa que emita directamente un valor, y que se puede complejizar en múltiples capas ocultas como lo es el Deep Learning.

La salida se puede decir que es la última capa oculta, ya que su función es emitir el valor o vector de valores necesario desde las capas anteriores necesario para el problema u objetivo que se busca, donde la forma de la función de activación de la misma (necesaria o no), está ligada necesariamente al tipo de problema a modelar.

2.3.5. Épocas y tamaño de lote

Las épocas (o epochs en inglés) es uno de los hiperparámetros importantes a considerar en Deep Learning, las épocas son básicamente cuando el dataset completo es pasado a través de la red neuronal, es decir, una época es cuando el dataset completo es pasado a través de la red neuronal una vez.

El temaño de lote (o batch size en inglés) es el número total de ejemplos de entrenamiento que se pasa en cada lote, es decir, básicamente la división de nuestro dataset o set de datos en distintos tamaños de datos, de manera de poder procesar de mejor manera los mismos a través de nuestra red neuronal.

También es importante agregar la definición de iteración, puesto que las iteraciones son la cantidad o número de lotes necesario para completar una época, es por esto que explicado de otra manera, el número de lotes (o batches) es igual al número de iteraciones para una época.

2.4. Métricas de evaluación

2.4.1. Sobre ajuste y Sub ajuste

El concepto de sobre ajuste (Overfitting en inglés) se refiere principalmente a la problemática que ocurre cuando el modelo se ajusta demasiado a los datos que utiliza para el entrenamiento, provocando que al ingresar nuevos datos al modelo la red no los identifique como tal, ya que no son estrictamente los valores de los datos principales. En otras palabras se puede decir que el sobre ajuste es la sobre generalización de los datos, algo que los humanos hacemos con demasiada frecuencia, y desafortunadamente las máquinas también pueden caer en la misma trampa si no tenemos cuidado. Los modelos complejos, como las redes neuronales profundas, pueden detectar patrones sutiles en los datos, pero si el conjunto de entrenamiento es ruidoso, o si es demasiado pequeño (lo que introduce ruido de muestreo), entonces es probable que el modelo detecte patrones en el mismo ruido. ([5] Pág. 26) Las posibles soluciones al sobre ajuste son: • Simplificar el modelo seleccionando uno con menos parámetros. • Reunir más datos de entrenamiento. • Reducir el ruido en los datos de entrenamiento (por ejemplo, corregir errores de datos y eliminar valores atípicos) ([5] Pág. 27).

El concepto de sub ajuste (Underfitting en inglés) como se puede inferir, es lo contrario al sobre ajuste. Sub ajuste se refiere principalmente a la problemática que ocurre con el modelo no está correctamente ajustado, es decir cuando hay una falta de datos para generalizar el concepto que se requiere presentar al modelo o cuando el modelo es demasiado simple para aprender la estructura de los datos, haciendo que éste no pueda reconocer correctamente el mismo. Esto puede ocurrir principalmente por la falta de datos representativos o simplemente

un bajo número de ellos, entre otros. Las principales opciones para solucionar este problema son: • Seleccionar un modelo más potente, con más parámetros. • Modificar el algoritmo mediante mejores funciones de aprendizaje. • Reducir las restricciones en el modelo. ([5] Pág. 28).

2.4.2. Precisión (Acuraccy) y Pérdida (loss)

El concepto de precisión (accuracy) corresponde a los datos que son clasificados correctamente por el modelo, es importante tener en cuenta la precisión para medir la confiabilidad del modelo, de manera de tener un porcentaje del mismo. En general, el término de precisión en modelos de clasificación se refiere al porcentaje de éxito que la clasificación tiene de acuerdo al modelo a los datos proporcionados. Hay que tener cierta precaución con este término, puesto que en modelos sobre ajustados (overfitting) puede tener demasiado rápido a un valor alto, luego puede corresponder a un sobre ajuste de los datos de entrenamiento.

La pérdida (loss) corresponde básicamente al error en el conjunto de datos, es decir, la pérdida de los mismos, dependiendo si son los datos de entrenamiento o validación. La pérdida de entrenamiento básicamente corresponde a el error en el conjunto de datos de entrenamiento, en tanto la pérdida de validación corresponde básicamente al error después de ejecutar el conjunto de datos de validación a través de la red utilizada o modelo.

2.4.3. Filtros

Los filtros (Filters en inglés) corresponden básicamente al número de convoluciones al cual será expuesta la imagen a tratar, dependiendo del modelo y el objetivo a realizar.

2.4.4. Intersección sobre unión (IOU)

Para evaluar el comportamiento o rendimiento de la detección y segmentación de grietas se utilizará como parámetro el IOU, es decir la intersección sobre unión (o Intersection over union en inglés, desde ahí sus siglas). Este parámetro se adapta bien a las necesidades que hay, por ejemplo en la detección de pixel a pixel dentro de imagenes predichas por un modelo en términos de detección de grietas [1]. A las imágenes predichas por el modelo también se les puede nombrar como cajas delimitadoras, puesto que se generan imágenes que básicamente predicen por cual parte de la imagen está la grieta, delimitándola de color blanco (binario 0) y dejando de color negro (binario 1) la parte de la imagen sin grieta.

Para utilizar de manera correcta este parámetro, se necesitan dos cosas fundamentales:

- Las imágenes etiquetadas (o cajas delimitadoras) de manera manual de las grietas que se quieran predecir, es decir el conjunto de imágenes de prueba que especifican directamente donde está el objeto (o en este caso la grieta).

- Las imágenes (o cajas delimitadoras) predichas por el modelo utilizado [1].

Como podemos ver en la imagen siguiente, los valores de IOU van desde 0 a 1 (o 0 a 100 por ciento), donde a mayor coincidencia de imágenes (o cajas delimitadoras) mayor es el valor de IOU. Un valor de 1 o 100 por ciento es casi imposible y no es lo que se busca,

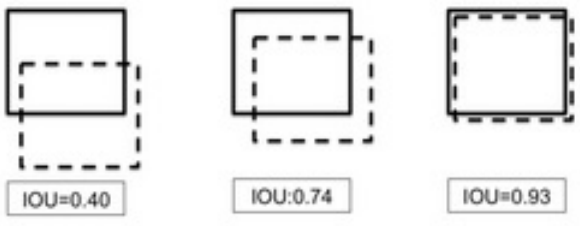

Figura 2.8: Ejemplo de parámetro IOU - [1] Juan Tapia, Enrique Lopez, Claudio Yanez, and Ruben Boroschek (Automatic Crack Detection and Quantification Based on Concrete Bridges Images.) Figure 3: Examples of IOU metric values, in order to understand the influence and relevance of this metric on semantic segmentation, page 5, 2019.

pero si tener en cuenta que el valor de IOU sea lo más grande posible y por lo tanto que las imágenes a comparar coincidan en un traslape lo más cercano posible.

2.5. Segmentación Semántica

La segmentación semántica en términos formales se define como "La segmentación semántica es la tarea de asignar una clase a cada píxel en una imagen dada"[9]. Específicamente corresponde a un algoritmo de Deep Learning que asocia o clasifica cada pixel de la imagen a una categoría, por ejemplo clasificando un objeto específico en una imagen donde están presentes varios objetos (por ejemplo un automóvil en una imagen de una calle en particular, peatones en una acera, etcétera). La segmentación semántica tiene variadas aplicaciones en el área de la inteligencia artificial, como por ejemplo la generación de inspecciones industriales, conducción autónoma, entre otros [9].

La segmentación semántica en este informe de tesis en particular se aplica al momento de identificar la presencia o no de grietas en el concreto analizado, esto se realiza con la misma técnica que se realiza en la imagen anterior, identificando pixel a pixel en las imágenes analizadas y clasificando las áreas correspondientes a grietas como grietas, y las áreas correspondientes a no grietas como no grietas, en este caso en particular, el área de grieta en color blanco y el área de no grieta en color negro.

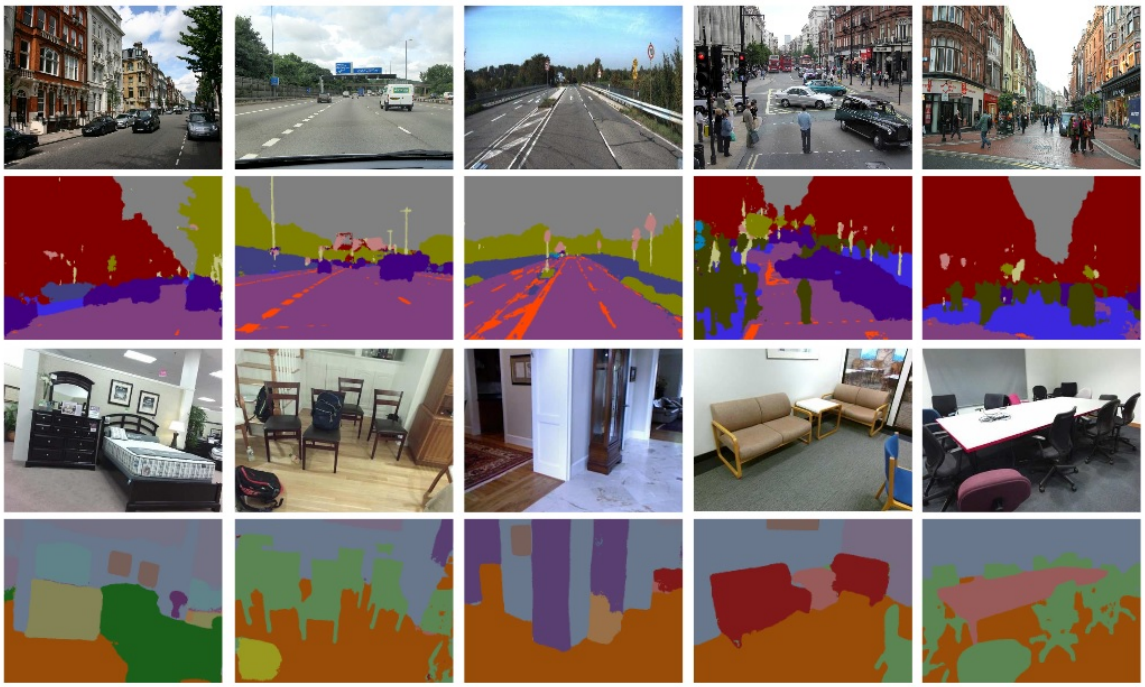

Figura 2.9: Ejemplo de segmentación semántica. - [3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla (SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation) Figure 1. SegNet predictions on road scenes and indoor scenes, page 2, 2016.

2.5.1. Red neuronal VGG

La red neuronal VGG es una arquitectura CNN (de red convolucional) de gran alcance, es utilizada en el reconocimiento de objetos y construida por el grupo de geometría visual (VGG Visual Geometry Group) de la Universidad de Oxford. Uno de los principales motivos de su popularidad es que está totalmente disponible para su uso, además de asegurar el primer y segundo puesto en tareas de clasificación y localización en ImageNet challenge 2014 (una competencia anual conocida como .[el] desafío de reconocimiento visual a gran escala ImageNet (ILSVRC)"[8]).

Para entender mejor la red es claro afirmar que ésta es una red CNN, de la cual casi todas son similares entre sí, un conjunto de convolución, capas de agrupación al principio y capas completamente conectadas al final. La diferencia en este sentido es que la red neuronal VGG contribuye a mostrar con eficacia la mejora en las tareas de clasificación y localización utilizando mayor profundidad (esto a pesar de utilizar campos receptivos pequeños en las capas). Existen dos variantes, VGG16 y VGG19, donde 16 y 19 corresponden al número de capas respectivamente.

Esta red en particular, entre muchas opciones, adopta la opción más simple, ya que sólo se utilizan convoluciones 3x3 y agrupaciones 2x2 en toda la red. Además de esto, esta red también muestra que la profundidad en las redes neuronales es realmente importante, puesto que se logra observar que las redes más profundas en general dan mejores resultados, unas de las razones también de la popularidad del aprendizaje profundo (Deep Learning).

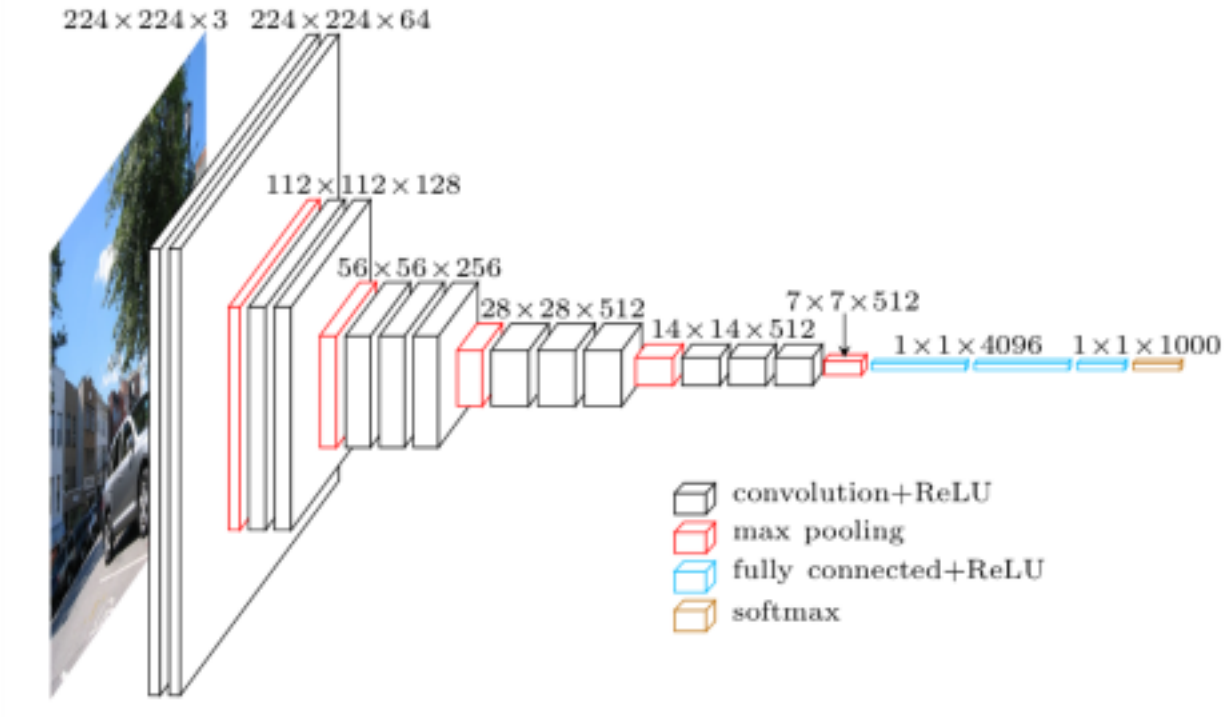

Figura 2.10: Arquitectura de red VGG16. - [8] Muneeb ul Hassan, (VGG16 – Convolutional Network for Classification and Detection) Fig. VGG16 - Noviembre 2018

2.5.2. DeepLabV3+

DeepLab es un modelo de segmentación semántica diseñado y abierto por la empresa Google en 2016. Desde ahí es donde se han desarrollado mejoras al modelo, entre ellas DeppLabV2, DeepLabV3 y el modelo principal del tema de memoria DeepLabV3+.

El modelo DeepLabV3+ ha sido ampliamente utilizado en segmentación semántica, principal tarea que se realiza en este tema de memoria, puesto que ésta misma corresponde al desarrollo de la clasificación de las imágenes pixel a pixel entre grietas y no grietas. Es por eso que este modelo es uno de los elegidos para realizar los objetivos requeridos.

La estructura del modelo DeepLab se compone básicamente de dos fases:

- Fase de codificación: En esta fase el objetivo principal es extraer la información esencial de la imagen recibida, esto utilizando una red neuronal convolucional pre-entrenada. La red neuronal corresponde a una CNN debido a que ésta facilita la clasificación de imágenes, las capas convolucionales buscan diferentes características en la imagen y pasan la información a capas subsiguientes, por lo que dado esto, las tareas de segmentación (que igualmente comprenden características e información esencial de la imagen) igualmente se facilitan, debido a que para estas se debe obtener información de la localización de los objetos en la imagen.

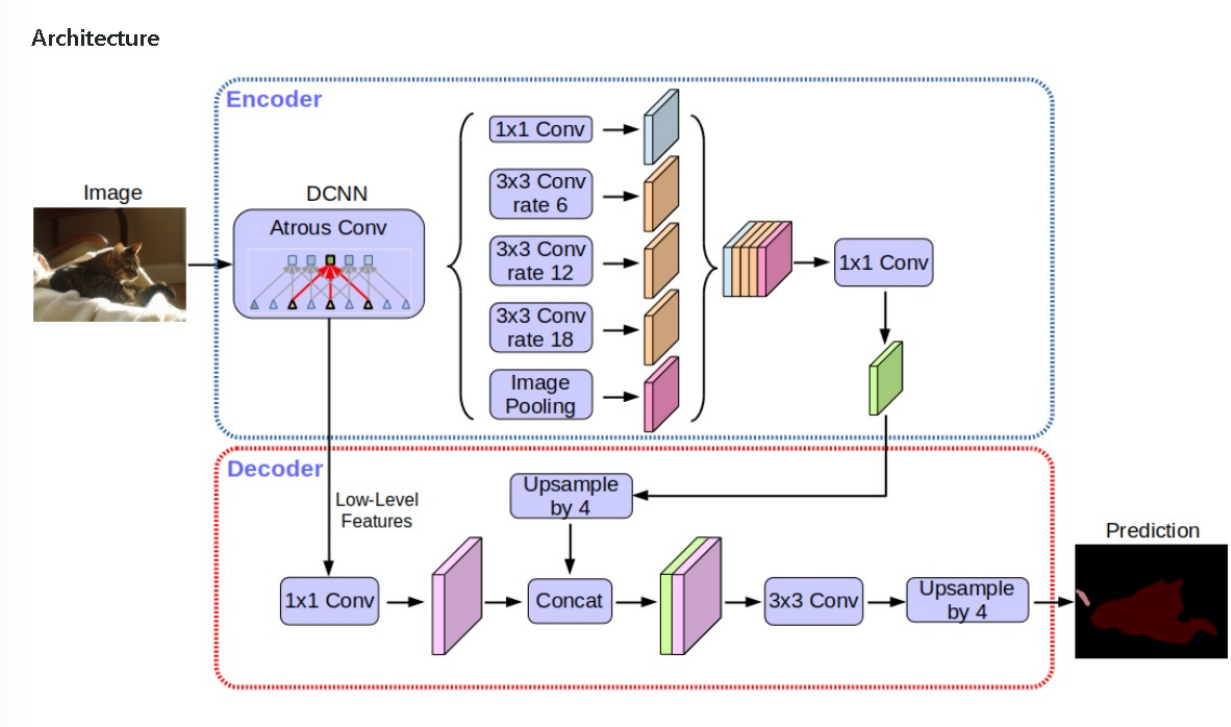

Figura 2.11: Ejemplo de funcionamiento de modelo DeepLabv3+ - [10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam (Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation) Page 4, Fig. 2: Our proposed DeepLabv3+ extends DeepLabv3 by employing a encoder-decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decodermodule refines the segmentation results along object boundaries, Agosto 2018.

- Fase de decodificación: Luego de la fase de codificación se utiliza la información extraída para construir (o reconstruir) la salida de la misma información de la imagen en dimensiones con parámetros propuestos [9].

Para entender de mejor manera la arquitectura DeepLab se puede decir que ésta es una combinación de dos redes neuronales ampliamente utilizadas, redes de agrupación espacial piramidal (a) y redes codificador-decodificador (encoder-decoder) (b). Para el caso del modelo mejorado DeepLabv3+ (c), se modifica desde el modelo DeepLabv3, dado que contiene información semántica enriquecida en el módulo codificador, mientras que el módulo decodificador a pesar de ser simple resulta realmente efectivo, puesto que recupera los límites de objetos detallados, es decir, posee mayor resolución en las imágenes de salida. Las redes de agrupación espacial piramidal son capaces de codificar información multi-escalar, es decir de distintas dimensiones, esto dado que se utiliza un campo de visión de los datos de manera efectiva, incluyendo desde la dimensión más grande hasta la dimensión más pequeña. En general, este tipo de red utiliza versiones paralelas de la misma información de red para entrenarla con diferentes tipos de entradas o inputs hacia la salida, de manera de combinarlas en el paso posterior [9].

Obviamente no toda la información que entre en estas particiones es importante, por lo

que aquí es donde entra el modelo codificador-decodificador (encoder-decoder), que actúa de filtrador en la entrada de manera de que la información recibida sea suficiente en cada una de las mismas para representar la información suficiente y requerida [9] .

El modelo DeeplabV3+ entonces básicamente corresponde a una estructura de agrupación espacial piramidal combinada con una codificador-decodificador, donde se utiliza en redes neuronales profundas para tareas de segmentación semántica. Este primer tipo de redes son capaces de codificar información contextual a múltiples escalas, de manera de sondear las funciones entrantes con filtros u operaciones de agrupación a múltiples velocidades y múltiples campos de visión efectivos. Las últimas redes en cambio pueden capturar límites de objetos más nítidos recuperando gradualmente la información espacial. Este tipo de redes son usadas en redes neurales profundas (Deep Learning) especialmente en tareas de segmentación semántica.

Para entenderlo mejor, hablando más específicamente el modelo DeepLabv3+ (o DeepLabV3plus) extiende el modelo DeepLabv3 al agregar un módulo decodificador simple pero efectivo, de manera de refinar los resultados de la segmentación, especialmente a lo largo de los límites de los objetos.

La estructura encoder-decoder es usada en redes profundas para tareas de segmentación semántica, específicamente el modelo DeepLabV3+ usado aquí se extiende desde el modelo DeepLabV3 simplemente añadiendo un módulo decodificador efectivo, de manera de refinar los resultados de segmentación a lo largo de los límites de los objetivos, resultando en un modelo más rápido y sólido.

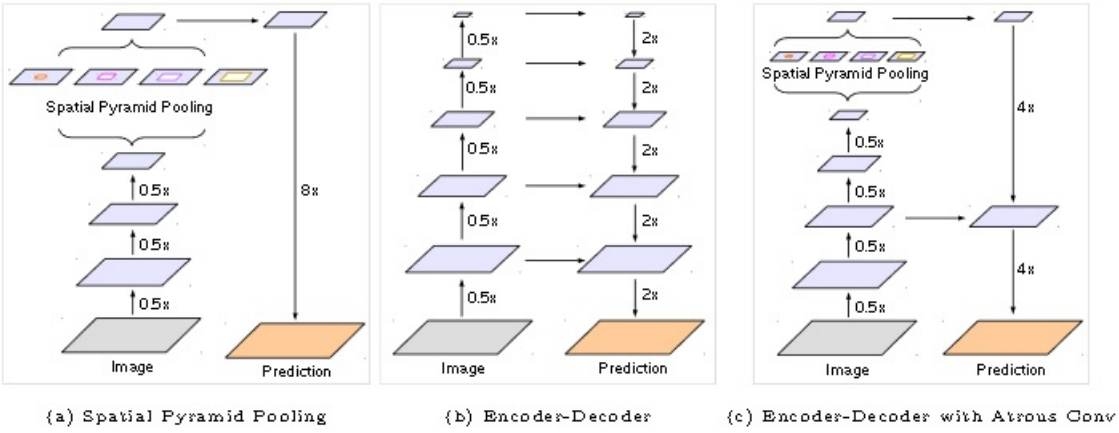

Figura 2.12: Comparación de estructuras - (a) Spatial pyramid Pooling - (b) Encoder-Decoder - (c) Encoder-Decoder with Atrous Convolutions (DeepLabv3+) - - [10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam (Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation) Page 4, Fig. 1: We improve DeepLabv3, which employs the spatial pyramid pooling module (a),with the encoder-decoder structure (b). The proposed model, DeepLabv3+, containsrich semantic information from the encoder module, while the detailed object bound-aries are recovered by the simple yet effective decoder module. The encoder moduleallows us to extract features at an arbitrary resolution by applying atrous convolution, Agosto 2018.

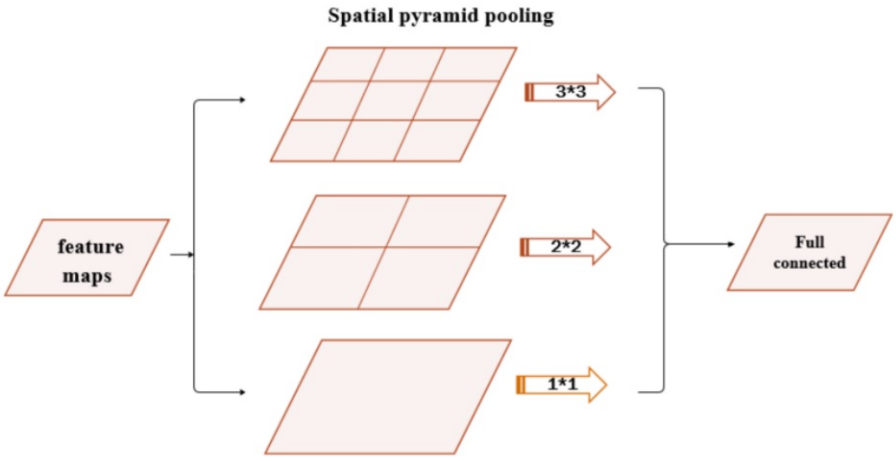

Figura 2.13: Ejemplo de procesamiento en agrupación espacial piramidal - [9] Saurabh Pal, Semantic Segmentation: (Introduction to the Deep Learning Technique Behind Google Pixels Camera!) Fig. Spatial pyramid pooling, Febrero 2019.

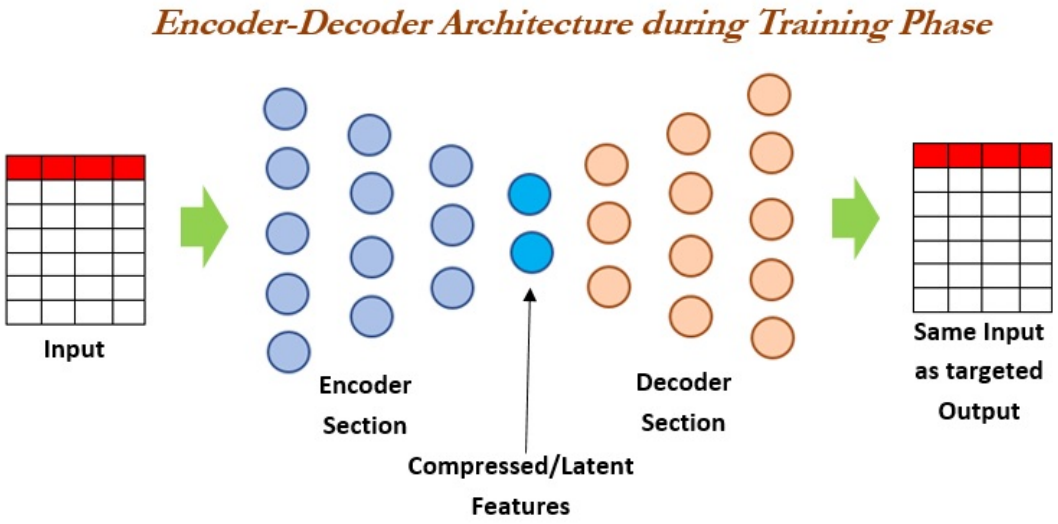

Figura 2.14: Ejemplo de la arquitectura de modelo encoder-decoder durante la fase de entrenamiento - [9] Saurabh Pal, Semantic Segmentation: (Introduction to the Deep Learning Technique Behind Google Pixels Camera!) Fig. Encoder decoder arqhitecture during training phase, Febrero 2019.

2.5.3. Encoder-Decoder

Este modelo corresponde, igualmente que el anterior, a una estructura de codificador-decodificador, éste por su parte en este caso basado en SegNet ([3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. Senior Member, IEEE: Octubre 2016).

Los beneficios clave y por los que son probados para este problema, es el enfoque son la capacidad de entrenar un único modelo de extremo a extremo directamente en las datos de origen y destino y la capacidad de manejar secuencias de entrada y salida de la misma data

o en este caso imágenes. Es por esto que se considera como uno de los modelos a comparar.

Esta red utiliza un codificador-decodificador de estilo VGG, donde el re muestreo en el decodificador se realiza mediante convoluciones transpuestas. En la arquitectura SegNet no hay capas completamente conectadas, por lo tanto sólo es convolucional, es decir se logra que el codificador en esta red sea más pequeño y fácil de entrenar que otras arquitecturas. A continuación se presenta una ilustración de la misma:

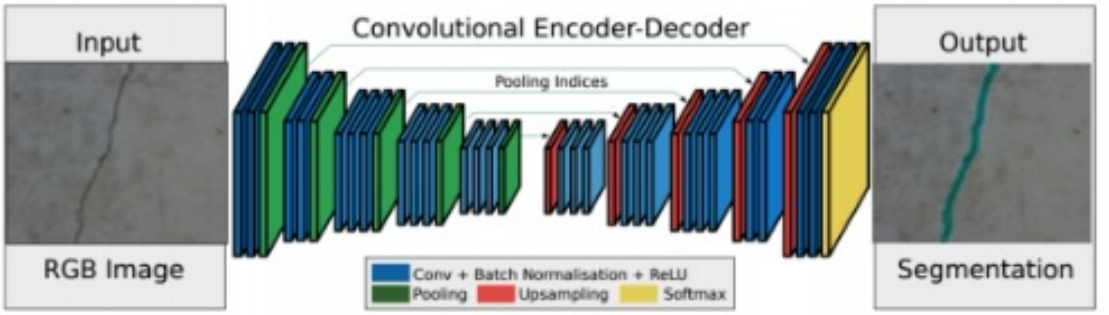

Figura 2.15: Ilustración de arquitectura encoder-decoder - [1] Juan Tapia, Enrique Lopez, Claudio Yanez, and Ruben Boroschek (Automatic Crack Detection and Quantification Based on Concrete Bridges Images.) Figure 11: The proposed Encoder-Decoder model. There are no fully connected layers, and hece it is only convolutional. The decoder upsamples its input using the transferred pool indices from its encoder to produce sparse feature maps, page 8, 2019.

- Encoder

En el codificador (encoder), se realizan convoluciones y la agrupación máxima (máx pooling). Por otro lado hay 13 capas convolucionales de VGG-16. (Las capas completamente conectadas originales se descartan) y 3 totalmente conectadas. Al realizar una agrupación máxima de 2 x 2 (máx pooling), los índices de agrupación máxima correspondientes se almacenan (ubicaciones). Es importante mencionar que los pesos del codificador están preentrenados desde la base de datos de objetos de clasificación en ImageNet.

- Decoder

En el decodificador (decoder), se realizan muestreos convolucionales y ascendentes. Luego del decodificador se encuentra un clasificador softmax para cada píxel. Durante el muestreo ascendente, los índices de agrupación máxima (índices del max pooling) en la capa del codificador correspondiente se recuperan, para luego obtener una muestra superior, en la imagen anterior podemos observar como es este proceso. Finalmente, se usa un clasificador softmax clase K, de manera de predecir la clase para cada píxel.

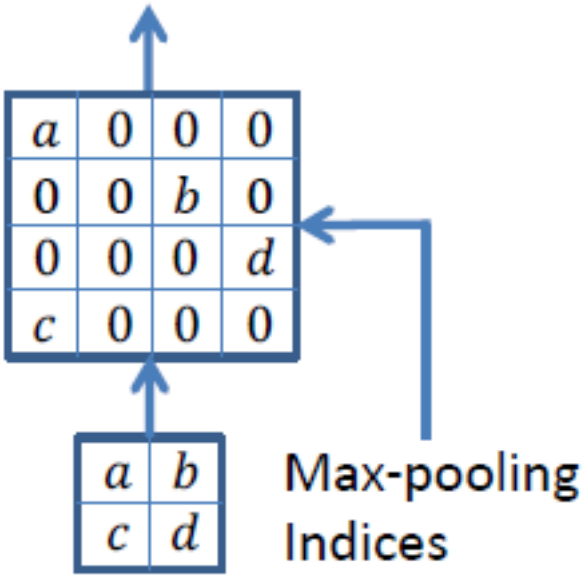

Figura 2.16: Ejemplo de la arquitectura de modelo encoder-decoder durante la fase de entrenamiento - [3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla (SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation) Fig. 3 An illustration of SegNet decoder, Octubre 2016.

2.5.4. Encoder-Decoder with Skip

El modelo Decodificador-codificador con salto (Encoder-decoder with skip) corresponde, al igual que el anterior, a una estructura codificador-decodificador, pero con la diferencia en que ésta tiene conexiones de salto basadas en SegNet. El modelo corresponde al mismo paper del modelo anterior, pero con saltos desde el codificador al decodificador. ([3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. Senior Member, IEEE: Octubre 2016).

Esta red utiliza un codificador-decodificador de estilo VGG (al igual que el anterior), donde el re muestreo en el decodificador se realiza mediante convoluciones transpuestas, pero además emplea conexiones de salto aditivas desde el codificador al decodificador.

En una red encoder-decoder, la imagen de entrada se pasa a través de una serie de capas que disminuyen progresivamente, hasta una capa de çuello de botella", donde en ese punto el proceso se invierte (este proceso se puede ver en la ilustración de la Figura 21). Esto produce que la red requiera básicamente que todo el flujo de información pase por todos las capas, incluido .[el] cuello de botella", de manera de que en la traducción de imágenes haya una gran cantidad de información de bajo nivel compartida entre la entrada y la salida, información que sería deseable transferir esta información directamente. Luego, el modelo decodificador-codificador con salto (Encoder-Decoder with skip) logra esto.

En una red encoder-decoder normal, la imagen de entrada se pasa a través de una serie de capas que disminuyen progresivamente, hasta una capa de cuello de botella, donde en ese punto el proceso se invierte (este proceso se puede ver en la ilustración de la Figura). Esto produce que la red requiera básicamente que todo el flujo de información pase por

todos las capas, incluido el llamado “cuello de botella", de manera de que en la traducción de imágenes hay una gran cantidad de información de bajo nivel compartida entre la entrada y la salida, información que sería deseable transferir directamente. Luego, el modelo decodificador-codificador con salto (Encoder-Decoder with skip) logra esto, por lo que por esto entra en la comparación de modelos, ya que puede realizar a priori la tarea requerida.

Un ejemplo concreto es el caso de la colorización de la imagen de entrada, por lo que le damos a esta información un cierto atajo, de manera de sortear el çuello de botella", agregando la conexión de omisión o salto (skip) a la red codificador-decodificador (encoder-decoder).

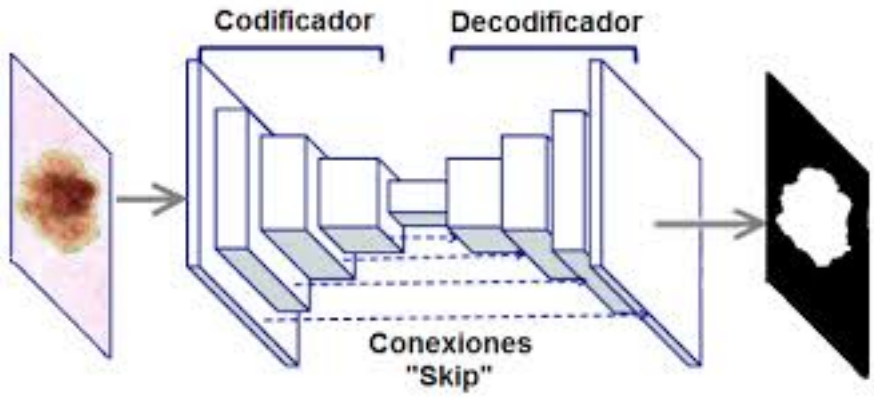

Figura 2.17: Ejemplo general de modelo encoder-decoder with skip

Capítulo 3

Metodología

Para realizar el modelo propuesto se requiere de un sistema computacional con GPU (Graphics Processing Unit) y la utilización del software Python y Tensor Flow, de manera de desarrollar el modelo empleando dos técnicas principales de Deep Learning, redes neuronales convolucionales (CNN) y redes neuronales recurrentes (RNN).

La metodología a seguir consiste en las siguientes etapas:

1. Recibir datos reales desde la industria.

1.1 Al recibir los datos se debe proceder al entendimiento de los mismos, de manera de efectuar análisis de calidad de datos, análisis descriptivo de datos, etc.

2. Prueba de modelos de segmentación.

2.1 Se procede a la prueba de distintos modelos de segmentación, de manera de escoger uno para la posterior profundización en el mismo.

2.2 Luego de tener el modelo escogido, se procede a variar distintos parámetros, obteniendo información de comportamiento respecto a otros modelos.

3. Realizar el modelo computacional en base a Deep Learning teniendo en cuenta los datos recibidos.

4. Aplicación del modelo a los datos.

5. Se procede a una comprobación de los datos generados por el modelo, en la cual se observará la veracidad y coherencia para la generación de protocolos.

6. Se procede a la realización de protocolos, tasas de falla y conclusiones finales.

La metodología se puede ver de mejor manera en el siguiente diagrama de flujo:

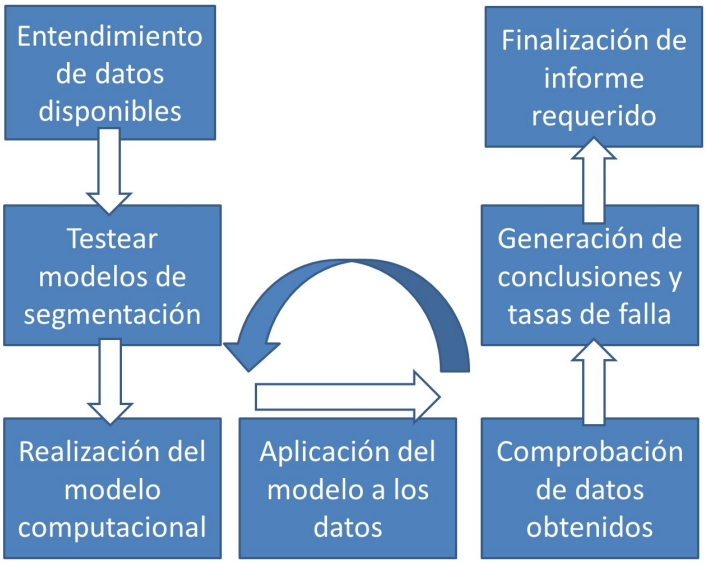

Figura 3.1: Ilustración de Metodología general

3.1. Recursos

Para la realización del tema de memoria se requiere básicamente de un sistema computacional capaz de procesar los datos correspondientes a los sensores de vibraciones y la realización del modelo computacional, realizado en base al aprendizaje en Deep Learning, es decir un sistema que cuente con GPU (Graphics Processing Unit) y el software Python y Tensor Flow.

Python es un lenguaje de programación de código abierto, potente y fácil de aprender. Tiene estructuras de datos eficientes de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. La sintaxis y la escritura dinámica de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para la creación de scripts y el desarrollo rápido de aplicaciones en muchas áreas en la mayoría de las plataformas. [15]. Python se puede utilizar de buena manera junto a Tensor Flow para programar algoritmos en base a Machine Learning y Deep Learning, como los que se propone a realizar.

TensorFlow es una biblioteca compleja para el cálculo numérico distribuido utilizando gráficos de flujo de datos, además permite entrenar y correr redes neuronales muy grandes. Fue creado en Google y admite muchos de sus aplicaciones de aprendizaje automático a gran escala ([5] Prefacio). El principio básico de Tensor Flow es el siguiente, primero se realiza en Python los cálculos gráficos a realizar, para luego insertarlos en Tensor Flow y procesar los cálculos de manera eficiente utilizando código C++ debidamente optimizado, por lo que se es capaz de realizar las tareas propuestas en base a Deep Learning.

Capítulo 4

Desarrollo

4.1. Comentarios iniciales

La principal tarea a realizar será programar el sistema computacional (algoritmo) mediante variación de parámetros hasta llegar a un óptimo, y entrenarlo a través de una información o data específica, luego los principales desafíos al trabajar en Machine Learning son generar este algoritmo de manera satisfactoria, es decir que cumpla de buena manera con el objetivo para el que fue diseñado, y segundo que la información utilizada en el modelo sea preparada de buena manera para procesarla en el algoritmo.

Sabiendo lo anterior, las etapas críticas del tema a realizar principalmente son la realización del modelo de manera satisfactoria y la aplicación del modelo a los datos obtenidos desde la industria.

La primera etapa crítica corresponde al entendimiento de los datos en conjunto, puesto que si los datos no se entienden de una manera correcta se procederá también a una mal procesamiento de los datos recibidos, esto es de gran importancia, puesto que de igual manera que en la programación del modelo, si la información utilizada para el modelo no es óptima, la información generada desde el modelo no tendrá coherencia y no seguirá una línea aplicable y acorde a la realidad.

La segunda etapa crítica corresponde a la realización del modelo, puesto que se requiere un aprendizaje en base a Deep Learning y conocimientos varios de programación en Python y Tensor Flow, además, se debe comprobar que el modelo computacional sea realizado satisfactoriamente, por lo que se deben realizar pruebas antes y después de aplicar el modelo a los datos obtenidos desde la industria.

La tercera etapa crítica, correponde a la aplicación del modelo a los datos recibidos, esta etapa correspondería básicamente a la segunda comprobación del modelo computacional, de manera de estar seguros de que se haya realizado de manera adecuada y así poder generar protocolos que sean fiables y acordes a la realidad donde se aplicarán.

Por último, se requiere la prueba de datos externos a la muestra recibida, de manera de

aplicar el modelo a datos reales y completar un modelo computacional que sea aplicable a un conjunto de grietas más amplio a nivel industrial.

4.2. Datos

Los datos recibidos corresponden a 1638 imágenes en concreto obtenidas mediante la obtención de la mismas desde archivos de video. Estos archivos de video son grabados desde superficies de concreto que presentan grietas de distinto tamaño y envergadura. Ejemplos de las imágenes reales en grietas se pueden ver a continuación:

Figura 4.1: Ejemplos de grieta real

Las imágenes descritas anteriormente son clasificadas en binario, de manera de que la grieta se limita de color blanco (color binario 0), y todo lo que no es grieta de color negro (color binario 1), de manera de segmentar las mismas y obtener muestras para las posteriores pruebas de predicción, quedando de la siguiente forma (no son necesariamente las mismas que las imágenes anteriores, son ejemplos de cómo quedarían):

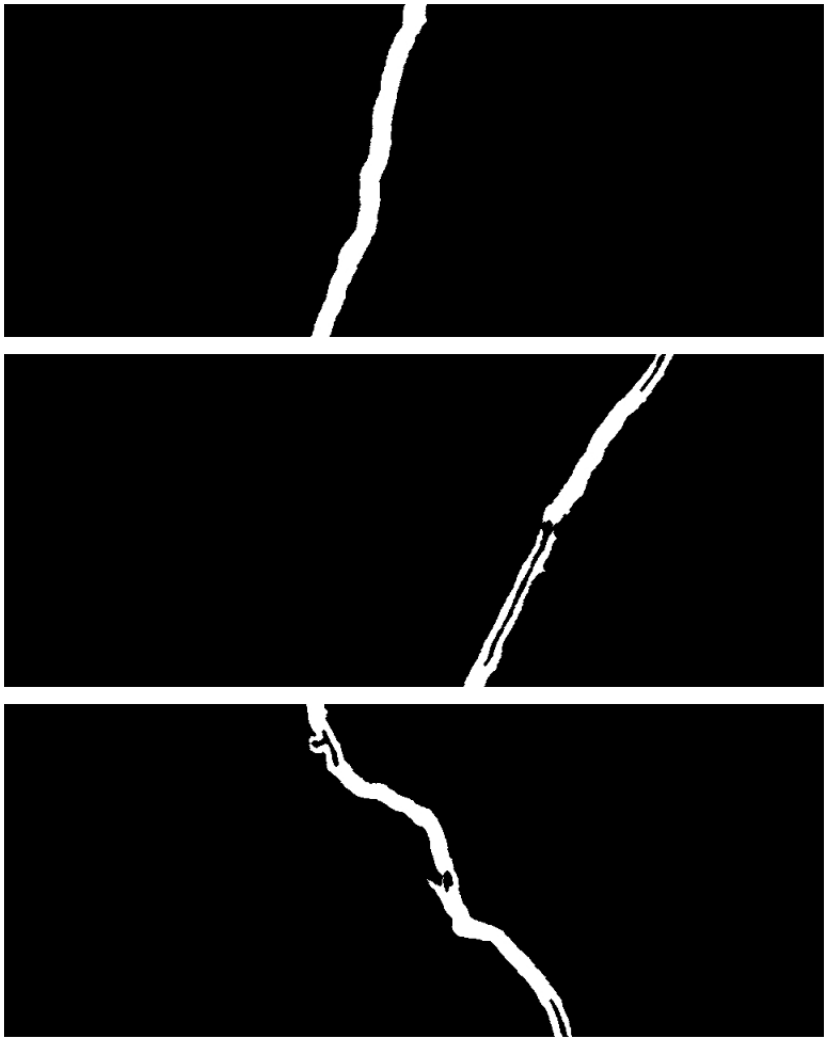

Figura 4.2: Ejemplos de grieta en binario

Luego, cada una de estas imágenes es separada en imágenes de 96x96 pixeles, de manera de ingresarlas al modelo computacional y que sean más fáciles de procesar. Las imágenes separadas de 96x96 son procesadas de manera que el mismo pueda predecir de manera clara una grieta cualquiera.

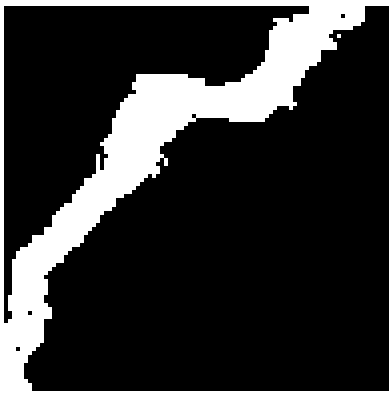

Figura 4.3: Ejemplo de grieta en 96x96 en binario

4.3. Pre-procesamiento

4.3.1. Variación de parámetros

Para el procesamiento de los modelos computacionales se realiza una variación de parámetros hasta encontrar valores óptimos o que alcancen un IOU de gran magnitud (mayor al 90 por ciento), esto a su vez teniendo en cuenta el tiempo de procesamiento del mismo algoritmo con esos parámetros, teniendo en cuenta que para el óptimo exista un equilibrio entre tiempo de procesamiento y mayor valor de IOU.

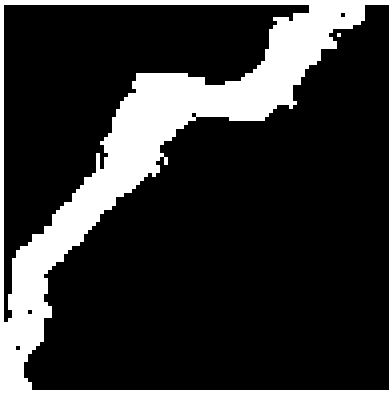

Figura 4.4: Ejemplo de imágen de grieta ya definida en vista binaria

Figura 4.5: Ejemplo de imágen de grieta ya definida en vista real

En el anexo se presentan los gráficos de número de épocas versus IOU promedio y la comparación de cada uno de los procesamientos de la predicción realizada con una imagen de grieta ya definida. Inicialmente se variaran los parámetros de épochs y batch size, para luego seguir con las variaciones de otros parámetros más específicos tales como el número de imágenes de validación, número de checkpoints (número de veces que se guarda el proceso para evitar tener que repetir el mismo por errores u cortes de energía), brillo, entre otros. La variación de parámetros están separados en intentos (o TRYs), que son especificados en cada una de las imágenes anexadas.

Capítulo 5

Experimentos y resultados

5.1. DeepLabV3+

5.1.1. Experimento 1: DeepLabV3+

En el experimento 1 se pretende encontrar el IOU óptimo (tomando en cuenta el tiempo de procesamiento) para el modelo DeepLabV3+, el procedimiento a realizar es el de variar los parámetros utilizados en el modelo partiendo por parámetros tales como épochs y batch size, y continuando con otros parámetros más específicos.

A continuación se presentan las siguientes tablas resumen de los procesamientos para el modelo DeepLabV3+ en su variación de parámetros:

Test	**Epochs**	**Batch size**	**Hours**	**IOU (train)**	**IOU (test)**
1	100	100	2	0,7620	0,6869
2	100	1	2 1/2	0,9312	0,9039
3	50	1	1 1/2	0,9305	0,9039
4	25	1	1/2	0,9328	0,9030
5	300	1	7 1/2	0,9312	0,9172
6	10	1	1/3	0,9211	0,8920
7	100	1 (flip)	2	0,9312	0,9120

Tabla 5.1: Tabla de resultados DeepLabV3+ (Epochs y batch size)

En el caso del modelo DeepLabV3+, podemos ver que nuestro óptimo lo tenemos con 100 épocas y un número de 1 batch size, con un IOU aproximado de 0,9312 y una duración de procesamiento de 2 horas y media aproximadamente. En este caso los procesamientos de datos con flip no tienen mayor influencia en los resultados de IOU del modelo computacional, puesto que en el caso de incluir el flip a pesar de disminuir el tiempo en el proceso, aumenta un poco el resultado pero no significativamente, dado que la diferencia de IOU (train) e IOU (test) es mayor que el caso sin flip.

Luego de obtener este óptimo fijamos el número de épocas y batch size, de manera de variar luego parámetros más específicos (en segunda instancia, número de imagenes de validación,

rotación y otros). A continuación se presenta una segunda tabla resumen con parámetros a variar, con un número fijo de epochs y batch size de 100 y 1.

Test	n img_val	Rotation	Hours	IOU (train)	IOU (test)
1	20	180	1 1/4	0,9285	0,9285
2	100	0	2 1/4	0,9271	0,9166
3	100	0	2 1/2	0,9274	0,9134
4	500	0	3 1/4	0,9250	0,9146
5	1000	0	3 1/2	0,9274	0,9161
6	679	0	3 1/2	0,9274	0,9171
7	679	180	2 2/3	0,9277	0,9257

Tabla 5.2: Tabla de resultados DeepLabV3+ (Número de imagenes de validación y rotación)

Como podemos observar, el cambio de rotación logra un cambio positivo en el IOU final, siendo éste un aumento relevante en el IOU final de la fase de test comparado con otros parámetros, además, hay una baja considerable en el tiempo de procesamiento, por lo que se considerarán 180 fijos para las siguientes evaluaciones. Luego el número de imágenes de validación hace que el IOU disminuya, esto es lógico puesto que se realiza una validación con un mayor número de imágenes que en la primera tabla, por lo que a pesar de disminuir el IOU se tiene una mayor confiabilidad al mayor comparación de resultados.

Luego de variar estos parámetros, seguimos con una fijación de parámetros en epochs, batch size, número de imágenes de validación y rotación (100, 1, 679 y 180 respectivamente). El número de imágenes de validación se fija en 679 debido a que hay 679 imágenes de validación, es decir se validarán los resultados de las imágenes en todas las imágenes de validación. Se sigue la variación de parámetros pero esta vez en parámetros de imagen (tamaño de imagen de entrada y brillo de la misma) de manera de observar también el comportamiento del IOU en otros tipos de tamaño. Se presenta la siguiente tabla de resumen con la variación de estos parámetros.

Test	Image size	Brightness	Hours	IOU (train)	IOU (test)
1	24 x 24	0	2 1/2	0,9617	0,6836
2	24 x 24	0,5 (50 %)	2 1/2	0,9581	0,7094
3	24 x 24	0,8 (80 %)	2	0,9534	0,6118
4	48 x 48	0	2 3/4	0,9219	0,8810
5	48 x 48	0,5 (50 %)	2 1/3	0,9072	0,8820
6	48 x 48	0,8 (80 %)	2 1/2	0,9221	0,8502
7	12 x 12	0	2	0,9790	0,6904
8	12 x 12	0,5 (50 %)	2 1/2	0,9904	0,5967
9	12 x 12	0,8 (80 %)	2 1/2	0,9898	0,4956

Tabla 5.3: Tabla de resultados DeepLabV3+ (Tamaño de imagen y brillo)

Como podemos observar en la tabla anterior, el aumento de brillo en las imágenes no tiene mayor relevancia a la hora de observar los resultados de IOU, por lo que el aumento de brillo como parámetro a variar queda descartado, fijando el brillo en 0. Por otro lado a menor tamaño de imagen mayor es el IOU de entrenamiento pero menor es el IOU de

prueba, esto se debe en primera instancia a que mientras menor sea el tamaño de imagen mayor es el área "negra"(o de fondo) en la imagen de grieta, por lo que el IOU aumenta al tener mayor coincidencia de fondo en vez de con la misma grieta. Algo parecido pasa con el IOU de prueba, donde al ser de menor tamaño la imagen en comparación a las imágenes de prueba se produce una menor coincidencia entre las mismas, resultando así en una baja considerable en el IOU.

5.1.2. Resultado experimento 1: DeepLabV3+

Con los parámetros para la optimización del modelo ya fijados, podemos obtener los gráficos para el óptimo encontrado, tales como número de épocas versus exactitud promedio de validación, número de épocas versus IOU promedio y número de épocas versus pérdida promedio.

Parámetro	**Valor**
Épocas (Epochs)	100
Tamaño de lote (Batch size)	1
Rotación (Rotation)	180 grados
Imágenes de validación (Validation images)	679
Brightness (Brillo)	0
Vuelta (Flip)	False
Tiempo de entrenamiento	3 3/4 hrs.
IOU de entrenamiento (IOU train)	0.9252 (93 por ciento aprox.)
IOU de prueba (IOU test)	0.9277 (93 por ciento aprox.)

Tabla 5.4: Tabla de resultados óptimos para DeepLabV3+

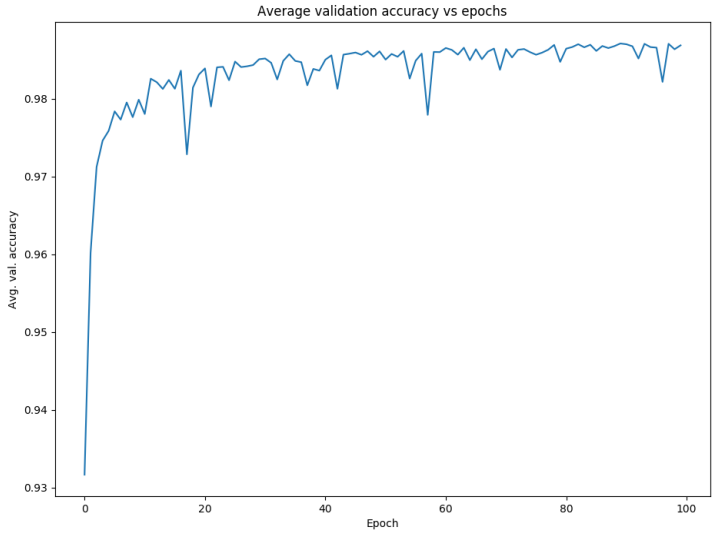

Figura 5.1: Gráfico Accuracy vs Epochs - Óptimo de modelo DeepLabv3+

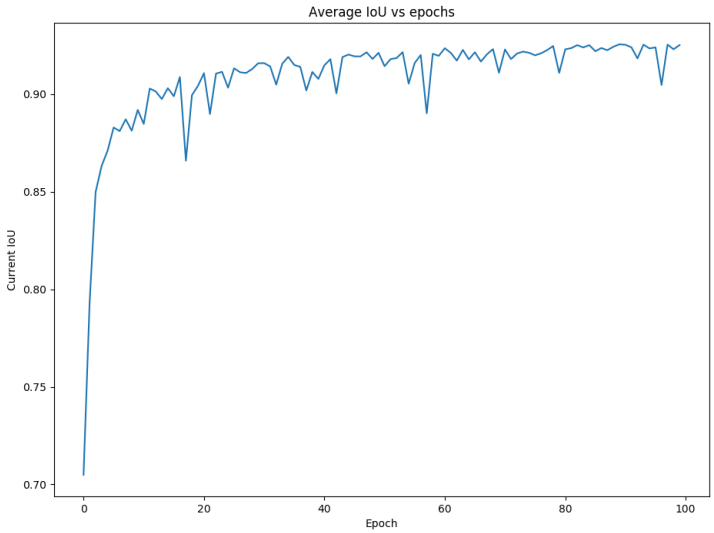

Figura 5.2: Gráfico IOU vs Epochs - Óptimo de modelo DeepLabv3+

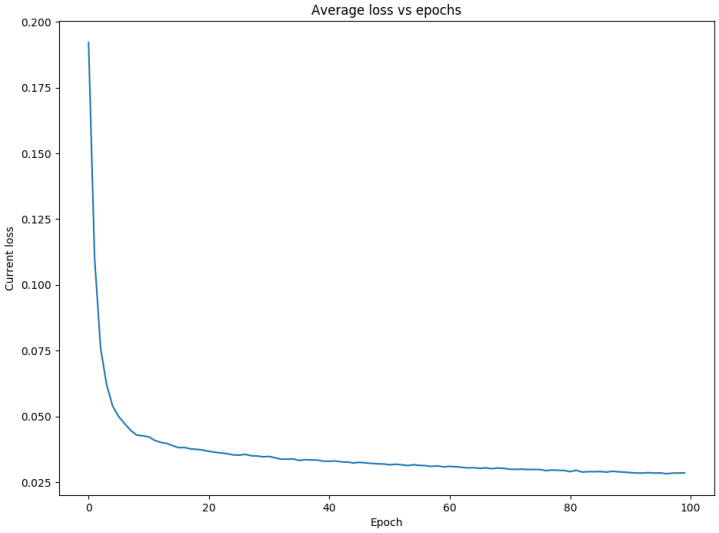

Figura 5.3: Gráfico Loss vs Epochs - Óptimo de modelo DeepLabv3+

5.2. Encoder-Decoder

5.2.1. Experimento 2: Encoder-Decoder

En el experimento 2 se pretende encontrar el IOU óptimo (tomando en cuenta el tiempo de procesamiento) para el modelo Encoder-Decoder, el procedimiento a realizar es el de variar los parámetros utilizados en el modelo partiendo por parámetros tales como épochs y batch size, y continuando con otros parámetros más específicos.

A continuación se presentan las siguientes tablas resumen de los resultados para el procesamiento del modelo Encoder-Decoder en su variación de parámetros:

Epochs	**Batch size**	**Hours**	**IOU (train)**	**IOU (test)**
100	1	5	0,9108	0,9013
300	14	5 1/2	0,8728	0,8552
300	14 (con flip)	4	0,8949	0,8912
300	1 (con flip)	12	0,9207	0,9149
300	1	12	0,9047	0,8952
100	1 (con flip)	5	0,8998	0,8919
100	1 (con flip)*	4	0,9096	0,9045
100	1 (con flip)**	4 1/4	0,9090	0,9005
100	1 (con flip)***	4 1/4	0,9029	0,9050

Tabla 5.5: Tabla de resultados Encoder Decoder (Epochs y batch size)

- *(aprendizaje cambiado de 0,001 a 0,00001).
- **(aprendizaje cambiado de 0,001 a 0,0001).
- ***(aprendizaje cambiado de 0,001 a 0,00001 y un número de imágenes de validación de 136).

En el caso del modelo Encoder-Decoder, podemos ver que nuestro óptimo para este caso, lo tenemos con 100 épocas y un número de 1 batch size, al igual que en el caso anterior. Se obtiene un IOU aproximado de 0,9108 y una duración de procesamiento de 5 horas aproximadamente. En este caso los procesamientos de datos con flip no tienen mayor influencia en los resultados de IOU del modelo computacional, puesto que en el caso de incluir el flip en el proceso disminuye un poco el resultado. Además en este caso se prueba el cambio del parámetro de aprendizaje, el cual tampoco tiene gran influencia en el resultado de IOU, por lo que se descarta su variación.

Luego de obtener los valores de batch size y epochs, podemos notar que estos son los mismo valores de batch size y epochs que para el modelo DeepLabv3+, de manera que no hace falta variar luego parámetros más específicos (en segunda instancia, número de imágenes de validación, rotación y otros), luego, el modelo se procesa para su óptimo con los mismos parámetros del modelo DeepLabV3+.

5.2.2. Resultado Experimento 2: Encoder-Decoder

Siguiendo con esto y con los parámetros para la optimización del modelo ya fijados, podemos obtener los gráficos para el óptimo encontrado, tales como número de épocas versus exactitud promedio de validación, número de épocas versus IOU promedio y número de épocas versus pérdida promedio.

Parámetro	**Valor**
Épocas (Epochs)	100
Tamaño de lote (Batch size)	1
Rotación (Rotation)	180
Imagenes de validación (Validation images)	679
Brillo (Brightness)	0
Vuelta (Flip)	False
Tiempo de entrenamiento	5 1/2 hrs.
IOU de entrenamiento (IOU train)	0.8936 (90 por ciento aprox.)
IOU de prueba (IOU test)	0.8930 (90 por ciento aprox.)

Tabla 5.6: Tabla de resultados óptimos para Encoder-Decoder

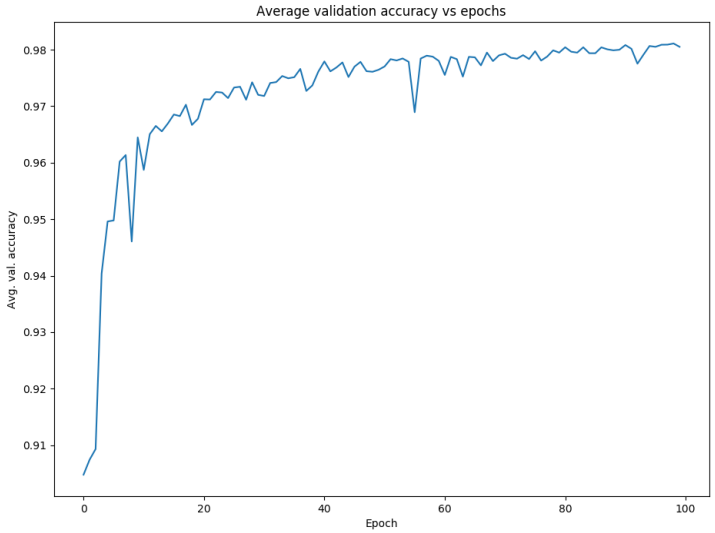

Figura 5.4: Gráfico Accuracy vs Epochs - Óptimo de modelo Encoder-Decoder

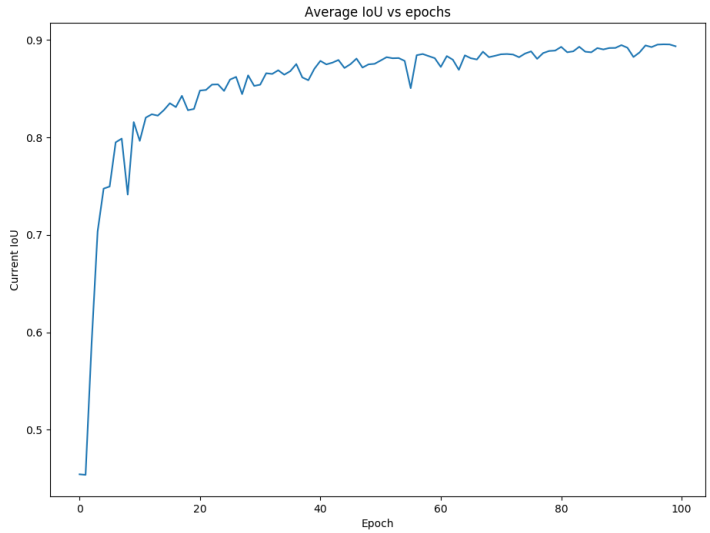

Figura 5.5: Gráfico IOU vs Epochs - Óptimo de modelo Encoder-Decoder

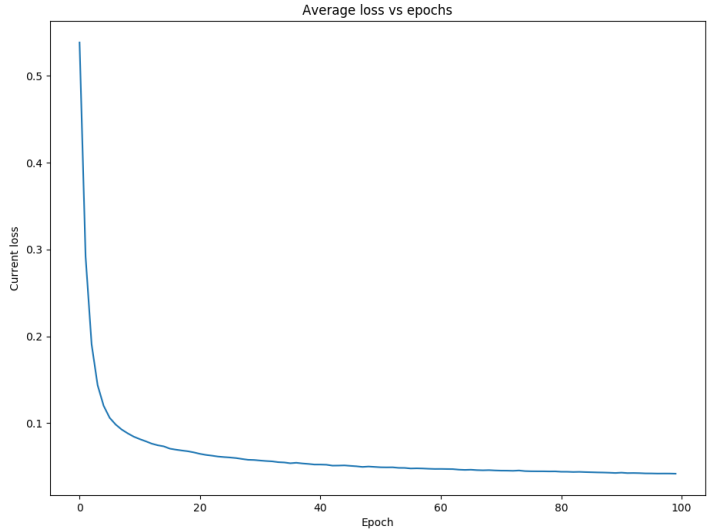

Figura 5.6: Gráfico Loss vs Epochs - Óptimo de modelo Encoder-Decoder

5.3. Encoder decoder skip

5.3.1. Experimento 3: Encoder-Decoder with skip

En el experimento 3 se pretende encontrar el IOU óptimo (tomando en cuenta el tiempo de procesamiento) para el modelo Encoder-Decoder with skip, el procedimiento a realizar es el de variar los parámetros utilizados en el modelo partiendo por parámetros tales como épochs y batch size, y continuando con otros parámetros más específicos.

A continuación se presentan las siguientes tablas resumen de los resultados para el procesamiento del modelo Encoder-Decoder skip en su variación de parámetros:

Epochs	**Batch size**	**Hours**	**IOU (train)**	**IOU (test)**
100	1	2	0,9551	0,9543
100	14	3	0,9127	0,9205
300	1	13	0,9572	0,9554
100	1 (con flip)	2 1/2	0,9570	0,9572
300	1 (con flip)	13	0,9560	0,9567

Tabla 5.7: Tabla de resultados Encoder Decoder with skip (Epochs y batch size)

En el caso del modelo Encoder-Decoder con skip, podemos ver que nuestro óptimo lo tenemos con 100 épocas y un número de 1 batch size, con un IOU aproximado de 0,9543 y una duración de procesamiento de 2 horas aproximadamente. En este caso los procesamientos de datos con flip no tienen mayor influencia en los resultados de IOU del modelo computacional, puesto que en el caso de incluir el flip a pesar de aumentar el resultado no lo hace significativamente, además de aumentar el tiempo, por lo que no calza o compite con el óptimo buscado.

Luego de obtener los valores de batch size y epochs, podemos notar que estos son los mismo valores de batch size y epochs que para el modelo deeplabv3plus y encoder-decoder, de manera que no hace falta variar luego parámetros más específicos (en segunda instancia, número de imágenes de validación, rotación y otros), luego, el modelo se procesa para su óptimo con los mismos parámetros del modelo DeepLabV3+.

5.3.2. Resultado Experimento 3: Encoder-Decoder with skip

Siguiendo con esto y con los parámetros para la optimización del modelo ya fijados, podemos obtener los gráficos para el óptimo encontrado, tales como número de épocas versus exactitud promedio de validación, número de épocas versus IOU promedio y número de épocas versus pérdida promedio.

Parámetro	**Valor**
Épocas (Epochs)	100
Tamaño de lote (Batch size)	1
Rotación (Rotation)	180 grados
Imagenes de validación (Validation images)	679
Brillo (Brightness)	0
Vuelta (Flip)	False
Tiempo de entrenamiento	5 2/3 hrs.
IOU de entrenamiento (IOU train)	0.9465 (95 por ciento aprox.)
IOU de prueba (IOU test)	0.9451 (95 por ciento aprox.)

Tabla 5.8: Tabla de resultados óptimos para Encoder-Decoder with skip

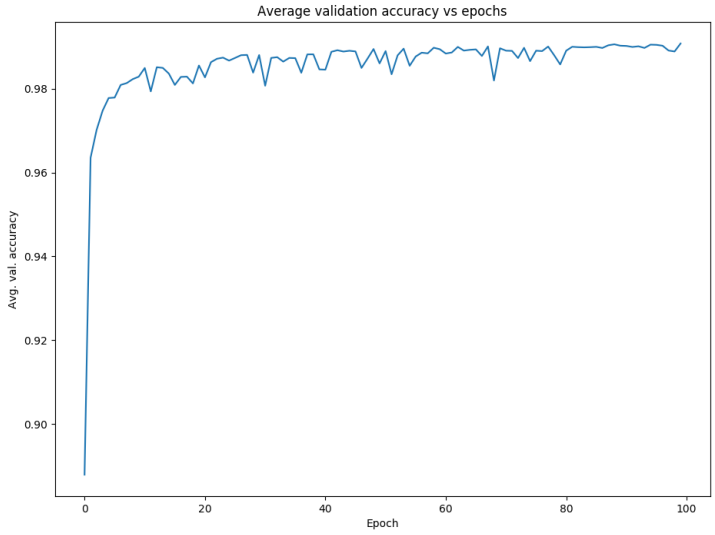

Figura 5.7: Gráfico Accuracy vs Epochs - Óptimo de modelo Encoder-Decoder with skip

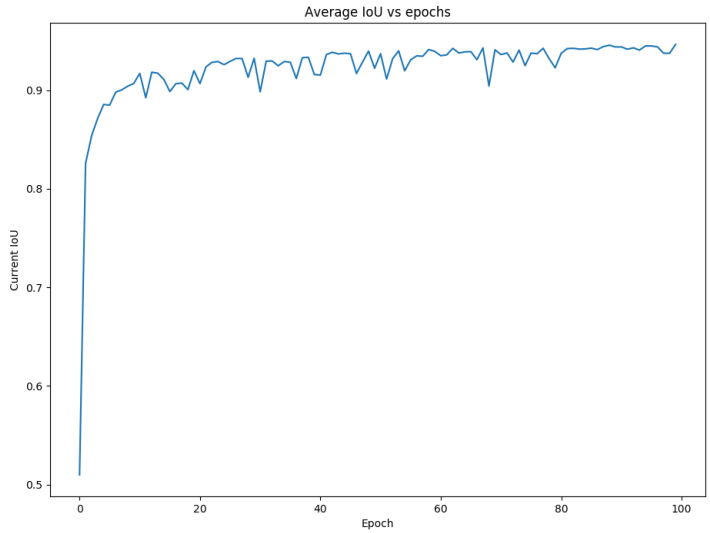

Figura 5.8: Gráfico IOU vs Epochs - Óptimo de modelo Encoder-Decoder with skip

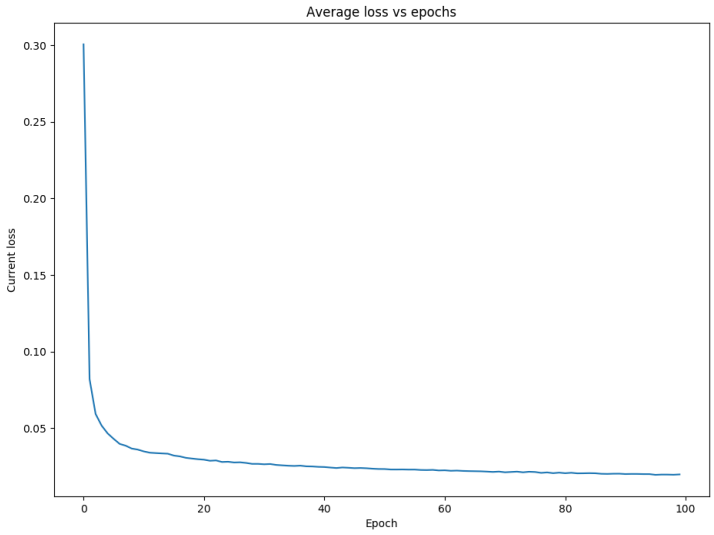

Figura 5.9: Gráfico Loss vs Epochs - Óptimo de modelo Encoder-Decoder with skip

5.4. Mapa de calor

5.4.1. Experimento 4: Verificación por mapa de calor

En el experimento 4 se pretende verificar que el procesamiento del codificador del modelo sea correcto, esto mediante la comprobación con un mapa de calor observando las áreas de detección de la grieta. Estos mapas de calor muestran básicamente el área de activación del clasificador hacia las imágenes de grietas, donde los colores más cálidos representan la activación más alta (es decir, áreas más relevantes) y los colores fríos las más bajas (es decir,

áreas no relevantes).

5.4.2. Resultado experimento 4: Verificación por mapa de calor

A continuación en las imágenes siguientes se puede apreciar esta verificación, donde efectivamente el mapa de calor muestra áreas más relevantes en las zonas de grieta.

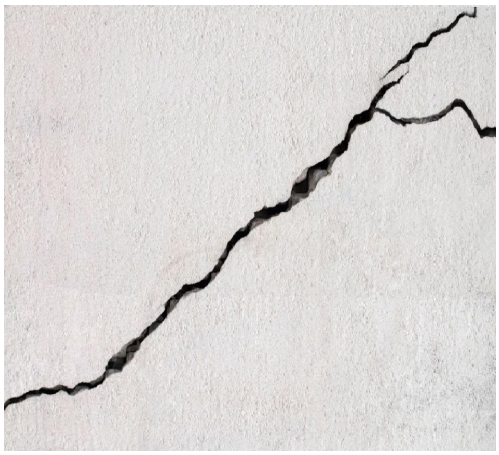

Figura 5.10: Imagen de ejemplo 1 de grieta real.

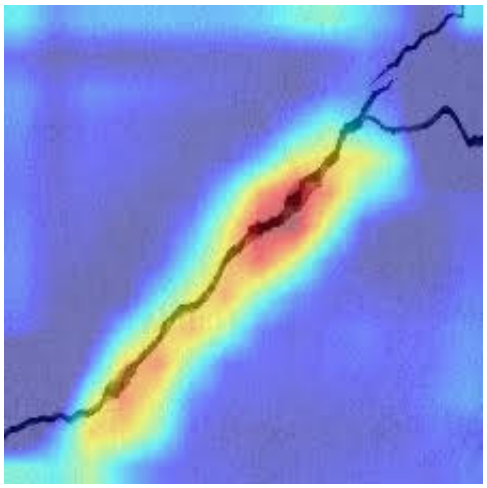

Figura 5.11: Imagen de ejemplo 1 de grieta real analizada a través de mapa de calor.

Figura 5.12: Imágen de ejemplo 2 de grieta real.

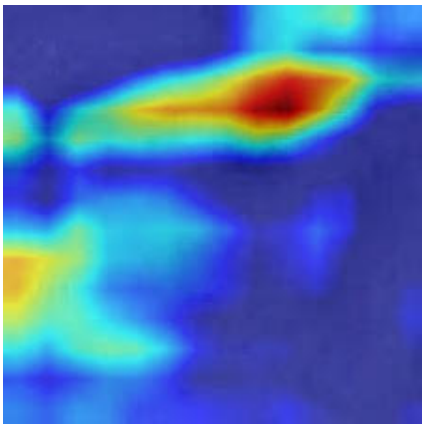

Figura 5.13: Imágen de ejemplo 2 de grieta real analizada a través de mapa de calor.

5.5. Comparación de resultados

En cada sección de modelo, se presenta una tabla resumen del mismo con la variación de parámetros en cada proceso, tiempo de procesamiento e IOUs de entrenamiento y de prueba, con gráficos de cada procesamiento y de cada modelo adjuntado en la sección de anexo.

A continuación se presenta una tabla resumen de los resultados del experimento 1: DeepLabv3+:

Parámetro	**Valor**
Épocas (Epochs)	100
Tamaño de lote (Batch size)	1
Rotación (Rotation)	180 grados
Imagenes de validación (Validation images)	679
Brillo (Brightness)	0
Vuelta (Flip)	False
Tiempo de entrenamiento	3 3/4 hrs.
IOU de entrenamiento (IOU train)	0.9252 (93 por ciento aprox.)
IOU de prueba (IOU test)	0.9277 (93 por ciento aprox.)

Tabla 5.9: Tabla de resultados óptimos para DeepLabV3+

Luego, usando estos parámetros se realiza una comparación entre los 3 principales modelos seleccionados a fin de observar los cambios en cada modelo respecto a los mismos parámetros de óptimo. Se presenta la siguiente tabla de estos resultados y posteriormente una comparativa de gráficos de IOU promedio v/s épocas:

	DeepLabV3+	**Encoder-Decoder**	**Encoder-Decoder Skip**
IOU train	0.9252	0.8936	0.9465
IOU test	0.9277	0.8930	0.9451
Tiempo de entrenamiento	3 3/4 hrs.	5 1/2 hrs.	5 2/3 hrs.

Tabla 5.10: Tabla de resultados comparativos entre modelos

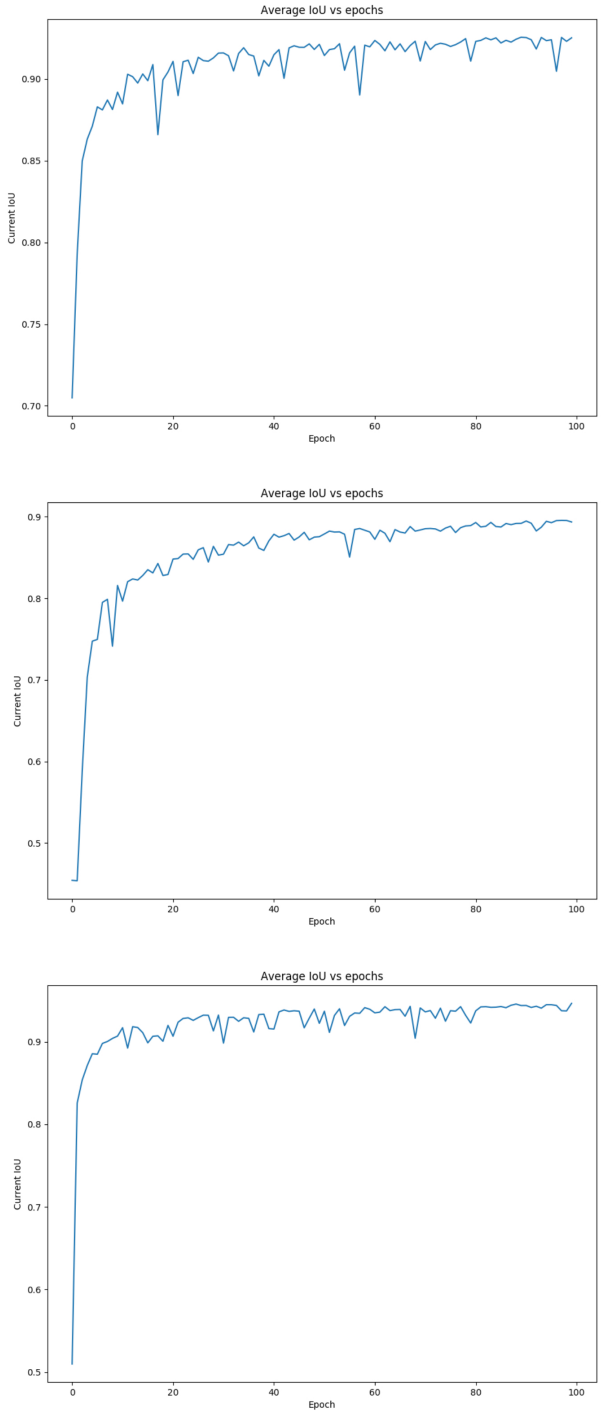

Figura 5.14: Comparación de gráficos de modelos analizados: Gráfico IOU vs Epochs - Óptimo de modelo DeepLabv3+ - Gráfico IOU vs Epochs - Óptimo de modelo Encoder-Decoder - Gráfico IOU vs Epochs - Óptimo de modelo Encoder-DecoderSkip (De arriba hacia abajo).

Capítulo 6

Análisis de resultados

Durante la variación de datos de los modelos analizados se pudo observar el cambio notorio en los gráficos de pérdida (loss) y precisión (accuracy) a medida que se llegaba al número de 100 épochs y 1 batch size (esto es observable en todos los modelos, gráficos de esto ubicados en el anexo). Luego, a medida que se continuaba la variación de parámetros se puede notar la poca o nula relevancia de ciertos parámetros, tales como el flip o el brillo de la imagen de entrada, además de no producir cambio relevante en el tiempo de procesamiento, quedando estos parámetros fijos y sin cambios. Con esto, podemos inferir la importancia de algunos parámetros a la hora de procesar imágenes de grietas.

Luego de obtener los parámetros del óptimo para el modelo DeepLabv3plus, pudimos observar un porcentaje bastante alto en el procesamiento de imágenes, alcanzando un IOU aproximado de 93 por ciento. Teniendo esto en cuenta, se analizan los valores de epochs y batch size para los demás modelos, resultando en un número de epochs y batch size igual al obtenido en el procesamiento del modelo DeepLabv3plus, por lo que se consideran los mismos parámetros a la hora de obtener los resultados en los distintos modelos.

Luego de procesar los distintos modelos, se puede observar una clara ventaja en el tiempo de procesamiento en el modelo DeepLabv3+. A su vez, podemos notar que el modelo que alcanza un IOU de mayor porcentaje es el modelo Encoder-Decoder with skip, alcanzado un 95 por ciento aproximadamente. Por último el valor de IOU para el modelo Encoder-Decoder fue el menor, con un porcentaje de 90 por ciento aproximadamente y con una diferencia en tiempo sólo un poco menor al de su par modelo Encoder-Decoder with skip. Ahora, considerando eficiencia en tiempos de procesamiento e IOU alcanzado, podemos notar que el modelo DeepLabV3+ es el mejor de los 3 modelos principales analizados, con un IOU de 93 por ciento (2 por ciento menos que el modelo Encoder-Decoder with skip) y un tiempo de entrenamiento de 3 horas 45 minutos aproximadamente (casi 2 horas menos que en el caso del modelo Encoder-Decoder with skip).

Si observamos los gráficos de IOU vs Epochs en la comparación de resultados, podemos observar que la métrica de IOU en el modelo DeepLabV3+ y en el modelo Encoder-DecoderSkip suben mucho más rápidamente que en el modelo Encoder-Decoder, el cual demora un poco más en llegar a una relativa constante de IOU. Esto se puede ver reflejado en que el IOU

alcanzado finalmente sea menor a los demás modelos.

El aumento de IOU en el modelo Encoder-DecoderSkip respecto al modelo Encoder-Decoder normal viene de la incorporación de saltos, esto puede provocar que la imagen resultante en la salida posea mayor resolución, haciendo que la coincidencia entre imagenes sea mayor, por lo que consecuentemente se obtiene un IOU más alto.

El aumento de IOU en el modelo Encoder-DecoderSkip respecto al modelo DeepLabV3+ viene de la resolución que logra en la imagen de salida el primero, con una mayor resolución en los límites de objeto, algo esencial a la hora de comparar la imágenes para comprobar el IOU en este problema en particular, esto dado que sólo se miden los límites de las grietas y no profundidad y detalles dentro de ellas.

El bajo tiempo de entrenamiento del modelo DeepLabv3plus respecto a los demás modelos viene de la mezcla entre la estructuras de la misma, entre spatial pyramid pooling y encoder-decoder, lo que logra una mayor rapidez en el codificador respecto a los demás modelos (algo relativamente observable en la figura 18).

El modelo DeepLabV3+ a pesar de ser un modelo más sofisticado que los otros modelos analizados (puesto que posee la estructura codificador-decodificador y además la estructura espacial piramidal) y a pesar de tener un menor tiempo de procesamiento, tiene un IOU menor que el modelo Encoder-DecoderSkip, luego, podemos inferir que este modelo a pesar de cumplir holgadamente con el objetivo de identificar automáticamente las grietas, puede estar sobredimensionado, puesto que el modelo Encoder-DecoderSkip también cumple con el objetivo requerido. Esto puede ser subjetivo, puesto que el modelo Encoder-DecoderSkip se demora más en el tiempo de procesamiento (aproximadamente 2 horas más), pero también tiene un 2 por ciento más en IOU que el modelo DeepLabV3+, luego, dependiendo de cuál sea la métrica que nos interese más (tiempo de procesamiento, IOU o una mezcla) podremos obtener el modelo más eficiente. Ahora, tomando en cuenta una eficiencia en tiempo de procesamiento e IOU, podemos concluir que el modelo más eficiente en general es el modelo DeepLabV3+, a pesar de estar sobredimensionado.

Capítulo 7

Conclusiones

Podemos concluir que el objetivo principal es completado, puesto que se realiza la detección automática de grietas a través de la red neuronal de forma satisfactoria logrando resultados de IOU que rondan el 90 por ciento, luego tenemos una correcta detección de las mismas. Además, tenemos una comparación de métricas entre tres modelos diferentes, modelo DeepLabV3+, modelo Encoder-Decoder y modelo Encoder-DecoderSkip.

Por otro lado, el modelo DeepLabv3+ a pesar de ser el más eficiente y cumplir con las expectativas esperadas (resultados satisfactorios y tiempo menor a los demás modelos expuestos) puede ser demasiado para la tarea que se le pide en esta instancia, puesto que la clasificación de grietas perfectamente se puede realizar con un modelo codificador - decodificador simple, como pudimos observar con el modelo encoder-decoder y encoder-decoder with skip, los cuales obtienen resultados aceptables (mayores al 85 por ciento). Este último modelo (Encoder-Decoder with skip) incluso obtiene como resultado un IOU mayor al modelo DeepLabV3+. Claro está que el tiempo de procesamiento del modelo principal es menor a los demás modelos, pero aún los demás modelos cumplen con la tarea de forma apropiada.

Capítulo 8

Bibliografía

[1] Juan Tapia, Enrique Lopez, Claudio Yanez, and Ruben Boroschek: Automatic Crack Detection and Quantification Based on Concrete Bridges Images. In: Universidad Tecnológica de Chile, INACAP - Departamento de Ingeniería Mecánica, Universidad de Chile - Departamento de Ingeniería Civil, Universidad de Chile - Mayo 2019.

[2] Carlos Quintero, Fernando Merchán, Aydee Cornejo, Javier Sánchez-Galán. Uso de redes neuronales convolucionales para el reconocimiento automático de imágenes de macroinvertebrados para el biomonitoreo participativo. En: KnE Engineering 6th Engineering, Science and Technology Conference (2017). Universidad tecnológica de Panama, Grupo de investigación en macroinvertebrados dulceacuícolas de Panamá. Instituto Conmemorativo Gorgas de Estudios de la Salud (ICGES), Instituto de Investigaciones Científicas y Servicios de Alta tecnología AIP (INDICASAT AIP), 2017.

[3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. Senior Member, IEEE: Octubre 2016.

[4] Isis Bonet Cruz, Sain Salazar Martínez, Abdel Rodríguez Abed, Ricardo Grau Ábalo, María M. García Lorenzo. Redes neuronales recurrentes para el análisis de secuencias, Revista cubana de ciencias de la informática (RCCi) Vol. (1), 48-57: Diciembre 2017.

[5] Aurélien Géron. Hands-on Machine Learning with Scikit-Learn and TensorFlow, concepts, tools and techniques to build intelligent systems, 1005 Gravenstein Highway North, Sebastopol , CA 95472, OReilly Media, Inc. 2017.

[6] Sik-Ho Tsang, Review: SegNet (Semantic Segmentation) - Encoder Decoder Arquitecture, Using Max Pooling Indices to Upsample, Outperforms FCN, DeepLabv1, DeconvNet [en línea] <https://towardsdatascience.com/review-segnet-semantic-segmentation-e66f2e30fb96>[consulta : 02 Septiembre 2019].

[7] José Ángel González Barba, Aprendizaje profundo para el procesamiento del lenguaje natural, (Máster universitario en inteligencia artificial, reconocimiento de formas e imagen digital). Valencia, España, Universidad Politécnica de Valencia, 2017. 105 h.

[8] Muneeb ul Hassan, VGG16 – Convolutional Network for Classification and Detection [en línea] <https://neurohive.io/en/popular-networks/vgg16/>[consulta : 11 Septiembre 2019].

[9] Saurabh Pal, Semantic Segmentation: Introduction to the Deep Learning Technique Behind Google Pixels Camera! [en línea] <https://www.analyticsvidhya.com/blog/2019/02/tutorial-semantic-segmentation-google-deeplab/>[consulta : 02 Noviembre 2019].

[10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam, Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, Google Inc, Agosto 2018.

[11] Karen Simonyan, Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Visual Geometry Group, Department of Engineering Science,University of Oxford, April 2015.

[12] Phillip Isola, Jun-Yan Zhu, Alexei Efros: Image-to-Image Translation with Conditional Adversarial Networks. In: University of California, Berkeley - Massachussetts Institute of Technology, November 2016.

[13] Francois Chollet. Deep Learning with Python, Manning publications Shelter Island, Inc. 2017.

[14] Xuefeng Zhao, Shengyuan Li, Hongguo su, Kenneth Loh, Lei Zhou: Image-Based Comprehensive Maintenance and Inspection Method for Bridges Using Deep Learning. In: Conference: ASME 2018 Conference on Smart Materials, Adaptive Structures and Intelligent Systems, April 2018.

[15] Python Software Foundation, The Python Tutorial, Copyright 2001-2019 [en línea] <https://docs.python.org/3/tutorial/>[consulta : 02 Noviembre 2019].

Capítulo 9

Anexos

9.1. Gráficos para variación de parametros principales (epochs y batch size)

A continuación se muestran los distintos graficos para la variación de epochs y batch size de los modelos de comparación, DeepLabv3+, Encoder Decoder y Encoder Decoder with skip, además de la predicción de las imágenes de grietas en binario para cada una de las pruebas (TRY) y modelos. El signo de asterisco (*) muestra la prueba con 100 epochs y 1 batch size para cada uno.

9.1.1. DeepLabv3+

- TRY 1: 100 Epochs - 100 batch size.
- TRY 2: 100 Epochs - 1 batch size*.
- TRY 3: 50 Epochs - 1 batch size.
- TRY 4: 25 Epochs - 1 batch size.
- TRY 5: 300 Epochs - 1 batch size.
- TRY 6: 10 Epochs - 1 batch size.
- TRY 7: 100 Epochs - 1 batch size (con flip).

9.1.2. Encoder decoder

- TRY 1: 100 Epochs - 1 batch size*.
- TRY 2: 300 Epochs - 14 batch size.
- TRY 3: 300 Epochs - 14 batch size (con flip).

- TRY 4: 300 Epochs - 1 batch size (con flip).
- TRY 5: 300 Epochs - 1 batch size.
- TRY 6: 100 Epochs - 1 batch size (con flip)
- TRY 7: 100 Epochs - 1 batch size (con flip)*.
- TRY 8: 100 Epochs - 1 batch size (con flip)**.

9.1.3. Encoder decoder skip

- TRY1: 100 Epochs - 1 batch size*.
- TRY 2: 100 Epochs - 14 batch size.
- TRY 3: 300 Epochs - 1 batch size.
- TRY 4: 100 Epochs - 1 batch size (con flip).
- TRY 5: 300 Epochs - 1 batch size (con flip).

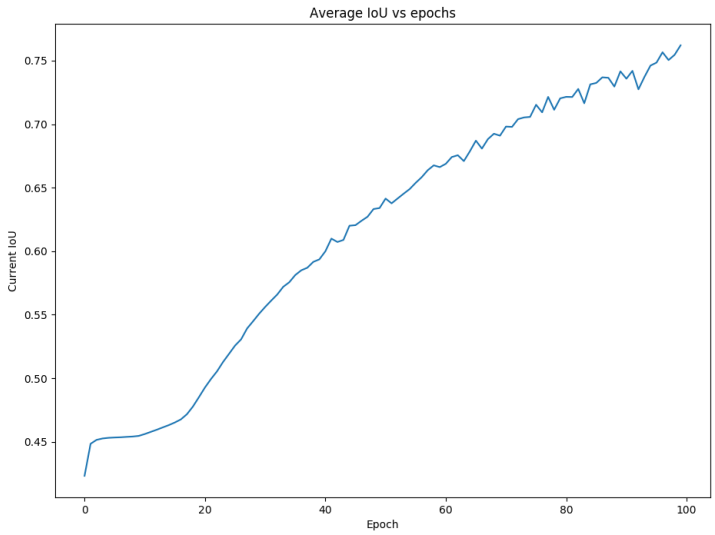

Figura 9.1: Average IOU vs epochs (TRY 1 DeepLabV3+)

- TRY 9: 100 Epochs - 1 batch size (con flip)***.

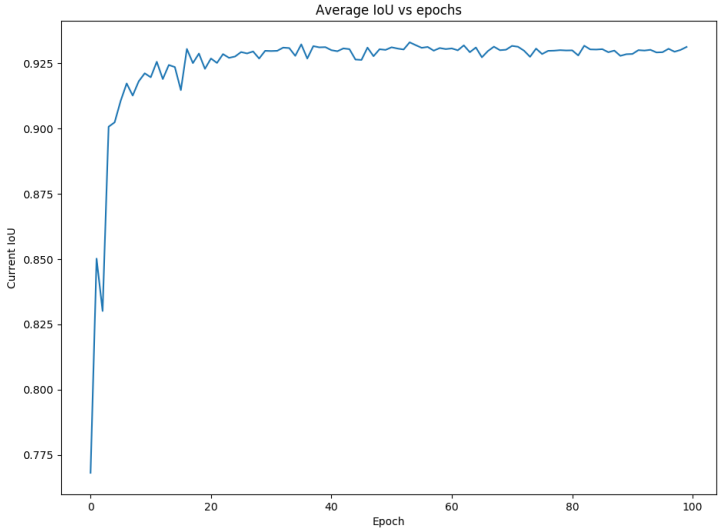

Figura 9.2: Average IOU vs epochs (TRY 2 DeepLabV3+)

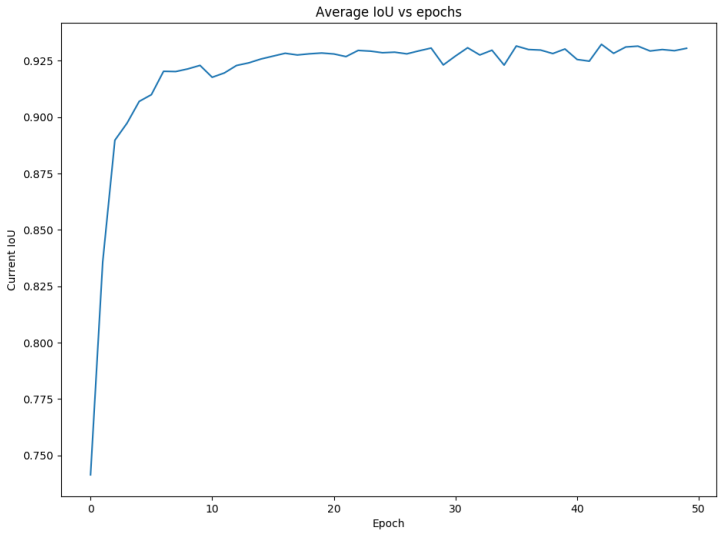

Figura 9.3: Average IOU vs epochs (TRY 3 DeepLabV3+)

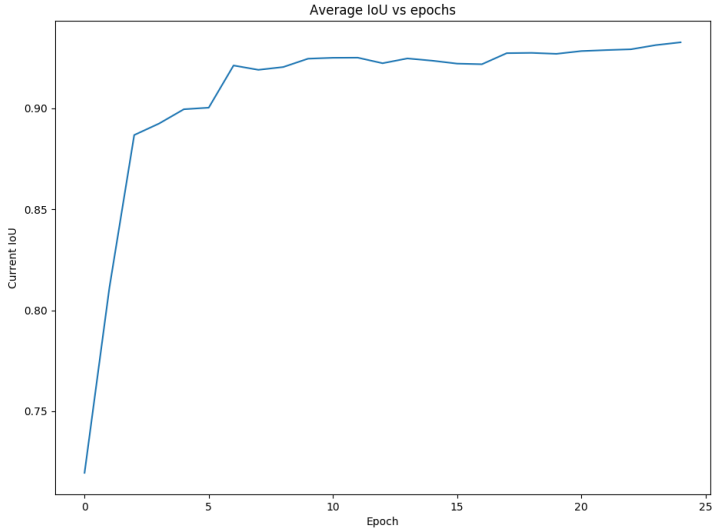

Figura 9.4: Average IOU vs epochs (TRY 4 DeepLabV3+)

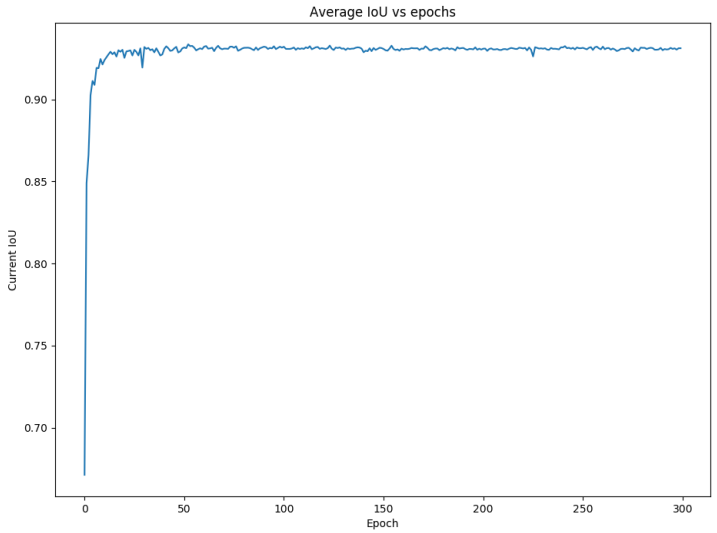

Figura 9.5: Average IOU vs epochs (TRY 5 DeepLabV3+)

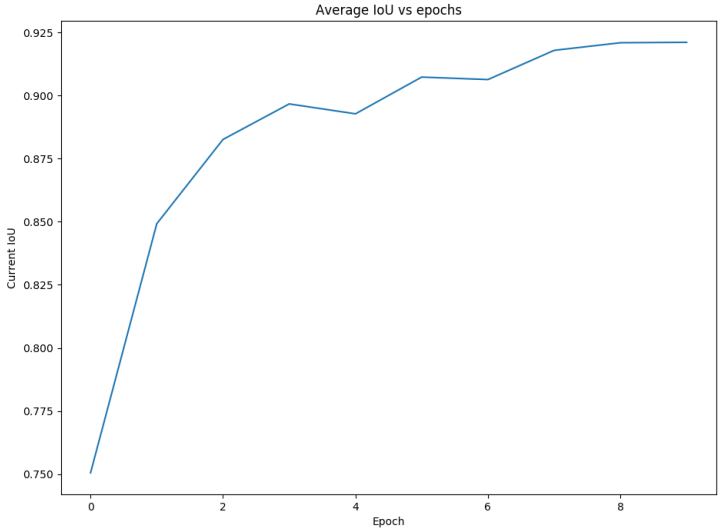

Figura 9.6: Average IOU vs epochs (TRY 6 DeepLabV3+)

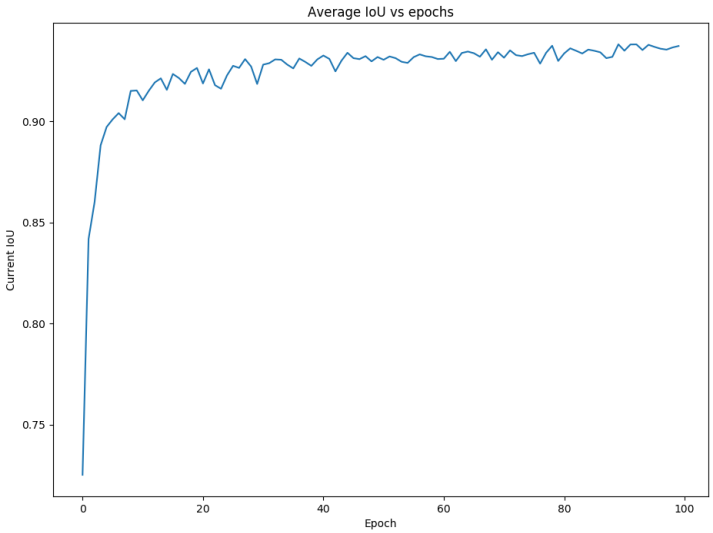

Figura 9.7: Average IOU vs epochs (TRY 7 DeepLabV3+)

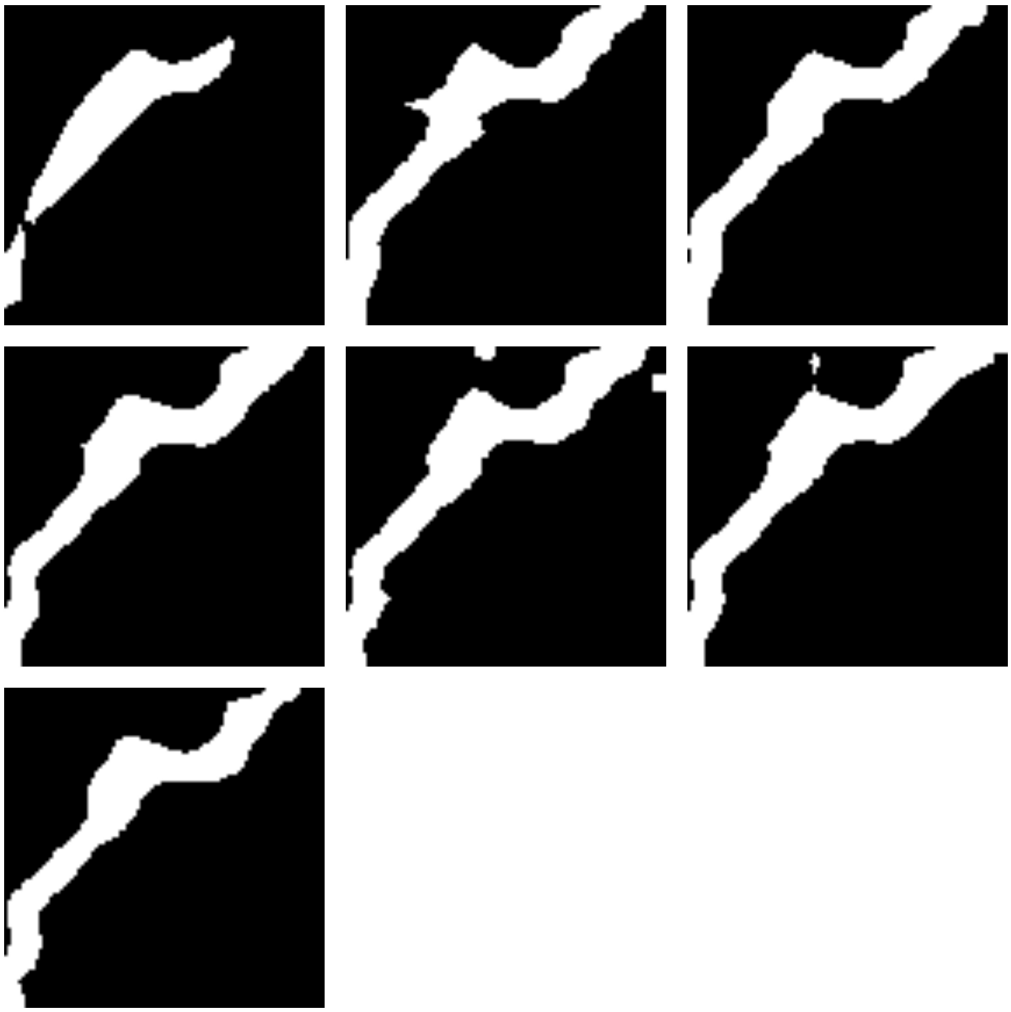

Figura 9.8: Predicción de imagen de grieta en binario (TRY 1 a TRY 7) DeepLabV3+

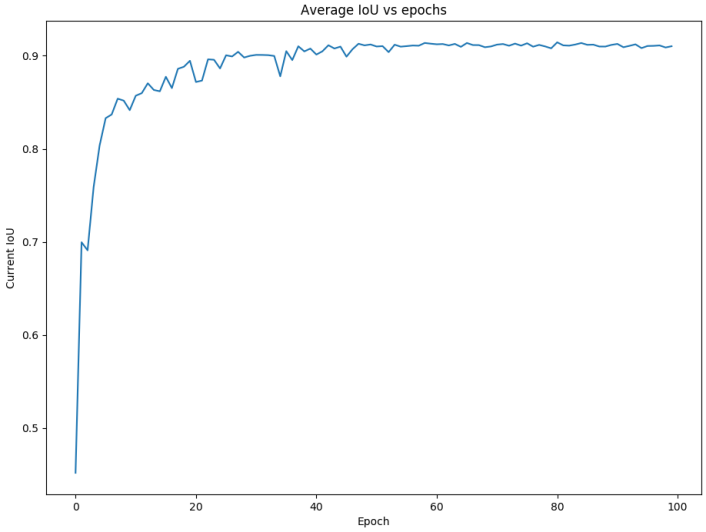

Figura 9.9: Average IOU vs epochs (TRY 1 Encoder-Decoder)

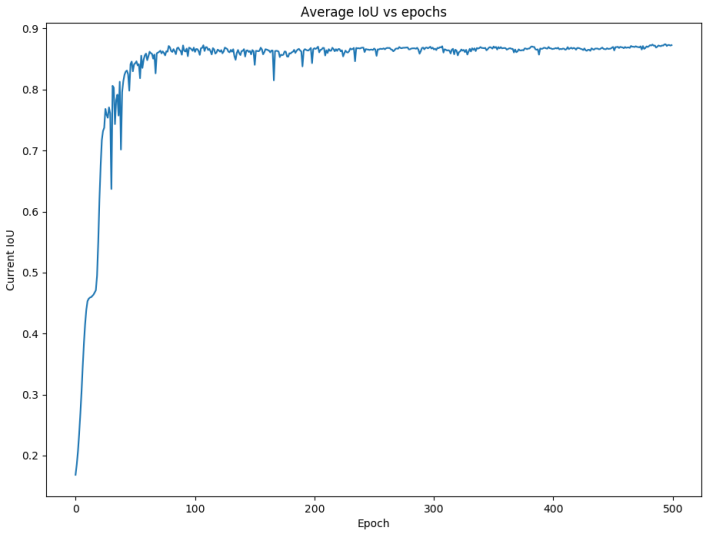

Figura 9.10: Average IOU vs epochs (TRY 2 Encoder-Decoder)

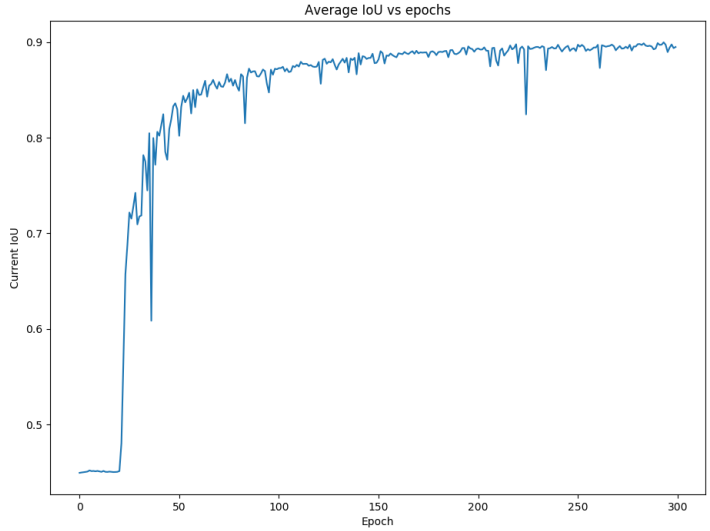

Figura 9.11: Average IOU vs epochs (TRY 3 Encoder-Decoder)

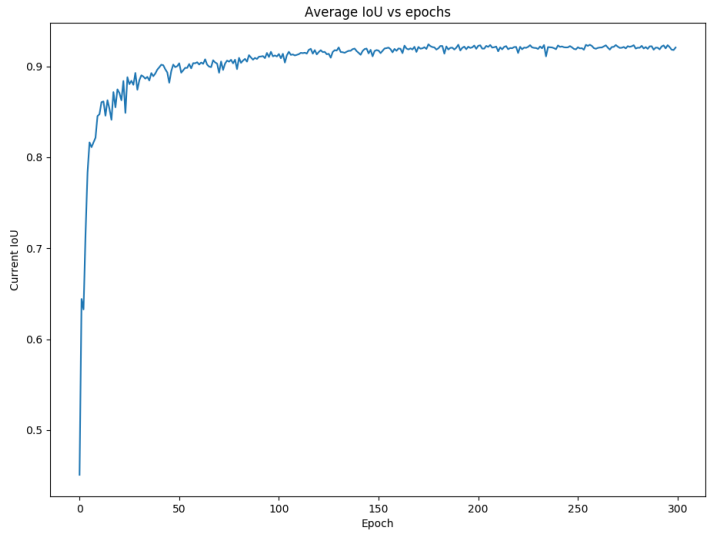

Figura 9.12: Average IOU vs epochs (TRY 4 Encoder-Decoder)

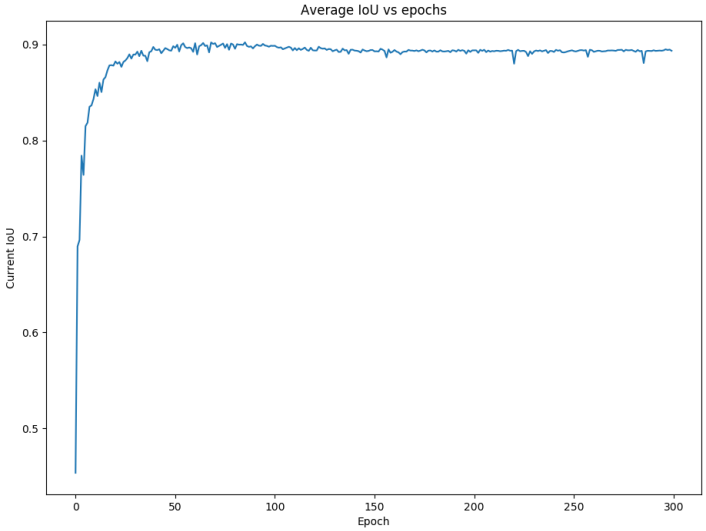

Figura 9.13: Average IOU vs epochs (TRY 5 Encoder-Decoder)

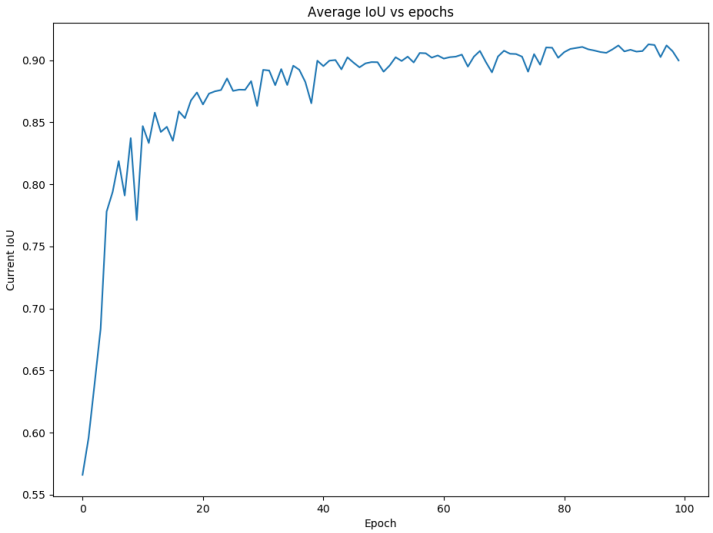

Figura 9.14: Average IOU vs epochs (TRY 6 Encoder-Decoder)

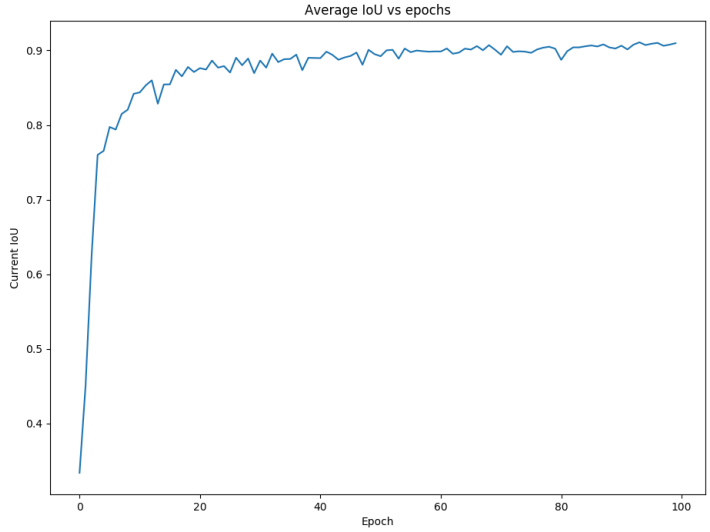

Figura 9.15: Average IOU vs epochs (TRY 7 Encoder-Decoder)

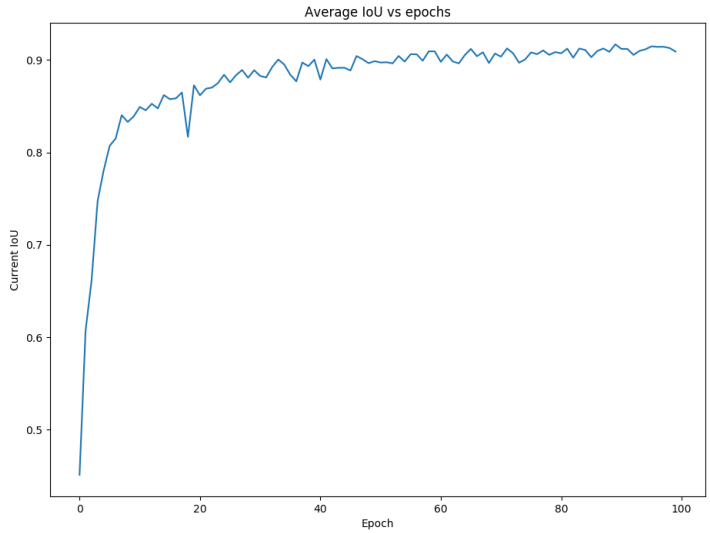

Figura 9.16: Average IOU vs epochs (TRY 8 Encoder-Decoder)

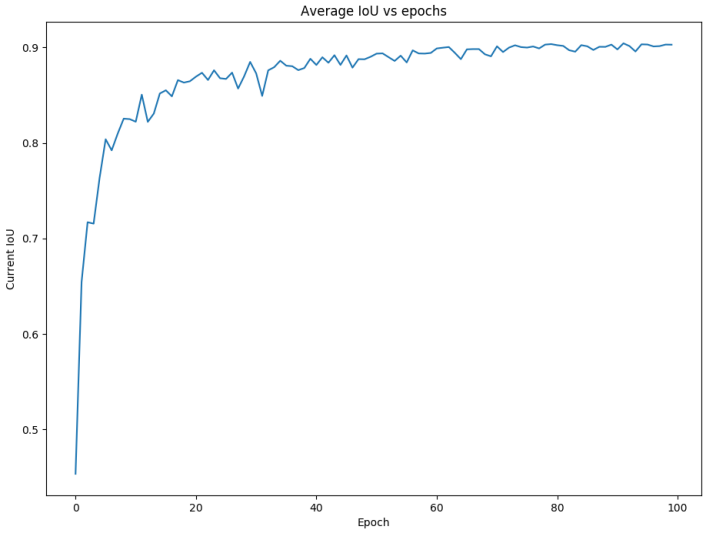

Figura 9.17: Average IOU vs epochs (TRY 9 Encoder-Decoder)

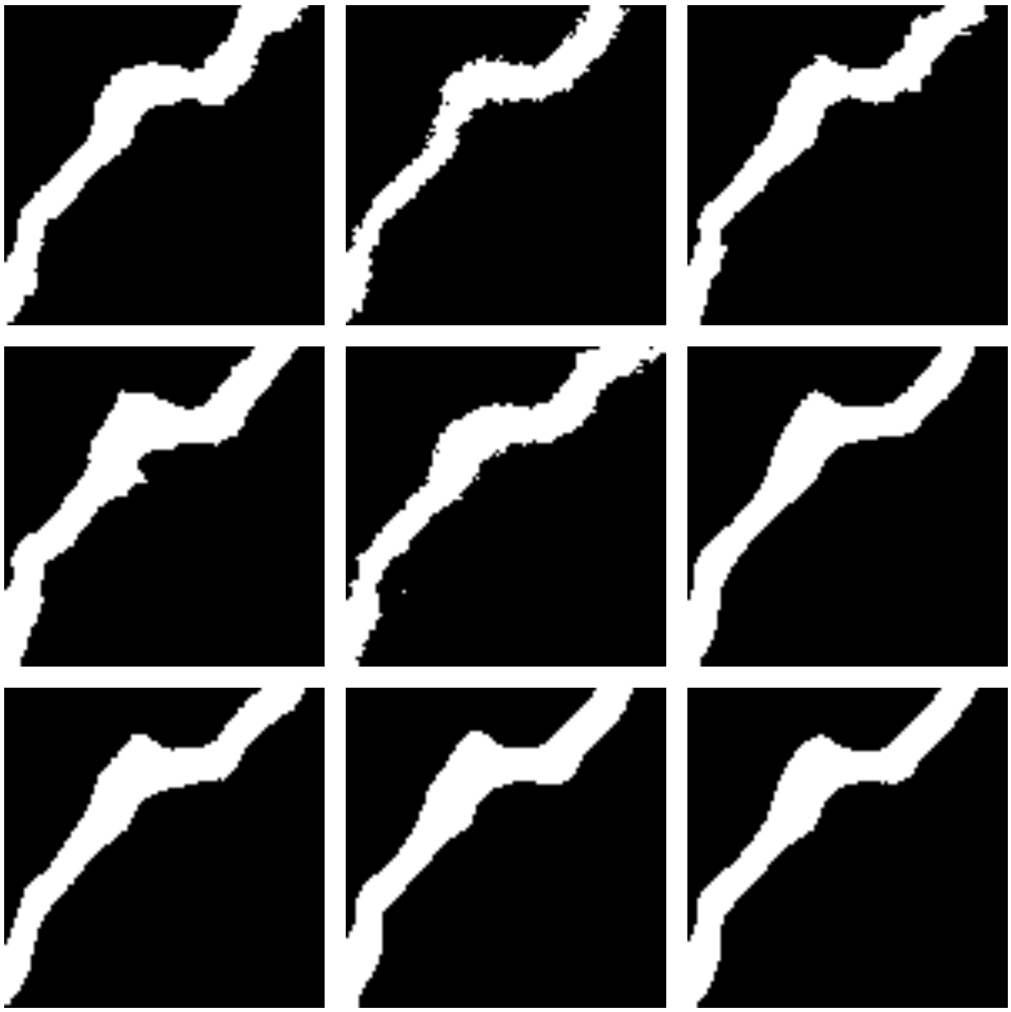

Figura 9.18: Predicción de imagen de grieta en binario (TRY 1 a TRY 9) Encoder-Decoder

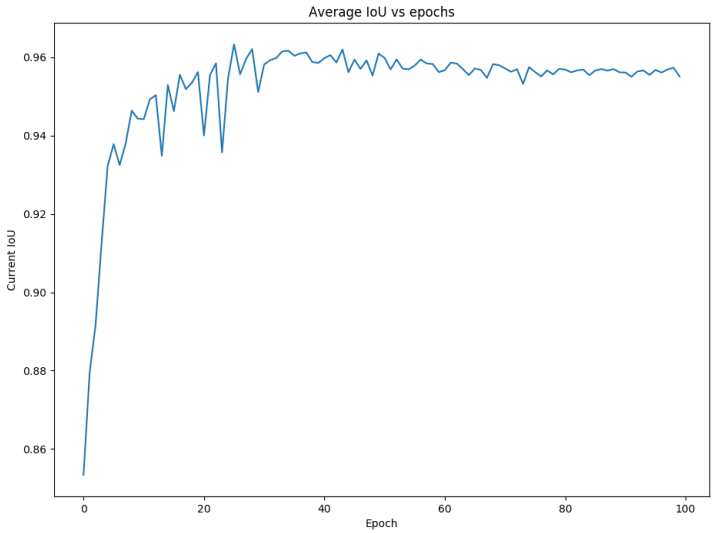

Figura 9.19: Average IOU vs epochs (TRY 1 Encoder-Decoder with skip)

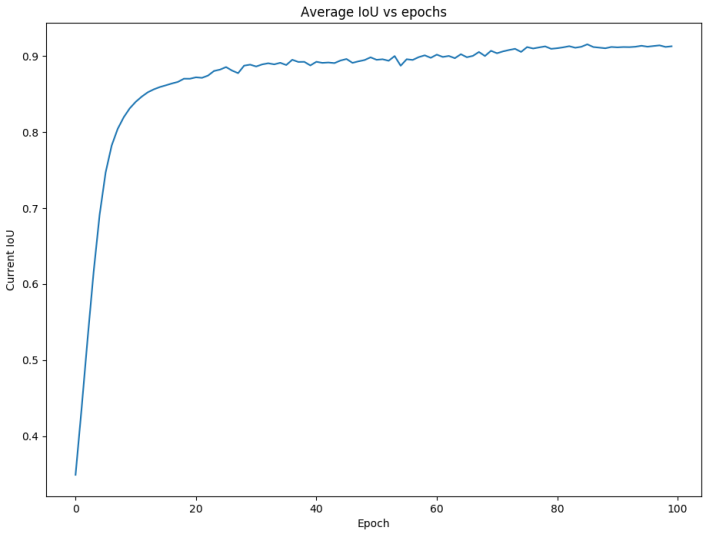

Figura 9.20: Average IOU vs epochs (TRY 2 Encoder-Decoder with skip)

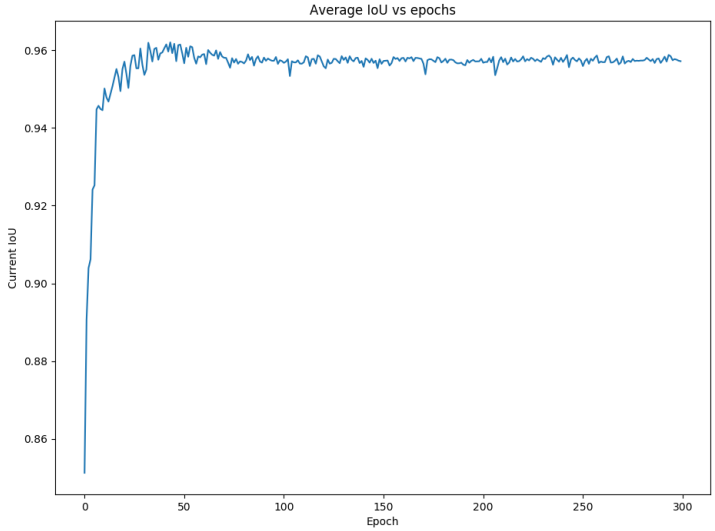

Figura 9.21: Average IOU vs epochs (TRY 3 Encoder-Decoder with skip)

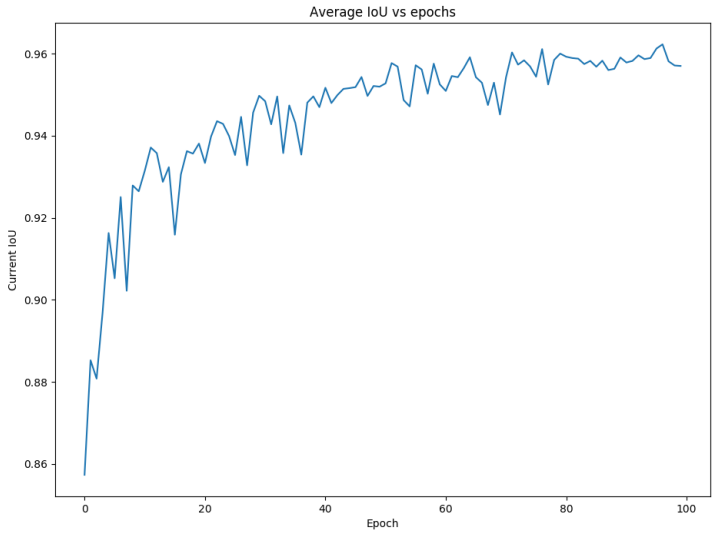

Figura 9.22: Average IOU vs epochs (TRY 4 Encoder-Decoder with skip)

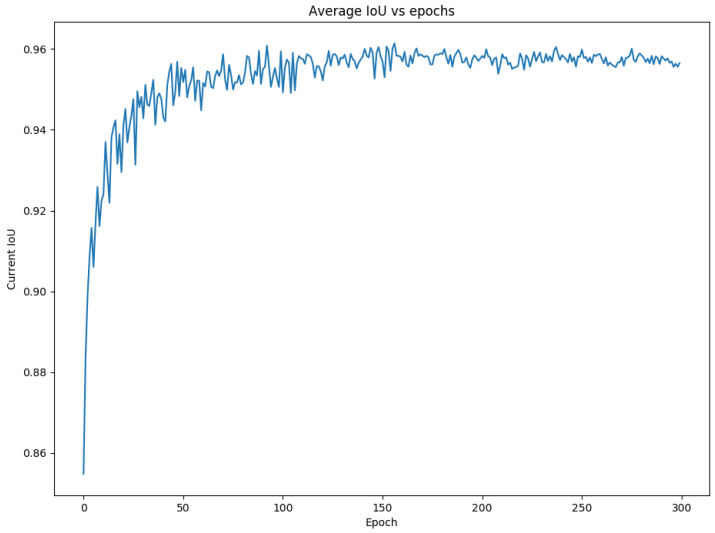

Figura 9.23: Average IOU vs epochs (TRY 5 Encoder-Decoder with skip)

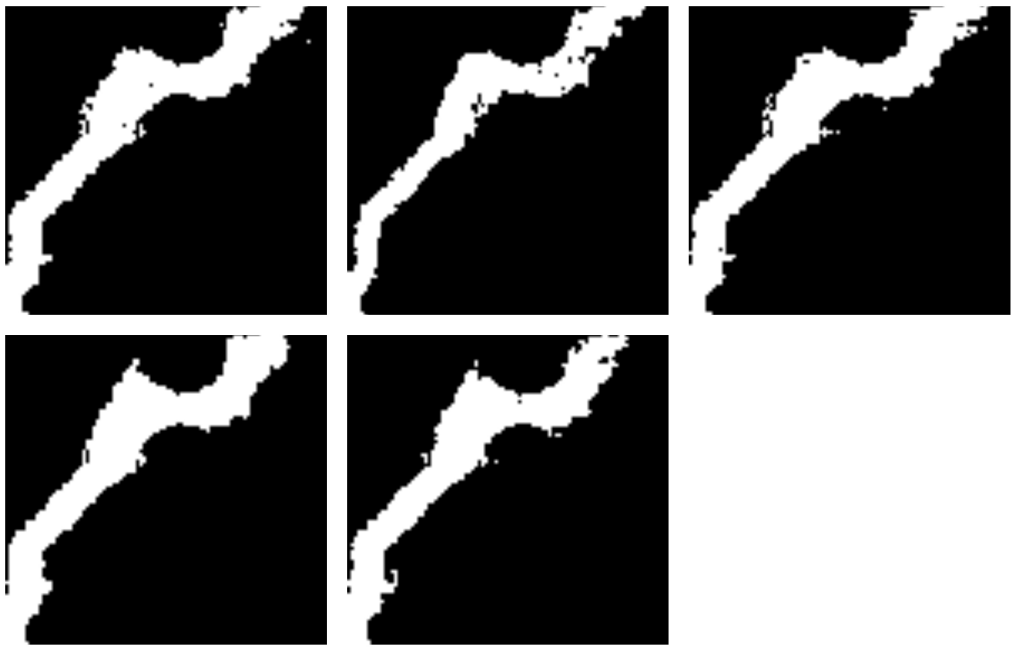

Figura 9.24: Predicción de imagen de grieta en binario (TRY 1 a TRY 5) Encoder-Decoder with skip