UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE UN CLASIFICADOR MEDIANTE REDES NEURONALES PARA UN SISTEMA DE INSPECCIÓN INDUSTRIAL 3D

AUTOR: PAU GARRIGUES CARBÓ

TUTOR: LUIS JOSÉ SAIZ ADALID

COTUTOR: ALBERTO JOSÉ PÉREZ JIMÉNEZ

Curso Académico: 2018-19

Resumen

En este proyecto se realiza el diseño e implementación de un clasificador basado en redes neuronales convolucionales con el objetivo de realizar el etiquetado de objetos 3D con el objetivo de alimentar un sistema de inspección industrial.

El sistema propuesto pretende superar las limitaciones de los clasificadores convencionales implementados actualmente en el sistema de inspección, donde un conjunto de objetos concretos pueden presentar una ambigüedad muy elevada debido al sistema de reconstrucción en tres dimensiones que se utiliza.

Para poder desarrollar el sistema se ha investigado la bibliografía existente con tal de encontrar una arquitectura de red neuronal orientada a la clasificación de objetos tridimensionales, ya sea mediante un sistema multivista o representaciones volumétricas basadas en nubes de puntos o en voxels.

Se diseña un clasificador de objetos 3D mediante un sistema multivista al que se le realiza una modificación para poder aprender la escala de los objetos a partir de la resolución de la imagen. Este clasificador se implementa utilizando diferentes arquitecturas para determinar cual es la más adecuada.

Finalmente se han realizado pruebas de funcionamiento generando un segundo dataset de piezas defectuosas deformando las piezas del dataset original, comprobando la respuesta del clasificador ante este tipo de entradas.

Palabras clave: aprendizaje automático, aprendizaje profundo, redes neuronales artificiales, visión artificial, inspección industrial.

Resum

En aquest projecte es realitza el disseny i implementació d'un classificador basat en xarxes neuronals convolucionals amb l'objectiu de realitzar l'etiquetatge d'objectes 3D amb l'objectiu d'alimentar un sistema d'inspecció industrial.

El sistema proposat pretén superar les limitacions dels classificadors convencionals implementats actualment en el sistema d'inspecció, on un conjunt d'objectes concrets poden presentar una ambigüitat molt elevada a causa del sistema de reconstrucció en tres dimensions que s'utilitza.

Per poder desenvolupar el sistema s'ha investigat la bibliografia existent per tal de trobar una arquitectura de xarxa neuronal orientada a la classificació d'objectes tridimensionals, ja sigui mitjançant un sistema multivista o representacions volumètriques basades en núvols de punts o en voxels.

Es dissenya un classificador d'objectes 3D mitjançant un sistema multivista al qual se li realitza una modificació per poder aprendre l'escala dels objectes a partir de la resolució de la imatge. Aquest classificador s'implementa utilitzant diferents arquitectures per determinar quina és la més adequada.

Finalment s'han realitzat proves de funcionament generant un segon dataset de peces defectuoses deformant les peces del dataset original, comprovant la resposta del classificador davant aquest tipus d'entrades.

Paraules clau: aprenentatge automátic, aprenentatge profund, reds neuronals, visió artificial, inspecció industrial

Abstract

In this project, the design and implementation of a classifier based on convolutional neural networks is carried out with the aim of carrying out the labeling of 3D objects in order to feed an industrial inspection system.

The proposed system aims to overcome the limitations of conventional classifiers currently implemented in the inspection system, where a set of concrete objects can present a very high ambiguity due to the three-dimensional reconstruction system that is used.

In order to develop the system, the existing bibliography has been investigated in order to find a neural network architecture oriented to the classification of three-dimensional objects, either through a multiview system or volumetric representations based on point clouds or voxels.

A 3D object classifier is designed by means of a multiview system that is modified to be able to learn the scale of the objects based on the resolution of the image. This classifier is implemented using different architectures to determine which is the most appropriate.

Finally, performance tests have been carried out generating a second dataset of defective parts, deforming the pieces of the original dataset, checking the response of the classifier to this type of inputs.

Keywords: machine learning, deep learning, artificial neural networks, computer vision, industrial inspection

Índice general

Parte I

Memoria

Capítulo 1

Introducción

1.1 Motivación

La creciente tendencia de la automatización de los procesos industriales ha impulsado el desarrollo de técnicas de inspección visual para tareas complejas, donde los métodos tradicionales serían demasiado costosos o incluso inefectivos.

Durante la realización del Máster en Ingeniería Industrial se ha cursado la asignatura optativa Visión Artificial, donde se introducen aspectos básicos de la tecnología de inspección por imagen, desarrollando una pequeña aplicación con métodos básicos. A partir de esta introducción a la inspección visual se desarrollaron una serie de conocimientos y capacidades para abordar estas tareas utilizando redes neuronales artificiales, en concreto las redes neuronales convolucionales. Este campo ha tenido una evolución muy rápida en los últimos años, y se está empezando a introducir en el ámbito industrial, ya sea realizando nuevas tareas o reemplazando otras tecnologías.

Este trabajo pretende realizar un estudio de la aplicación de la tecnología de redes neuronales artificiales aplicadas a la visión artificial dentro de un entorno industrial aplicado a la inspección visual de la producción.

1.2 Objetivos

El objetivo principal de este trabajo es diseñar y validar un clasificador automático mediante el uso de redes neuronales convolucionales, el cual sería posteriormente integrado en el dispositivo de inspección industrial ZeroGravity3D^{TM} desarrollado por el Instituto Tecnológico de Informática.

Para ello se deberán cumplir una serie de objetivos:

- Investigación bibliográfica sobre clasificadores basados en redes neuronales orientadas a la clasificación de objetos 3D.
- Formación en redes neuronales y entornos de programación para ello, implementación de diferentes arquitecturas.
- Entrenamiento de estas redes con un conjunto de datos significativos para la tarea designada.
- Comparación de los resultados obtenidos con otros métodos y obtención de conclusiones.

Capítulo 2

Contexto y estado del arte

2.1 ZeroGravity3D^{TM}

Este trabajo se encuentra enmarcado dentro del proyecto Zerogravity3D^{TM} (Figura 2.1) desarrollado por el Instituto Tecnológico de Informática, orientado a la inspección industrial mediante visión artificial. El funcionamiento del dispositivo se realiza en tres pasos: captura, reconstrucción y medición. El objetivo de este sistema es poder realizar una verificación de la producción completa en productos donde por su coste unitario, diseño, complejidad u otras razones, no es viable técnica o económicamente y es requerido por el cliente.

Figura 2.1: Dipositivo ZeroGravity3D^{TM} (ITI)

Figura 2.2: Ejemplo de objetos interesantes para ZeroGravity3D^{TM} (ITI)

El sistema de captura consta de 16 cámaras distribuidas espacialmente de manera optimizada para disminuir las ocultaciones y aumentar así la precisión y fiabilidad (Figura 2.3). La captura se realiza lanzando el objeto dentro del dispositivo desde abajo, y al alcanzar el zénit, cuando la velocidad es cero, se disparan las 16 cámaras simultáneamente, obteniendo una representación completa gracias a la ausencia de contacto.

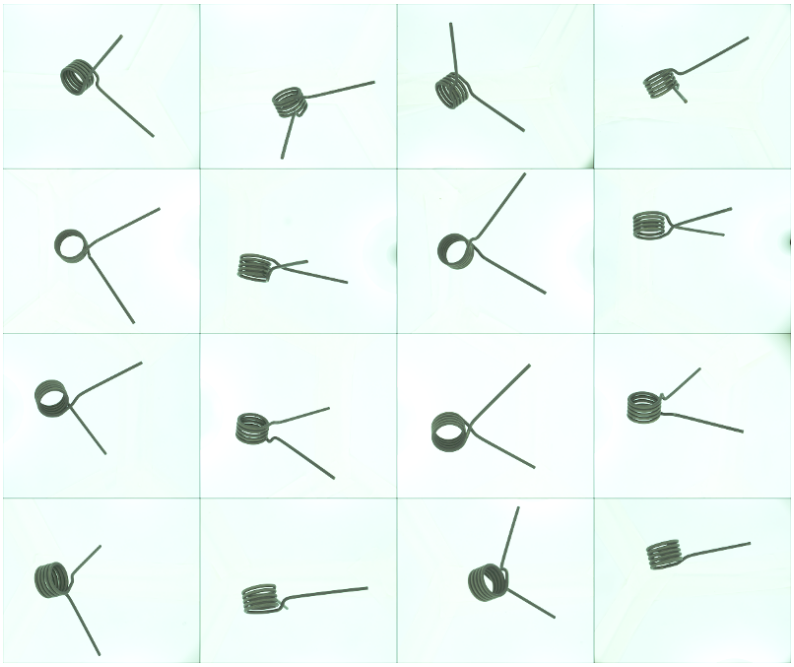


Figura 2.3: Capturas ZeroGravity3D^{TM} (ITI)

Seguidamente se realiza la reconstrucción del objeto en 3D a partir de las imágenes (Figura 2.4), utilizando para ello varias técnicas para prevenir la formación de abultamientos sintéticos en la reconstrucción debido a partes ocultas, ya sea por concavidades u oquedades en el objeto, o a la distribución de las cámaras (Perez-Cortes y col. 2018).

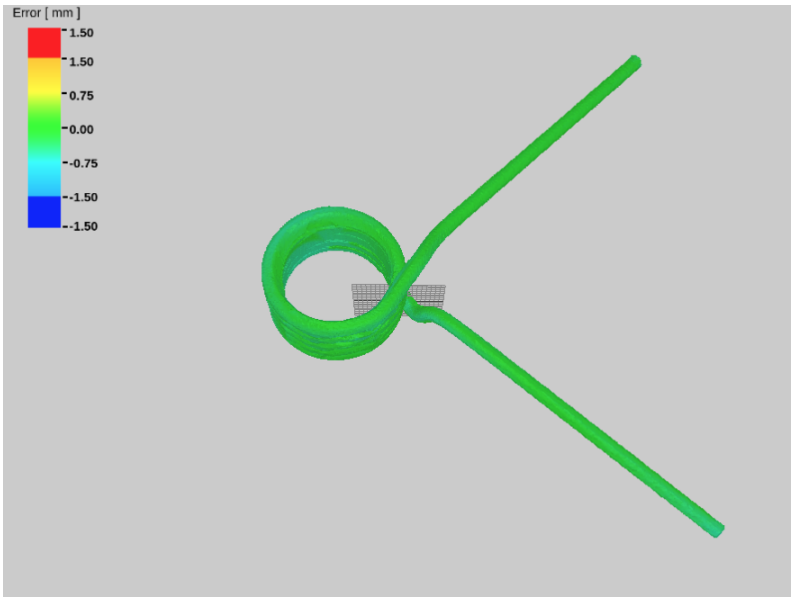


Figura 2.4: Reconstrucción ZeroGravity3D^{TM} (ITI)

Una vez reconstruido el modelo se realizan las tareas de inspección programadas para la pieza. Estas pueden ser tareas de análisis geométrico, metrología dimensional (GD&T) y análisis superficial incluyendo texturas, ya sea mediante especificaciones técnicas, o si no se dispone de ellas, se utilizan técnicas de aprendizaje automático a partir de muestras correctas de la pieza a inspeccionar.

El trabajo realizado en este TFM se implementaría entre la primera y la segunda etapa. Esta utilizaría las diferentes vistas capturadas para alimentar una red neuronal convolucional con el objetivo de clasificar las piezas introducidas para identificar y poder realizar una mejor reconstrucción y posterior análisis.

2.2 Aprendizaje Automático

El aprendizaje automático es el estudio de algoritmos y modelos estadísticos utilizados para mejorar progresivamente el desempeño de una tarea en un sistema computerizado. Es la rama de la inteligencia artificial basada en la idea de que los sistemas pueden aprender a partir de los datos a identificar patrones y poder tomar una decisión sin ser explícitamente programados para ello (Samuel 1959).

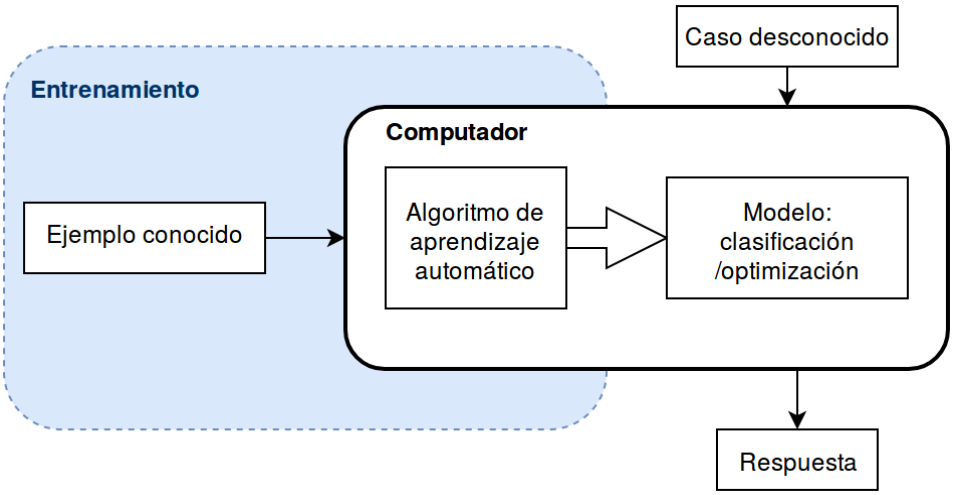


Figura 2.5: Esquema aprendizaje automático

El principio del aprendizaje automático se podría definir mediante la "Ley de Hebb" (Hebb 1949):

> "Se dice que un programa aprende de una experiencia **E** con respecto a la tarea **T** con rendimiento **R**, si su desempeño en la tarea T medida por R mejora la experiencia E."

Una tarea **T** describe cómo un sistema debe procesar un caso. El caso se puede representar como un vector $x \in R^n$, donde cada entrada x_i del vector es una característica, como por ejemplo los valores de cada píxel de una imagen.

Alguna de las tareas de aprendizaje automático más comunes son:

- Clasificación
- Regresión
- Eliminación de ruido
- Traducción
- Transcripción

La medida de rendimiento **R** suele ser específica para el tipo de tarea **T**. Por lo general el rendimiento se mide viendo el desempeño del algoritmo con datos no vistos anteriormente.

Las técnicas de aprendizaje automático se pueden dividir principalmente en dos grupos según el tipo de experiencia **E**: el aprendizaje supervisado, donde se decide una función a partir de unos datos de entrenamiento formados por pares de datos de entradas conocidas y salidas deseadas; y aprendizaje no supervisado, donde los objetos de entrada se tratan como un conjunto de variables aleatorias, construyendo un modelo de densidad para el conjunto de los datos.

Se pretende realizar una clasificación, donde el sistema intenta aprender a etiquetar una serie de vectores a una categoría concreta. Para esta tarea se van a utilizar redes neuronales convolucionales, que han demostrado un gran potencial para realizar tareas de clasificación a partir de imágenes.

2.3 Redes Neuronales

Una red neuronal artificial es un conjunto de elementos simples, unidades o nodos, cuya funcionalidad esta remotamente basada en el funcionamiento de las neuronas animales. La capacidad de procesamiento de la red reside en la fuerzas de conexión entre unidades, o peso, obtenido por un proceso de adaptación o aprendizaje sobre un conjunto de patrones de entrenamiento (Aleksander y Morton 1990).

Una neurona artificial imita a su contrapartida biológica mediante entradas y salidas, aunque su funcionamiento difiere completamente. La neurona artificial j se puede dividir en las siguientes partes

- Una función de entrada in_j
- Una función de activación f
- Un valor θ_j umbral
- Una función de salida f_{out}

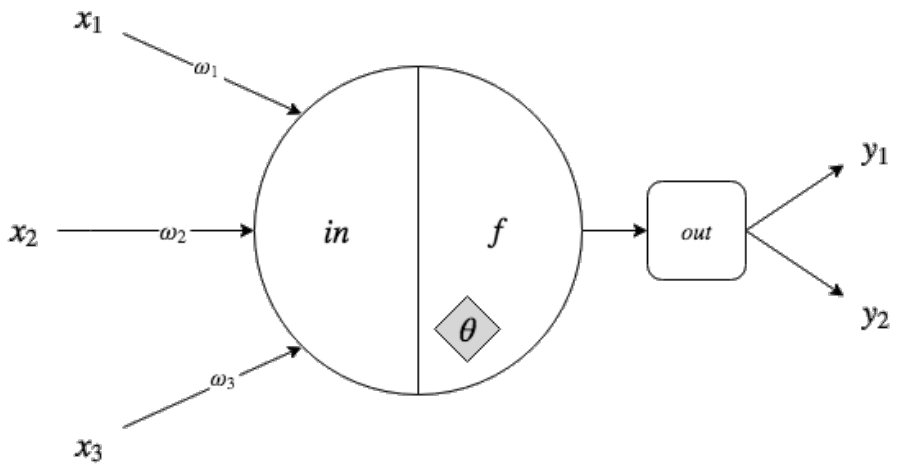


Figura 2.6: Esquema neurona artificial

A la función de entrada in toma las entradas pesadas $x_n\omega_n$ y realiza una operación para obtener una única entrada: estas operaciones pueden ser el sumatorio $\sum_{i=1}^{n} x_i\omega_i$, el productorio $\prod_{i=1}^{n} x_i\omega_i$ o bien el máximo de las entradas $Max(x_i\omega_i)$.

La función de activación f calcula el estado de actividad de la neurona a partir del resultado de la función de entrada menos el valor de umbral θ, obteniendo un estado de activación con un rango normalmente entre $(-1, 1)$ o $(0, 1)$. Algunos ejemplos de función de activación se pueden observar en la figura 2.7.

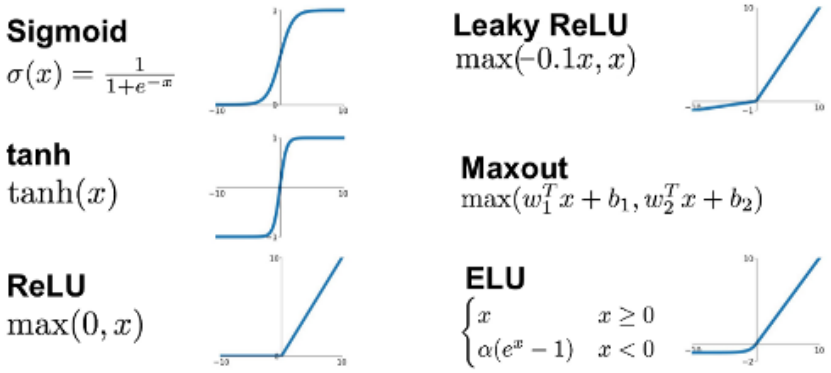


Figura 2.7: Funciones de activación

El último elemento es la función de salida *out*. Esta determina el valor de salida de la neurona que entrara en las siguientes. Habitualmente toma la función identidad, pero puede utilizarse en algunos casos para acotar la salida dentro de un rango de valores.

Las neuronas de entrada no tienen predecesoras, actuando como interfaz de entrada de la red. De la misma manera las neuronas de salida no tienen sucesoras, actuando de manera análoga como interfaz de salida de la red.

Las neuronas se distribuyen dentro de la red formando capas o niveles (Figura 2.8), con un número determinado de neuronas en cada capa. Estas capas se pueden clasificar en tres tipos:

- Entrada: esta capa recibe la información externa a la red.
- Ocultas: son las capas internas de la red y no tiene contacto con el entorno exterior.
- Salida: toma la información generada por las capas predecesoras y la transfiere al exterior de la red.

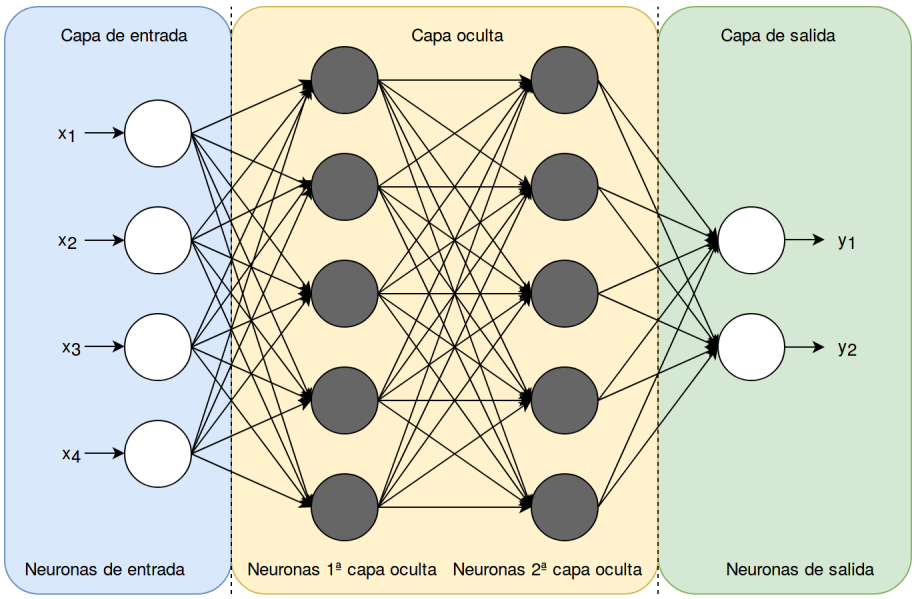


Figura 2.8: Esquema conexión red neuronal artificial

Durante el entrenamiento, la red aprende modificando los pesos de las conexiones entre las diferentes neuronas. Para el aprendizaje se define una función de costes $C : F \to \mathbb{R}$ de manera que para la solución óptima f^* tal que $C(f^*) \leq C(f) \forall f \in F$, es decir, ninguna solución puede tener un valor de coste superior a la solución óptima.

El proceso de aprendizaje se realiza en tres fases:

1. Forward progpagation: en esta fase se alimenta la capa de entrada con datos etiquetados y se recogen los resultados proporcionados por la capa de salida.
2. Calcular la función de costes: se calcula el error entre el valor proporcionado por la red y el valor deseado.
3. Backpropagation: a partir del resultado de la función de costes se ejecuta el algoritmo de backpropagation (Rumelhart, Geoffrey E. Hinton y Williams 1986) con el objetivo de determinar que neuronas han dado paso a una repuesta incorrecta y ajustar los pesos.

2.4 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (ConvNets o CNNs) son un tipo concreto de redes neuronales artificiales que han demostrado ser muy efectivas en tareas de visión artificial, ya qué tratan de imitar a las neuronas en la corteza visual de un cerebro biológico utilizando para ello el concepto de Neocognitron (Fukushima 1980). Estas redes son una variación del perceptrón multicapa diseñadas para precisar un preprocesamiento mínimo (LeCun y col. 1990).

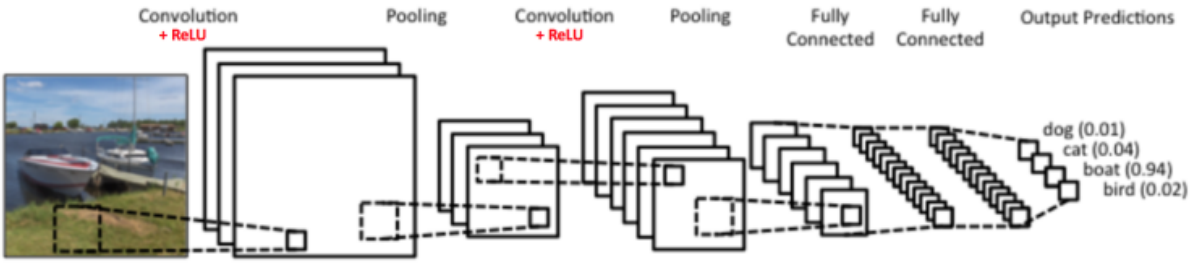


Figura 2.9: Ejemplo arquitectura CNN

Las capas de convolución sustituyen a las neuronas simples, realizando la operación descrita en la ecuación 2.1, donde la salida Y_j es la salida de la operación de convolución más una influencia b_j y pasada a través de una función de activación no-lineal g.

$$Y_j = g\left(b_j + \sum_i K_{ij} \otimes Y_i\right) \tag{2.1}$$

La operación de convolución se controla mediante tres parámetros:

- **Depth:** profundidad a la que se realiza la convolución. En imagen, el color se distribuye en tres capas, por lo que la profundidad será de 3. En escala de grises solo hay un canal, por lo que la profundidad será 1.
- **Stride:** número de píxeles de paso que se desplaza el *kernel* de convolución cada vez.
- **Zero-padding:** número de píxeles que me puedo "salir por el borde", rellenando de ceros.

La operación de convolución consigue filtrar la imagen de entrada con diferentes kernels K_{ij} destacando así diferentes aspectos de la imagen y obteniendo mapas de características, como bordes o elementos simples, que al ser aprendidos pueden aparecer en cualquier lugar de la imagen y ser identificados.

Se realiza generalmente la operación no lineal Rectified Linear Unit (ReLU) (Figura 2.7), mediante la cual aquellas zonas que tras la convolución dan como resultado un valor negativo se sustituyen por cero, consiguiendo así un resultado no-lineal a partir de una operación lineal como es la convolución. Además de la función ReLU se pueden utilizar otras, como la función sigmoidea o la tangente hiperbólica (tanh).

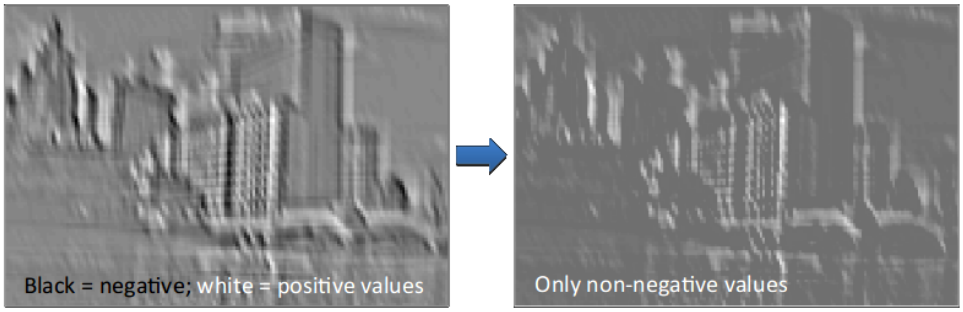


Figura 2.10: Efecto función ReLU (Fergus 2015)

Para reducir el tamaño de muestreo y hacer la red más robusta frente a pequeñas perturbaciones, se realiza una operación de *pooling*. Esta operación reduce la dimensión de los mapas de características manteniendo la mayoría de la información importante. El *pooling* se puede realizar de diferentes maneras, mediante la suma, el máximo, la media, etc.

En las redes neuronales convolucionales es habitual utilizar la operación de *max-pooling*, donde se define un tamaño de ventana y se asigna el valor máximo de los que se encuentren en su interior. Esta operación se realiza de manera independiente para cada mapa de características, por lo que para una imagen en color que genere tres capas, obtendremos tres salidas.

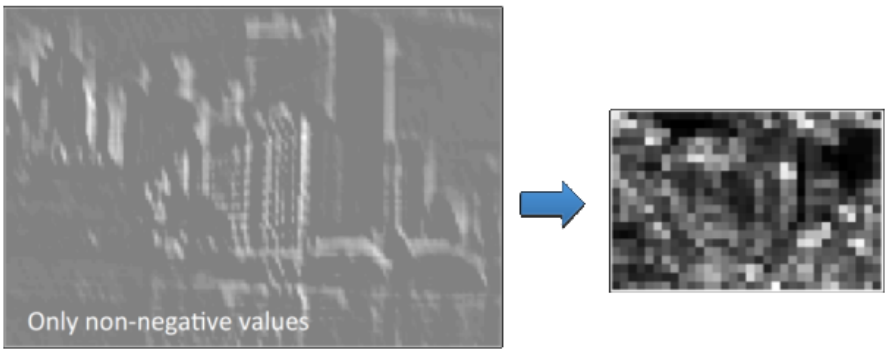


Figura 2.11: Efecto función MaxPool (Fergus 2015)

Además de reducir la dimensión, la operación de *pooling* añade invariabilidad ante pequeñas transformaciones en la imagen, ya que al agrupar los valores de cada zona, un pequeño cambio no provocará apenas cambios.

Las diferentes arquitecturas de redes neuronales convolucionales se construyen a partir de estos elementos, modificando las dimensiones, las cantidades y el orden de las diferentes capas, con el objetivo de obtener redes profundas más precisas y eficientes.

2.5 Procedimiento de aprendizaje

La clave del entrenamiento de redes neuronales es la función de pérdida o *loss function* (Goodfellow, Bengio y Courville 2016). Esta permite medir cómo de alejadas están las predicciones $\hat{y}$ de las verdaderas y. Durante el entrenamiento se minimiza este valor actualizando los parámetros ω con el objetivo de mejorar la precisión.

La función de pérdidas utilizada generalmente para el entrenamiento de CNNs es la *Cross Entropy Loss*, ya que es especialmente útil en problemas de clasificación con clases excluyentes. La función viene dada por:

$$H(y, \hat{y}) = \sum_x y_x \log \frac{1}{\widehat{y_x}} = -\sum_x y_x \log \widehat{y_x} \tag{2.2}$$

El proceso de entrenamiento se plantea como un problema de optimización, utilizando el método del descenso del gradiente estocástico o *stochastic gradient descent* (Mei, Montanari y Nguyen 2018), un método iterativo para optimizar una función objetivo diferenciable. Se denomina estocástico debido a que las muestras se seleccionan de manera aleatoria.

Dentro de las técnicas de descenso del gradiente estocástico utilizamos el algoritmo de optimización de Adam (contracción de *Adaptative Moment Estimation*) (Kingma y Ba 2014), descrito en el algoritmo 1.

Uno de los hiperparámetros de la red neuronal es la tasa de aprendizaje o *learning rate*. Mediante este parámetro el descenso del gradiente puede determinar el siguiente punto de la siguiente manera, siendo w_j un parámetro, f la función de perdidas a minimizar y α el *learning rate*.

$$w_j = w_j - \alpha \frac{\partial f(w_j)}{\partial w_j} \tag{2.3}$$

Si la tasa de aprendizaje es demasiado pequeña, el gradiente convergería muy lentamente, y si es demasiado grande, este podría no llegar a converger, e incluso divergir, como se puede observar en la figura 2.12.

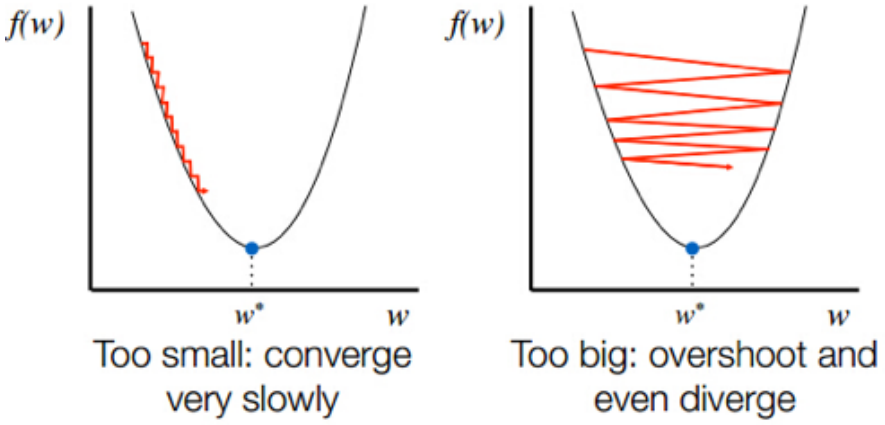


Figura 2.12: Efecto de diferentes tasas de aprendizaje

Algoritmo 1 Algoritmo optimizador Adam (Kingma y Ba 2014). g_t^2 indica el cuadrado elemento a elemento $g_t \odot g_t$. η es un pequeño escalar para evitar divisiones entre 0. Todas las operaciones con vectores son elemento a elemento. Parámetros por defecto que han demostrado ser efectivos $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\epsilon = 10^{-8}$.

Entrada: α: Tamaño de paso
Entrada: $\beta_1, \beta_2 \in [0, 1\}$:
Entrada: $f(\theta)$: Objetivo estocástico con parámetros θ
Entrada: θ_0: Vector de parámetros iniciales

$m_0 \leftarrow 0$ (Inicializar el primer vector de momentos)
$v_0 \leftarrow 0$ (Inicializar el segundo vector de momentos)
$t \leftarrow 0$ (Inicializar tiempo)
mientras θ_t no convergente **realizar**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Obtener los gradientes respecto a los objetivos estocásticos en el tiempo t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Actualizar la desviación de la estimación del primer momento)
$v_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Actualizar la desviación de la estimación del segundo momento)
$\widehat{m_t} \leftarrow m_t / (1 - \beta_1^t)$ (Calcular la estimación del primer momento con corrección de desviación)
$\widehat{v_t} \leftarrow v_t / (1 - \beta_2^t)$ (Calcular la estimación del segundo momento con corrección de desviación)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon)$ (Actualizar parámetros)
fin mientras
devolver θ_t: Parámetros Resultantes

El descenso del gradiente es un proceso iterativo donde los pesos de la red se calculan en cada iteración. Debido a esto es necesario procesar el *dataset* más de una vez. Para realizar esto se divide en partes llamadas *batch* de un determinado tamaño ya que el recurso memoria es determinante, y pasar el conjunto de datos completo no es posible. Un mayor tamaño del *batch* consigue converger en menos iteraciones. Cada vez que se procesa el *dataset* completo se denomina época (*epoch*). En la figura 2.13 se puede observar como para ajustar una función de dos parámetros, esta converge más rápidamente al punto de menor coste cuanto mayor es el tamaño del *batch*.

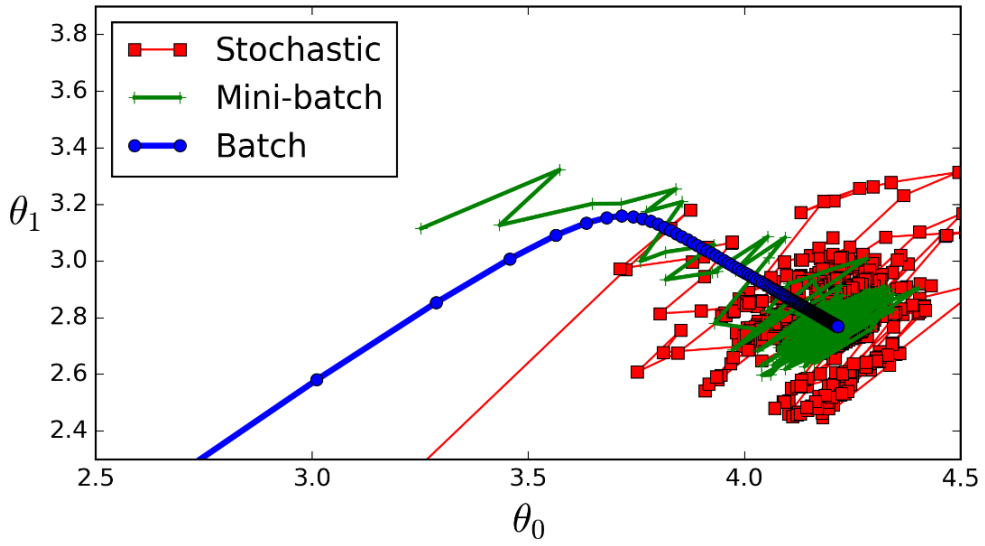


Figura 2.13: Efecto del *batch* en el descenso del gradiente

Para realizar un entrenamiento completo se realizan varias de estas épocas, al final de las cuales se ejecuta una etapa de validación, presentando a la red un conjunto de imágenes que no ha visto anteriormente y a partir de las cuales no aprenderá, y se calcula la función de perdidas *loss* y la precisión del ensayo. Si esta no mejora en cierto numero de épocas, el entrenamiento termina. Si continuara correríamos el riesgo de presentar sobreajuste en el modelo.

La salida de la red es un conjunto de puntuaciones asociadas a cada clase en números reales. Para facilitar la interpretación de estos resultados se suele utilizar la función SoftMax o función exponencial normalizada, donde se "comprime" el vector de valores reales $\mathbb{R}^K$ en un rango $[0, 1]$ y de suma unidad.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; para \; j = 1, ..., K \tag{2.4}$$

Gracias a esta función podemos interpretar la salida de la red como el índice de probabilidad de que un objeto pertenezca a una clase concreta según la red.

Capítulo 3

Materiales

3.1 Entorno de programación

Dentro del campo de la inteligencia artificial y el aprendizaje automático se han desarrollado multitud de software ya sea libre, como Torch, TensorFlow o Caffe; o propietario, como la *Math Kernel Library* de Intel o la *Neural Network Toolbox* de Matlab.

Los lenguajes de programación dominantes en el sector son Python y C++, utilizando en este caso Python en su versión 3.6 para el desarrollo de este trabajo. En este caso se ha trabajado con TensorFlow™, desarrollado por el equipo de Google Brain, y PyTorch™, apoyado principalmente por el grupo de inteligencia artifial de Facebook. Tanto TensorFlow como PyTorch tienen soporte para aceleración mediante GPU utilizando la tecnología CUDA® de NVIDIA®.

Además de las librerías para redes neuronales y aprendizaje automático, se han utilizado librerías auxiliares todas ellas libres, como NumPy para la computación de vectores y matrices, *Python Imaging Library* (PIL) para la carga y transformación de imágenes y OpenCV en su versión 3.4.4 para realizar operaciones más complejas con imágenes para las cuales PIL no es suficiente.

Todo el software se ha ejecutado sobre el sistema operativo basado en Linux Ubuntu 16.04.

3.2 Hardware

El Instituto Tecnológico de Informática ha proporcionado un ordenador con una CPU Intel® Core™ i7 920 @ 2.67GHz con 10GB de RAM y el sistema operativo libre Ubuntu 16.04. En este equipo se han realizado las tareas de programación, edición de imágenes y redacción de la memoria.

Para realizar el entrenamiento de la red neuronal se ha dado acceso remoto a un segundo equipo el cual posee una Unidad de Procesamiento Gráfico (GPU) NVIDIA® GTX 1080 con 8GB de memoria gráfica y tecnología de procesamiento en paralelo CUDA® mediante la cual el procedimiento de entrenamiento de las redes neuronales artificiales se ve acelerado.

3.3 Dataset

Una red neuronal artificial necesita de una gran cantidad de imágenes para poder realizar un entrenamiento correcto y garantizar un buen desempeño. Como trabajar con imágenes obtenidas a partir de lanzamientos de objetos reales en ZeroGravity3D^{TM} sería muy costoso en tiempo y limitaría la cantidad de objetos diferentes a clasificar, desde el ITI han desarrollado una herramienta para simular el lanzamiento de un objeto en el dispositivo a partir de un archivo CAD. Mediante esta herramienta podemos simular suficientes lanzamientos para obtener los diferentes ángulos de visión de cada objeto. Con este método se produce un sesgo, ya que no hay variabilidad en cada tipo de pieza, por lo que seguramente el modelo obtenido presente sobreajuste.

Para poner a prueba el funcionamiento de una red neuronal convolucional realizando la tarea de clasificación de los diferentes objetos, se han obtenido todos los modelos CAD 3D de la ferretería online `www.mootio-components.com`, correspondientes a los artículos tornillos, arandelas, casquillos, rodamientos, coronas y muelles. Tras revisar el dataset y eliminar algunos objetos con el CAD defectuoso y clases repetidas, en total se tienen 224 clases diferentes de objetos a clasificar.

Figura 3.1: Ejemplo de objetos a clasificar (mootio-components)

Categoría	Nº de Objetos
Tornillos	94
Arandelas	52
Casquillos	16
Coronas	4
Muelles	31
Rodamientos	26

Tabla 3.1: Objetos en cada superclase

El hecho de obtener el dataset de una ferretería es debido a que estos son tipo de productos para los cuales podría ser interesante integrar ZeroGravity3D^{TM} durante el proceso de producción, es decir, objetos de bajo valor unitario cuya verificación durante la producción representaría un coste inasumible por el fabricante.

A partir de los modelos descargados se simulan 110 tiradas de cada objeto, obteniendo las 16 vistas de cada tirada en formato JPEG. De las 101 tiradas se separan en 90 para el set de entrenamiento, 10 para el set de test y 10 para realizar la validación del algoritmo de etiquetado.

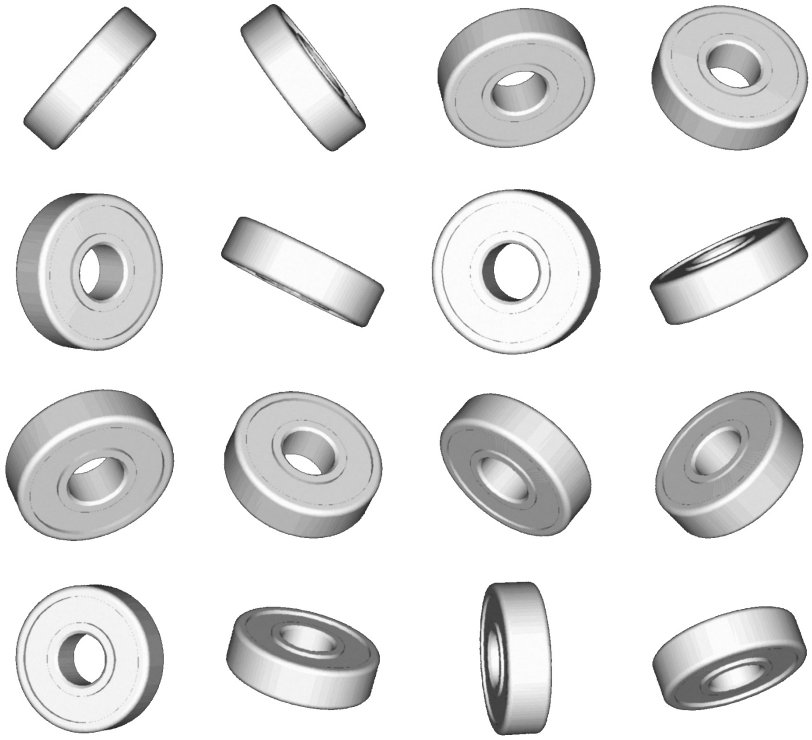


Figura 3.2: Mosaico vistas lanzamiento rodamiento 012873

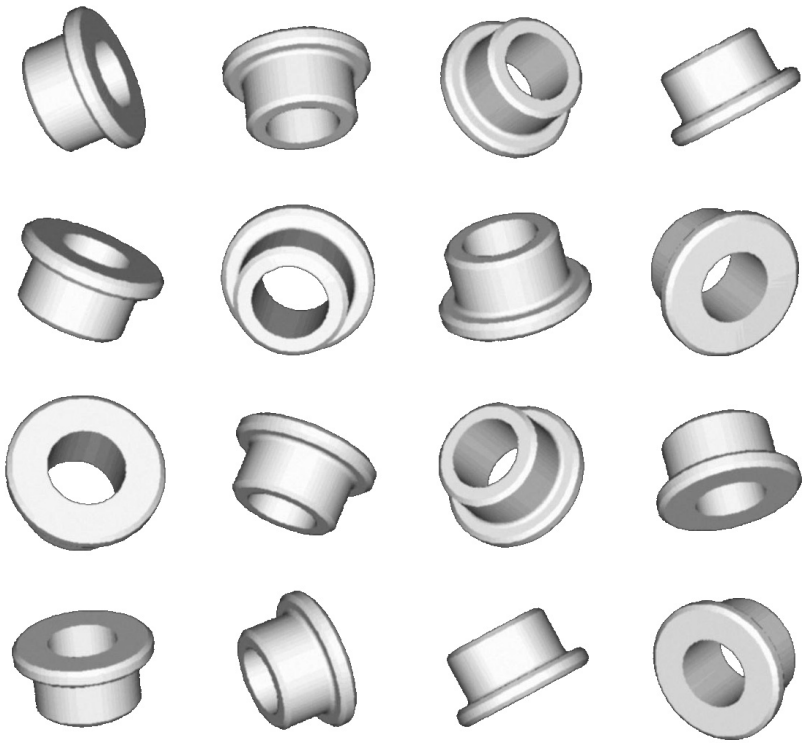


Figura 3.3: Mosaico vistas lanzamiento casquillo 013185

El dataset se organiza en carpetas, identificadas por el tipo de objeto y el número de serie de cada producto en la tienda. Durante el proceso de carga del dataset se tomará la etiqueta del objeto del nombre de cada carpeta. Estas carpetas dentro se dividen en tres subdirectorios, llamados train, test y eval, correspondientes a sendas actividades. Dentro de cada subdirectorio se encuentra de manera numerada cada lanzamiento simulado.

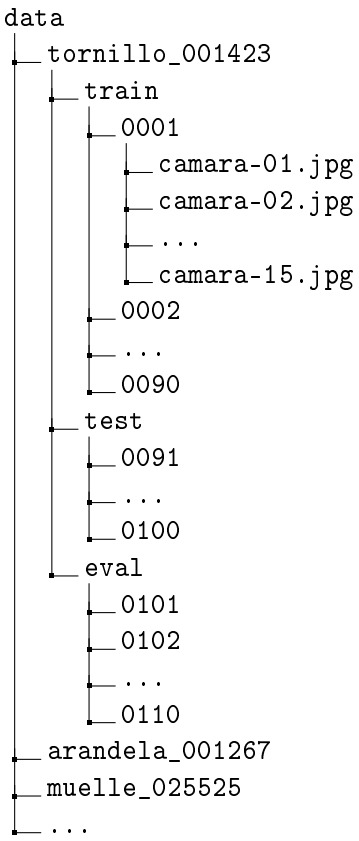


Figura 3.4: Estructura del dataset

Capítulo 4

Procedimiento

4.1 Aproximación a la clasificación mediante redes neuronales

Se propone una primera aproximación a la clasificación de imágenes utilizando deep learning mediante la técnica de *transfer learning*, permitiéndonos así utilizar el conocimiento aprendido en otro entorno para resolver un nuevo problema relacionado.

Para esta implementación se ha utilizado la arquitectura de red InceptionV3 (Szegedy, Vanhoucke y col. 2016), una versión mejorada de la arquitectura GoogLeNet (Szegedy, Liu y col. 2015), donde se introducen los módulos *Inception* (Figura 4.1). Estos módulos aplican 3 capas convolucionales junto con una capa de *max pooling* en paralelo, en lugar de en serie como se realiza habitualmente, reduciendo la profundidad de la red a la vez que se utilizan convoluciones de menor tamaño, siendo computacionalmente más eficiente y reduciendo el riesgo de producirse sobreajuste.

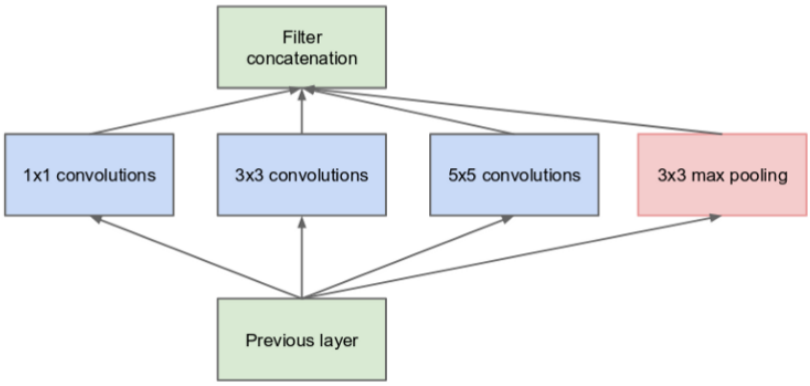


Figura 4.1: Módulo Inception sencillo (Szegedy, Liu y col. 2015)

Las mejoras implementadas en las siguientes versiones de Inception tratan factorizar las convoluciones para hacerlas todavía más pequeñas y expandirlas para evitar la perdida de información, logrando ensanchar los módulos sin hacer la red más profunda.

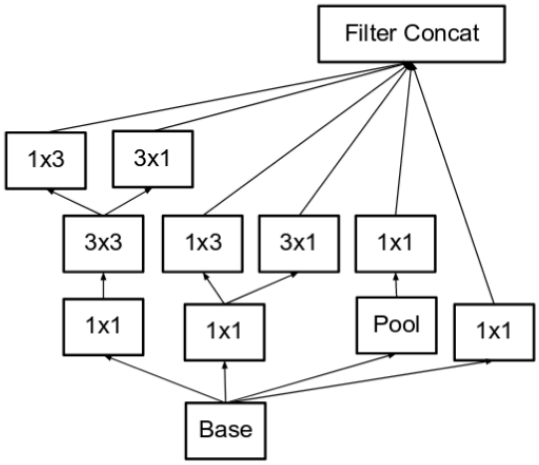


Figura 4.2: Módulo Inception2 (Szegedy, Liu y col. 2015)

Todas estas arquitecturas terminan con una serie de capas totalmente conectadas encargadas de realizar la tarea de clasificación. Al reentrenar la red, se elimina esta última capa del modelo Inception-v3 (Figura 4.3), pero mantenemos los pesos asignados a las características extraídas por las etapas anteriores, que están entrenadas con el dataset ImageNet. Este dataset es utilizado en el "ImageNet Large Scale Visual Recognition Challenge" (Russakovsky y col. 2015), el cual se compone de 150.000 fotografías etiquetadas manualmente en 1000 categorías, por lo que originalmente tenemos 1000 nodos de salida. Aparte de la competición, este dataset es ampliamente utilizado como benchmark para comparar distintas arquitecturas de redes.

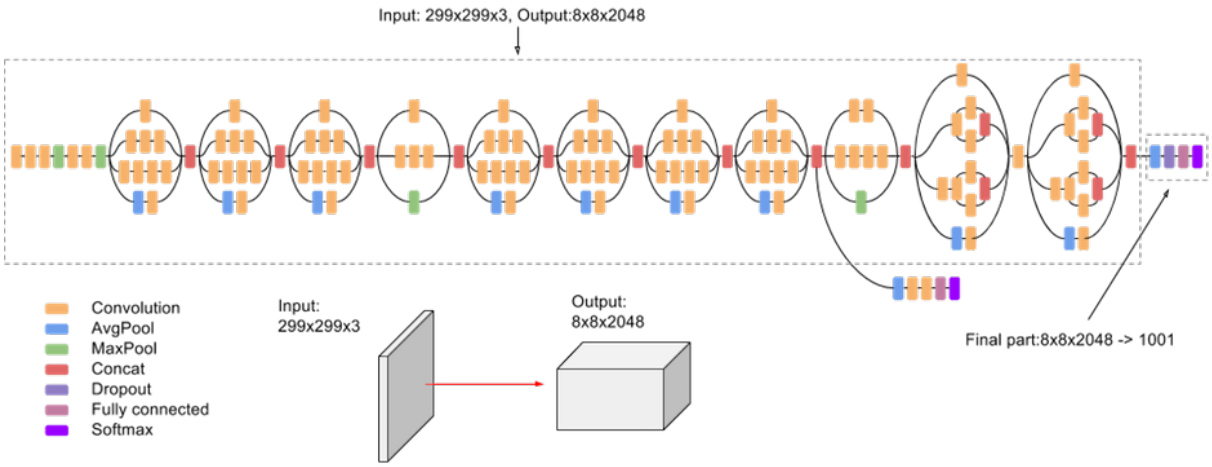
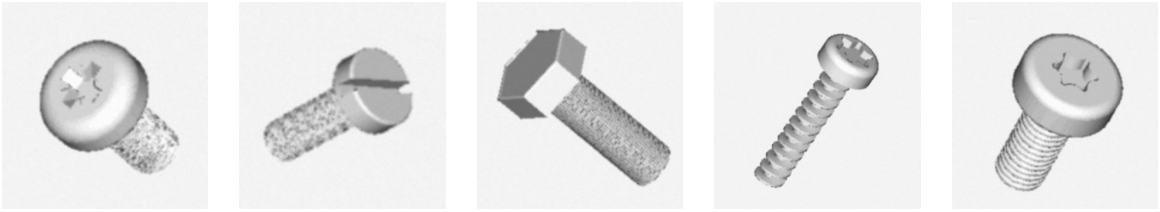


Figura 4.3: Grafo del modelo Inception V3 (Szegedy, Vanhoucke y col. 2016)

Con el objetivo de reentrenar la red para nuestro propósito, se modifica el tamaño de la capa de salida de la red, teniendo ahora solo los nodos de salida para nuestras categorías. Esta última capa es entrenada mediante algoritmos de *backpropagation*, utilizando la entropía cruzada como función de coste para ajustar los pesos de las salidas calculando el error entre la salida de la función softmax y el vector de etiquetas de la categoría de la muestra dada. Mediante este procedimiento podemos obtener resultados de manera mucho más rápida, aprovechando las capacidades de la red original.

Para realizar esta tarea se ha proporcionado un dataset formado por cinco tipos diferentes de tornillos. Este dataset reducido servirá para poder realizar pruebas ligeras que no necesiten mucho tiempo de computo con el objetivo de testear la arquitectura y familiarizarse con el entorno de programación y a las redes neuronales convolucionales.



001423 002802 003970 012816 022929

Figura 4.4: Tornillos dataset reducido (recortados)

Tras reentrenar la red se ha testeado utilizando para ello imágenes obtenidas a través del modelo CAD, variando los ángulos de las vistas y los fondos de imagen. Se pasan a la red 16 imágenes individualmente, simulando las imágenes obtenidas por el dispositivo.

La clasificación del objeto se realiza pasando cada vista independientemente por la red y almacenando el resultado obtenido en cada vista. Para determinar a que clase pertenece el objeto se utiliza un sistema de votaciones utilizando la salida normalizada mediante la función *SoftMax*, obteniendo así un índice de probabilidad de pertenencia a una clase entre 0 y 1. El sistema de votación asigna puntos de la siguiente manera: se suma un punto si se ha asignado una probabilidad de pertenencia a una clase mayor a 0.75, 0.6 puntos si esta se encuentra en el rango de 0.6 a 0.75, y finalmente si existe una diferencia menor de 0.15 y el mayor esta por encima de una probabilidad de 0.5, se asignan 0.5 puntos la primero y 0.4 puntos al segundo. Además se ha establecido como condición de aceptación que la puntuación obtenida finalmente tenga que ser al menos un tercio de la puntuación máxima posible, para así descartar objetos de los que no se ha obtenido suficiente información, ya sea porque las vistas obtenidas no sean representativas o que el objeto no es conocido.

- Índice mayor a 0.75 → 1 punto
- Índice en rango 0.6 a 0.75 → 0.6 puntos
- Índice mayor a 0.5 y diferencia con el segundo menor a 0.15:
  - Primero → 0.5 puntos
  - Segundo → 0.4 puntos

El modelo es reentrenado con el set de imágenes original, teniendo entonces 80 imágenes de cada tipo de tornillo disponibles para las tareas de entrenamiento y validación de la red neuronal. Con este primer entrenamiento, se ha testeado con diferentes vistas de los objetos obtenidas a partir del modelo CAD en 3D, obteniendo los resultados de la siguiente tabla.

Tipo de pieza	**Puntuación**	**Resultado**
001423	6.1/16	Correcto
002802	5.6/16	Incorrecto (022929)
003970	10.9/16	Correcto
012816	9.6/16	Correcto
022929	12.7/16	Correcto

Tabla 4.1: Resultados primer ensayo

Como se puede observar en la tabla 4.1, de las cinco clases de tornillos a clasificar se obtiene un resultado positivo en 5 de ellas. Estos resultados son bastante positivos teniendo en cuenta el limitado tamaño del dataset.

Se realizan estas pruebas como primera aproximación a la clasificación de objetos mediante redes neuronales, comprobando que con un planteamiento básico se obtienen resultados prometedores. El siguiente paso será implementar una arquitectura de red que considere simultáneamente todas las vistas obtenidas para un objeto.

4.2 Red neuronal convolucional multivista

La principal desventaja del planteamiento anterior es que al tratar cada vista por separado, aquellas que no contienen información relevante pesan igual que aquellas más determinantes para la clasificación.

Tras consultar la bibliografía disponible, una de las estructuras que plantean el reconocimiento de figuras 3D a partir de imágenes 2D es la arquitectura textitmulti-view CNN (Su y col. 2015). El concepto de base encaja perfectamente en el funcionamiento de ZeroGravity3D y sus 16 cámaras.

Aunque se han diseñado arquitecturas de red capaces de realizar operaciones de clasificación de objetos 3D a partir de representaciones como una nube de puntos u objetos formados a partir de voxels (el equivalente al píxel en espacio 3D), se ha optado por la arquitectura MVCNN ya que al trabajar con imágenes se conservan detalles del objeto que de otra manera se perderían en el proceso de reconstrucción, como ya se ha mencionado anteriormente.

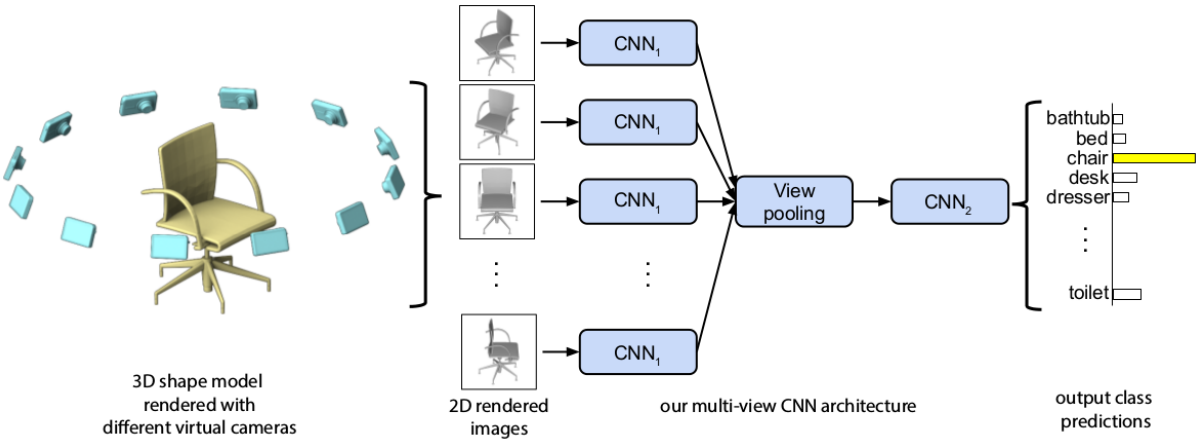


Figura 4.5: Arquitectura MVCNN (Su y col. 2015)

Cada imagen de la representación multivista del objeto 3D es pasada a través de la primera parte de la red (CNN_1) por separado, uniéndolas mediante la capa de agrupación de vistas (*view-pooling*) y enviando la información por la segunda parte de la red (CNN_2) donde se realiza la clasificación.

Este planteamiento utiliza arquitecturas de red conocidas y trabaja con ellas dividiéndolas en la parte de extracción de características y la de clasificación. Para la estructura de la red neuronal convolucional de partida se ha utilizado como base AlexNet (Krizhevsky, Sutskever y Geoffrey E Hinton 2012) (Fig 4.6), ya que es una de las arquitecturas más sencillas que ha demostrado ser eficiente en tareas de clasificación de imágenes. Esta red consta de 2 capas convolucionales y pooling mapeando las imágenes de entrada de 224x224 a mapas de características de 11x11. La arquitectura sigue con una secuencia de 3 capas convolucionales que de manera eficiente aplica una capa convolucional de kernel 9x9, aun con un mayor grado de no linealidad. La secuencia de capas convolucionales es seguida con una capa *pooling* y 3 capas totalmente conectadas, de las cuales la última tendrá tantas salidas como categorías a clasificar.

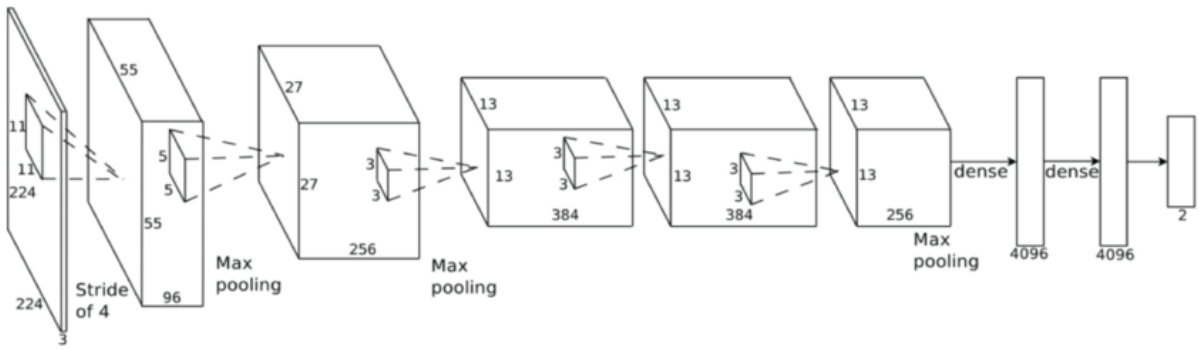


Figura 4.6: Estructura AlexNet (Krizhevsky, Sutskever y Geoffrey E Hinton 2012)

AlexNet utiliza la función ReLu (Rectified Linear Unit) como función de activación entre neuronas. Además se coloca una capa de Dropout entre las capas totalmente conectadas. Esta capa de

Dropout tiene probabilidad p asociada, la cual aplica individualmente a cada neurona, anulando la activación de manera aleatoria con probabilidad p. Se aplica este método con el objetivo de ayudar a evitar el sobreajuste de nuestro modelo a los datos de entrenamiento (Srivastava y col. 2014).

La adaptación realizada a AlexNet para utilizarla en la estructura MVCNN ha sido cortar la red tras la ultima etapa de *max pooling* antes de entrar a las capas totalmente conectadas. Este primera parte se repite para cada imagen de entrada, correspondiendo con la CNN_1 de la figura 4.5. Esta parte se genera de manera recursiva para cada imagen de entrada, haciendo el sistema flexible a diferente número de cámaras. Las salidas de esta etapa se introducen en la operación *view-pooling*, donde extraemos las características con más peso de cada imagen, dando así un tensor del mismo tamaño que cada una de las salidas. Con las características obtenidas se alimentan las 3 capas totalmente conectadas y se realiza una operación *SoftMax* a la salida para obtener los índices de probabilidad normalizados.

Otro aspecto importante es el tiempo necesario para realizar el entrenamiento. Las primeras pruebas usando la CPU (i7-920 @ 2.67GHz) emplearon 20 horas para realizar 1000 pasos de entrenamiento y generar un primer checkpoint. Este checkpoint puede utilizarse para proseguir el entrenamiento o como modelo en sí. Está pendiente repetir este experimento usando una GPU Nvidia modelo GTX1080, que gracias a la tecnología CUDA rebajará los tiempos sensiblemente (una estimación 10-20 veces más velocidad de entrenamiento).

Se ha implementado el modelo en PyTorch, una librería open-source para *machine learning* basada en Troch implementada en Python, ya que presenta una sintaxis de creación del modelo más transparente, por lo que presenta ventajas a la hora de modificar la estructura de la red y poder acceder a las variables durante la ejecución, muy útil para tareas de debug y comprender mejor el funcionamiento de la red. Existe la posibilidad de transformar el modelo resultante a otros formatos mediante el uso de ONXX.

En PyTorch el modelo se define como una clase en Python, donde los atributos se corresponden a las diferentes capas de la red. Definiendo un método *forward*, se pasan las entradas a la red, se realizan las llamadas a las diferentes capas y se obtiene la salida.

Para el entrenamiento de esta red se ha proporcionado un nuevo dataset mucho más amplio que el anterior, con 95 tipos de tornillos diferentes, con los cuales se han realizado 101 simulaciones de lanzamientos para cada uno, obteniendo cada vez 16 imágenes con una resolución de 2448x2048 píxeles en formato *.jpg*. De estos lanzamientos se utilizarán los 90 primeros para entrenar la red, mientras que los 10 restantes se emplearan en el testeo y un último para validar.

Aunque sería interesante preprocesar la imagen para recortar la zona del objeto y maximizar el área ocupada por este, se realizan los primeros ensayos sin ningún pretratamiento, ya que una de las ventajas de las redes neuronales convolucionales frente a otros métodos clásicos de clasificación es la capacidad de extraer características de las imágenes en crudo.

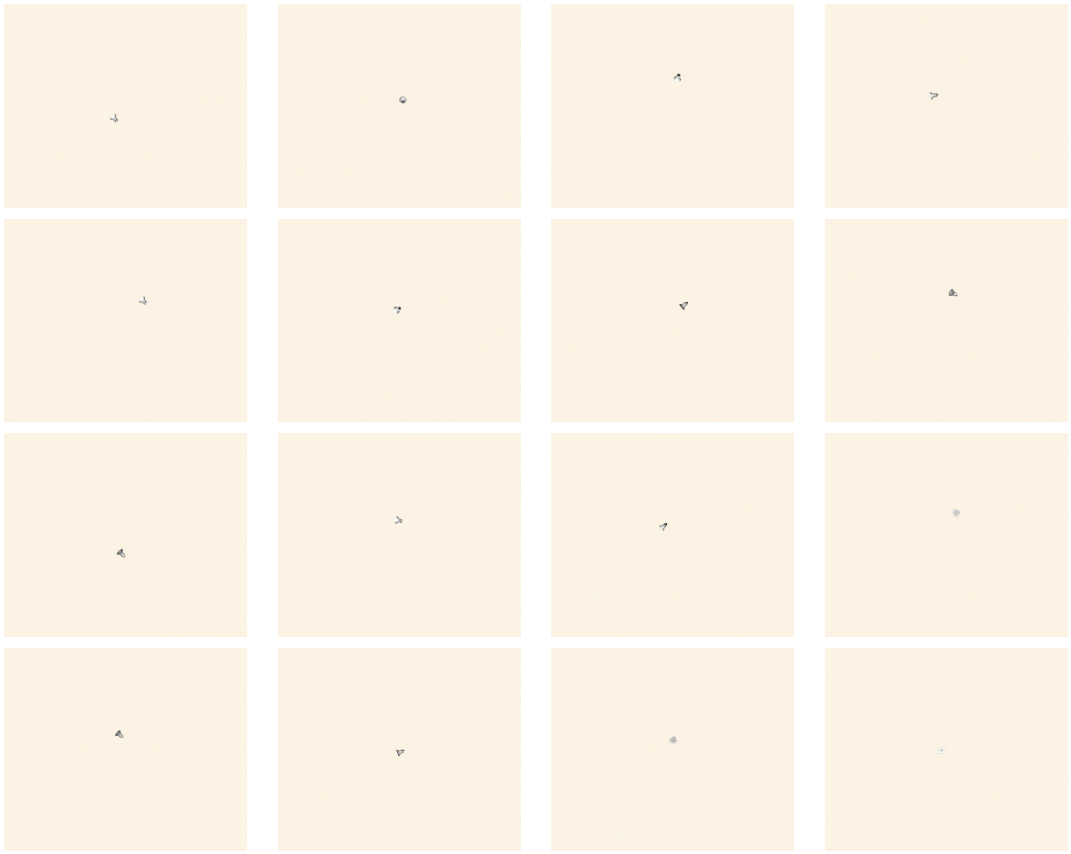


Figura 4.7: Vistas de ZG3D del modelo de tornillo 000058

Se realiza un primer ensayo con el script en PyTorch cargando las imágenes de solo 6 categorías en escala de grises, de entre las cuales se han incluido algunas de las señaladas como "imposibles de diferenciar" mediante el sistema de clasificación utilizado anteriormente, realizando un recorte central cuadrado de 500 píxeles de lado y posteriormente un escalado a 224x224 píxeles para adaptar la imagen a la entrada de AlexNet.

Cada vez que termina una época se realiza una etapa de validación sobre el modelo entrenado, guardando un checkpoint si este mejora la precisión sobre la iteración anterior. Ejecutando en CPU cada época tarda en finalizarse entre 30 y 50 minutos. El objetivo de este ensayo es probar si la red es capaz de extraer características con entradas adversas, donde el objeto apenas se diferencia del fondo y en algunos casos puede aparecer cortado. Tras pasar las primeras épocas, la función de costes empieza a minimizarse rápidamente, alcanzando una precisión del 100 % sobre el set de validación en la época 21. A partir de este punto el entrenamiento se detiene y el checkpoint se puede utilizar como modelo para realizar predicciones.

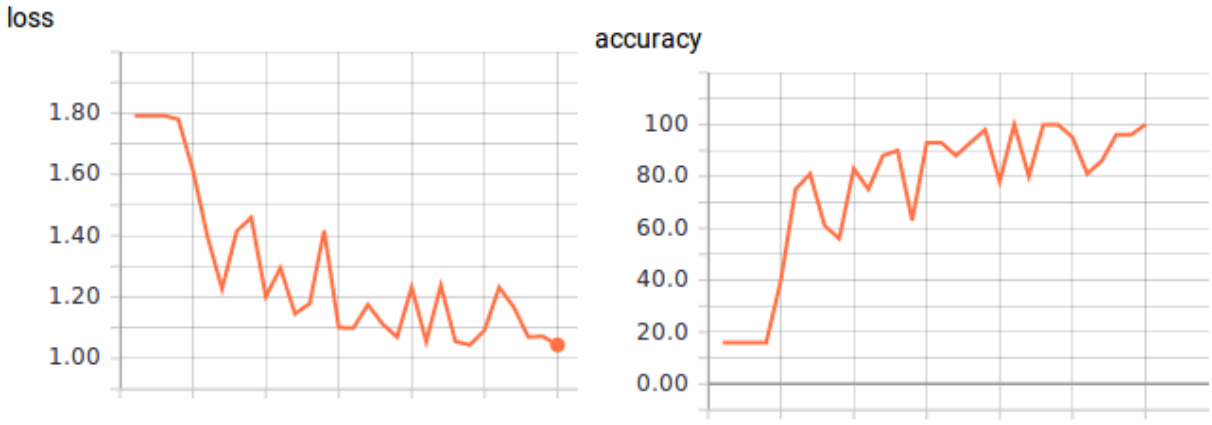


Figura 4.8: Función de costes y precisión del Ensayo 1

Un segundo ensayo es realizado, esta vez cortando 2000 píxeles de centro, quedándonos con la casi totalidad de la imagen original, y luego redimensionando a los 224x224 píxeles de las entradas. A partir de este punto se deja de utilizar TensorBoard como herramienta para almacenar y visualizar la evolución del entrenamiento. En su lugar se escriben los datos en un fichero .csv para su posterior análisis y visualización. Se ha ejecutado una época de este ensayo en GPU, presentando una velocidad 4 veces mayor frente a CPU, de 60 minutos a 15 minutos por época. En 30 épocas el valor de la función de costes no ha cambiado, por lo que la red no es capaz de extraer características diferenciantes para un objeto tan pequeño, por lo que habrá que preprocesar las imágenes.

4.3 Preprocesado del dataset

En el dataset los tornillos representan una pequeña área dentro de la imagen, como ya se ha podido observar en la figura 4.7, por lo que si se redimensiona la imagen completa a la resolución de entrada de la red, se pierden detalles del objeto, como por ejemplo la cabeza del tornillo, siendo esta una característica muy distintiva. Si se cambia la resolución de entrada a la red hay que tener en cuenta el consumo de memoria, ya que no debemos olvidar que se trabaja con 16 imágenes simultáneamente y se agota rápidamente la memoria del equipo. Un batch de menor tamaño implica una fluctuación mayor en la minimización del gradiente. Convertir las imágenes de entrada a escala de grises reduce sustancialmente el uso de memoria, pero no afecta tanto al tiempo de ejecución.

Para realizar el preprocesado se han utilizado las herramientas proporcionados por OpenCV con el objetivo de realizar una segmentación y poder detectar el área ocupada por el objeto (región de interés, ROI) para obtener un área cuadrada de recorte que maximice el área ocupada por el tornillo. Interesa que este recorte sea cuadrado para evitar deformaciones al redimensionar a la resolución de entrada de la red neuronal, las cuales suelen ser de formato cuadrado.

El procedimiento (Figura 4.9) se basa en realizar un binarizado,realizar una dilatación para eliminar las "motas" que quedan sobretodo en la zona de la rosca y se rellenan los huecos para obtener un único objeto binarizado en la imagen. Tras esto se pasa el algoritmo `connectedComponentsWithStats`, el cual lista los objetos encontrados mediante 8-conectividad, y se obtiene una matriz de características donde se encuentran las coordenadas del objeto, con las cuales podemos obtener una máscara cuadrada para el recorte. Además también obtenemos la matriz de centroides, permitiéndonos así realizar el recorte con el objeto centrado en la imagen. Este procedimiento es realizado con las diferentes tiradas de un mismo objeto para encontrar el recorte máximo que englobe a todas las imágenes.

Figura 4.9: Secuencia de preprocesado (los pasos intermedios se muestran ya recortados por claridad)

Este procedimiento se aplica previamente al entrenamiento de la red, cargando las imágenes ya recortadas, leyendo la resolución original e introduciéndola posteriormente en la red para conservar el parámetro de escala. En el algoritmo de clasificación esto se realizará en el procedimiento de carga de imágenes.

Se puede utilizar la resolución de la imagen como equivalente al tamaño ya que las cámaras a partir de la cual se obtiene siempre están a la misma distancia, por lo que al capturar varios lanzamientos se obtiene una muestra representativa de la distancia media a la que se encontrará el objeto.

Figura 4.10: Vistas de ZG3D del modelo de tornillo 000013 tras preprocesado

Como se puede ver en la figura 4.10, ahora el objeto es claramente visible en la imagen al contrario que en la figura 4.7. Este preprocesado además presenta la ventaja de aligerar el dataset, ocupando menos en disco duro (de 54.6 GB a 2.6 GB) y realizando el procedimiento de carga de imágenes más rápido.

4.4 MVCNN sensible a escala

Una vez realizado el preprocesado del dataset, se plantea una modificación a la estructura de la red para aprovechar el parámetro resolución de la imagen como dato de escala del objeto. Este parámetro se almacena durante la carga del dataset para luego poder acceder a él durante las fases de entrenamiento y evaluación.

La modificación en la arquitectura de la red se basa en añadir una neurona de entrada adicional en la primera capa de la fase totalmente conectada. A esta neurona se le pasa el valor de la resolución de la imagen antes de realizar el escalado, de esta manera la red obtiene un dato mediante el cual podrá aprender que objetos de mayor tamaño tienen una resolución mayor, pudiendo diferenciar objetos iguales en forma pero de diferente tamaño. La equivalencia entre resolución y escala es posible ya que el entorno es siempre el mismo para todas las imágenes.

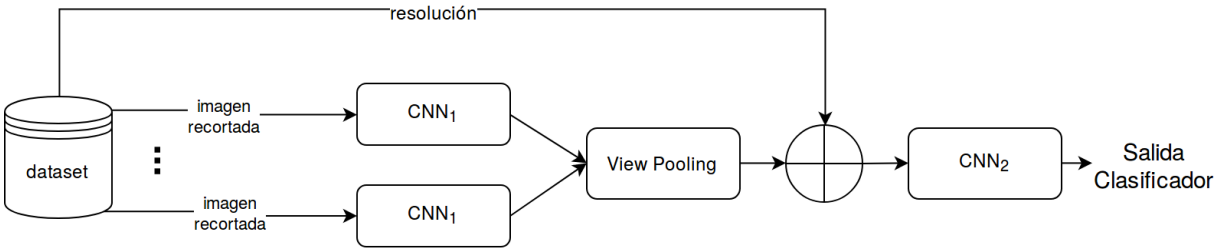


Figura 4.11: Arquitectura MVCNN sensible a escala

Con esta modificación se realiza un ensayo previo con las 6 clases mencionada anteriormente, obteniendo una tasa de acierto del 100 % sobre el set de validación en la época 23 tras 4 horas de entrenamiento con GPU. La mejora de tiempos frente a CPU se ven afectadas por los tiempos de carga de las imágenes a través de la red local. La evolución de la función de costes y la tasa de aciertos de la validación se pueden observar en la figura 4.12. Se verifica además que el modelo es invariante al orden en el que se presentan las imágenes, ya que se obtienen los mismos valores de salida de la función SoftMax para diferentes ordenaciones.

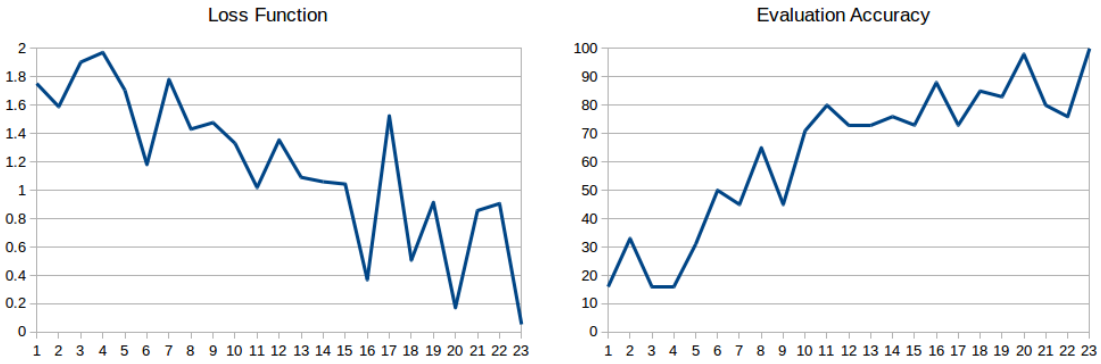


Figura 4.12: Función de coste y precisión del ensayo 6 clases con escala

Al construirse la red de forma recursiva para cada imagen, y mediante la función *view-pooling* siempre tenemos la misma dimensión de entrada a la capa completamente conectada, el algoritmo de clasificación puede evaluar el conjunto con diferente número de imágenes, por lo que el sistema podría seguir funcionando en el caso de que fallara alguna cámara.

Tras verificar que todo funciona correctamente se lanza el script para entrenar con el dataset completo de 95 clases. Se fija un tamaño de batch de 16 y se inicia el learning rate a 0.0001. Cada época necesita 2.5 horas para completarse en GPU. Al ser ahora el dataset más ligero, se copian las imágenes en el disco duro del ordenador con GPU, ya que se detecta que la lectura de estas por red es un cuello de botella que no permite exprimir al máximo la potencia de cálculo, pasando de 2 horas y media a tan solo **4 minutos por época**, notando ahora los beneficios de realizar el entrenamiento de redes neuronales en GPU (una mejora frente a CPU de x10).

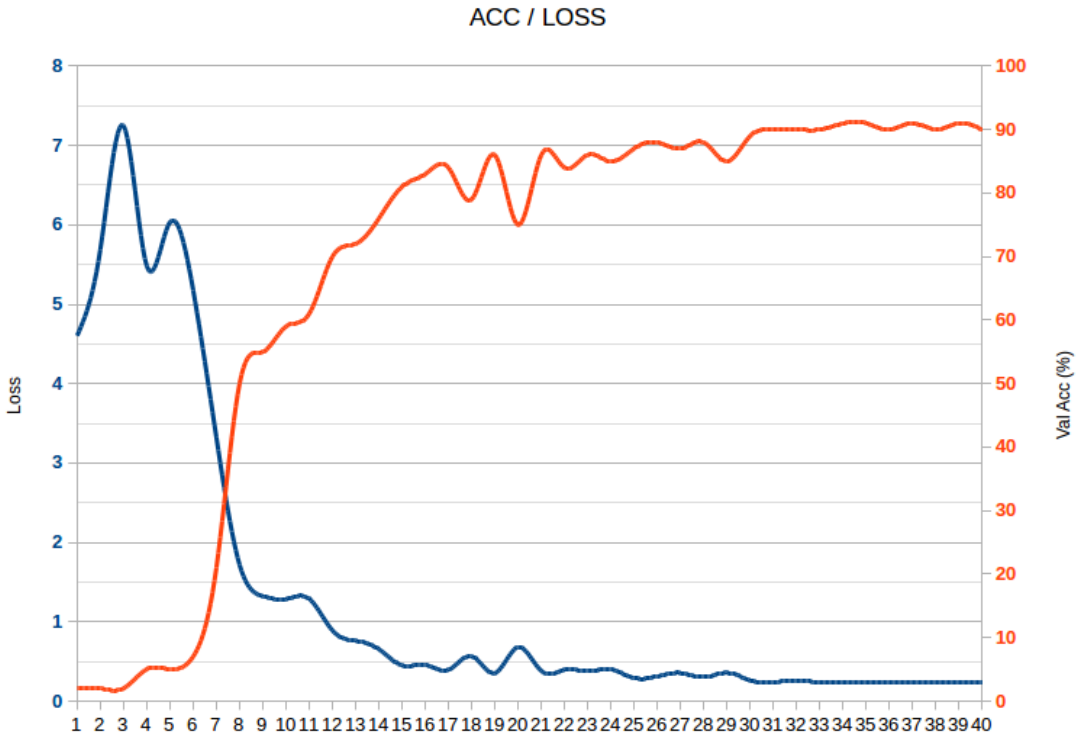


Figura 4.13: Función de coste y precisión del ensayo 95 clases con escala

En un principio parece ser que el entrenamiento se queda atascado en una precisión en la etapa de validación del 90 %.

Se elabora una matriz de confusión para detectar que clases están fallando. Una matriz de confusión es una matriz nxn en la que las filas se nombran según las clases reales y las columnas según las clases previstas por el modelo, pudiendo detectar de manera rápida que clase es confundida por otra.

Mediante la matriz de confusión (Figura 4.14) se detectan que clases están siendo imposible clasificar, revisando así el dataset y viendo que estas clases han tenido un preprocesado erróneo y las imágenes han quedado corrompidas. Se propone una mejora en el preprocesado para evitar esta situación.

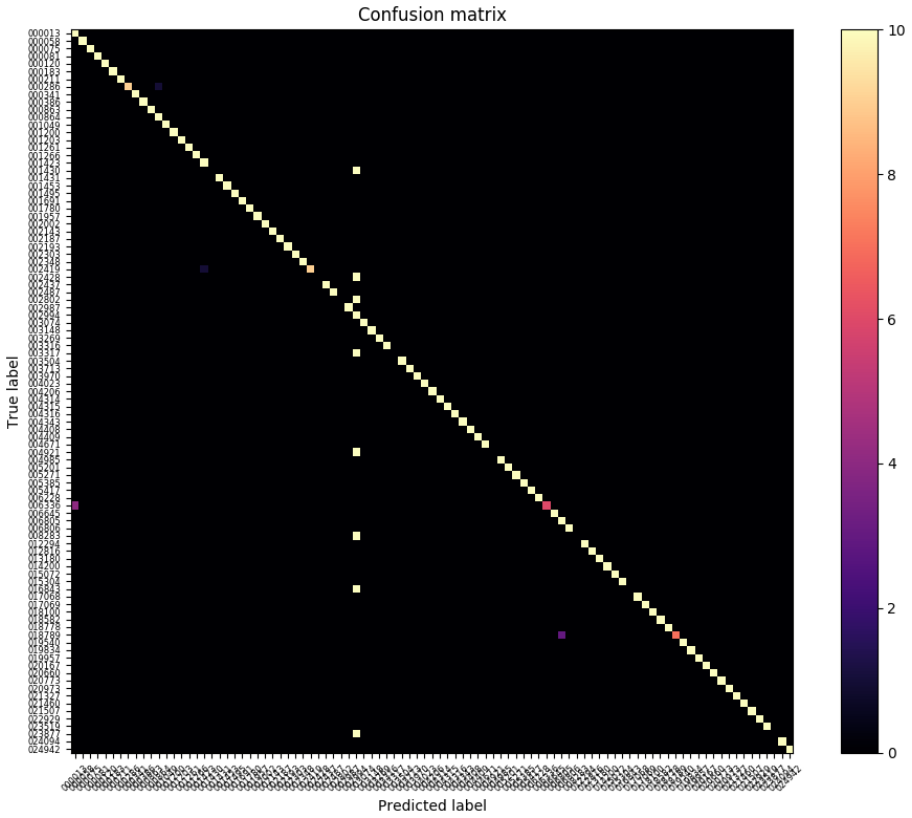


Figura 4.14: Matriz de confusión obtenida en el periodo de evaluación de la época 41

Se da un máximo de 2047 píxeles para el recorte (uno menos que la altura de la imagen original), por lo que valores por encima de este umbral se omitirán, previniendo que casos como el mostrado en la figura 4.15, donde el reflejo de la luz sobre la cabeza del tornillo hace que se vuelva casi indistinguible del fondo, por lo que el binarizado no se realiza correctamente y tras la dilatación toda la imagen queda como un único objeto, dando una medida errónea para toda la clase. Si se da este caso, se supondrá el objeto en el centro de la imagen y se realizará ahí el recorte a la resolución asignada por las otras vistas del objeto.

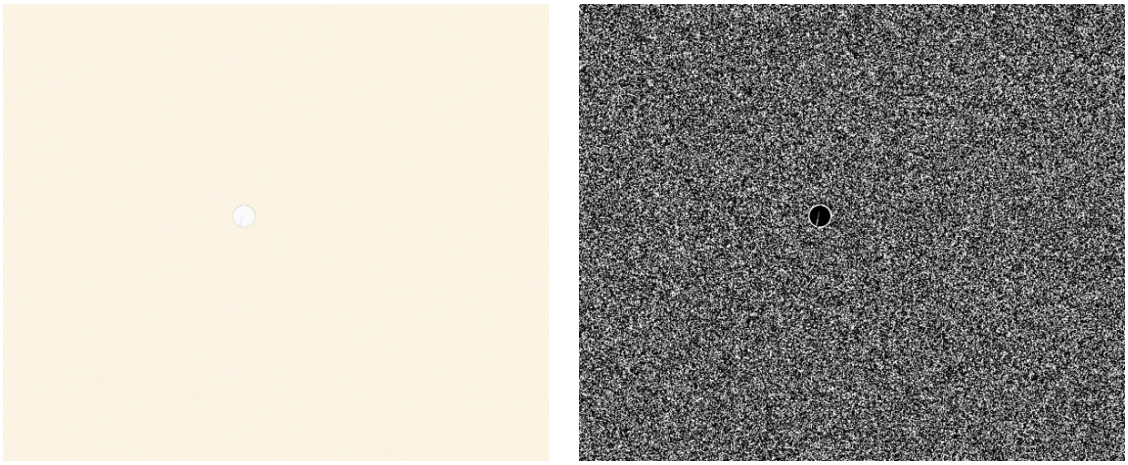


Imagen original Imagen binarizada

Figura 4.15: Fuente de errores en el preprocesado

Una vez solucionado este problema, volvemos a lanzar el entrenamiento, pero esta vez con un batch size de 32. En la época 89 se alcanza una precisión del 99 % (Figura 4.16), se mantiene el entrenamiento hasta la época 200 sin alcanzar el 100 %.

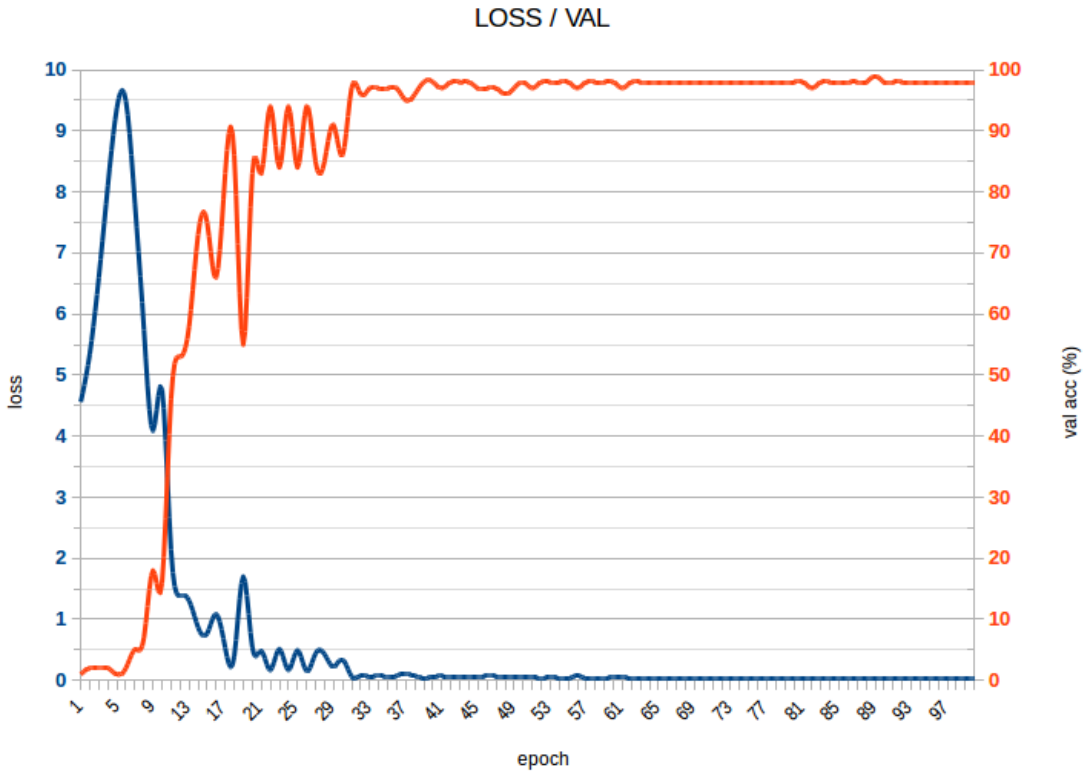


Figura 4.16: Función de coste y precisión del ensayo 95 clases con escala

Observando la matriz de confusión (Figura 4.17) vemos que el tornillo con más errores de clasificación es el 018789, clasificándola erróneamente como el tornillo 006805. Estas dos clases son tornillos idénticos pero de tamaños ligeramente distintos (Figura 4.18). En el preprocesado del

tornillo 006805 se recorta a 509 píxeles y el 018789 a 725 píxeles, siendo una diferencia bastante notable.

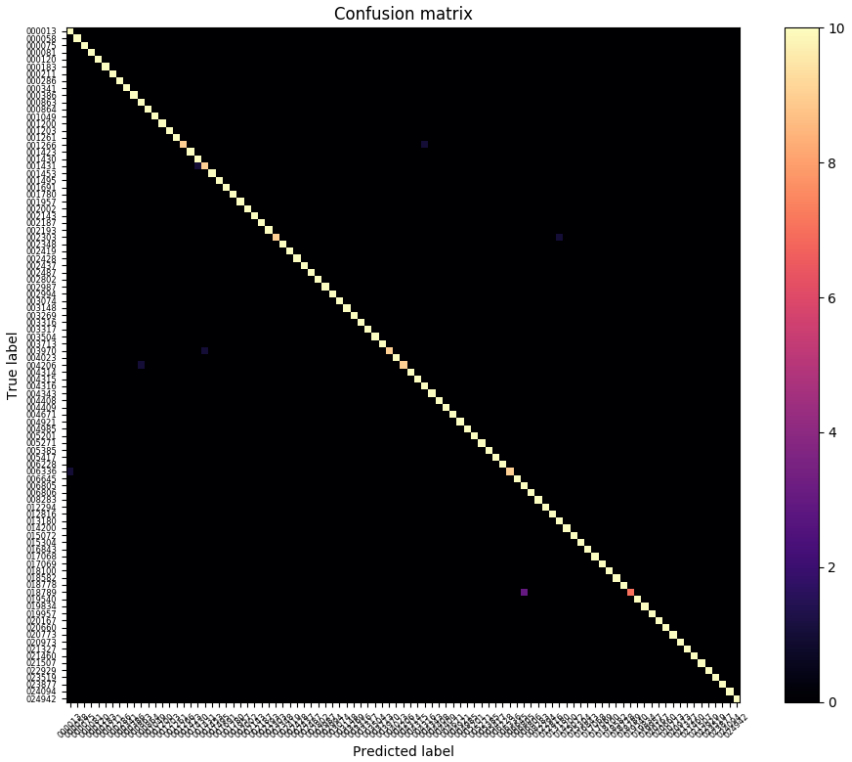


Figura 4.17: Matriz de confusión obtenida en el periodo de evaluación de la época 41

018789 006805

Figura 4.18: Fuente de errores en la clasificación

Se detecta que se ha incurrido en un sesgo al recortar todas las tiradas de cada clase a la misma resolución, por lo que se vuelve a realizar el preprocesado esta vez obteniendo la resolución de recorte independiente para cada tirada, dando cierta variabilidad a esta medida para cada clase, ya que se ha detectado que la red esta tomando este parámetro como principal, ya que al tener cada clase una resolución fija y probablemente única, es mucho más fácil identificar sobre esta medida que sobre información extraída de la imagen.

Durante este entrenamiento se detecta que los tornillos 000013 y 006336 son iguales, ambos corresponden al tornillo M3X8 TT85T aunque tengan un número de serie diferente. Se procede a eliminar la clase 006336, por lo que ahora tenemos 94 clases de tornillos a clasificar.

Entrenando el modelo hasta que tenemos un 99 % de acierto en validación, se realiza una prueba de clasificación con un ejemplo de cada clase. Tres ejemplos no son clasificados a la clase correspondiente, siendo evidente que faltan más datos para que estas medidas sean concluyentes.

Tipo de pieza	Clasificado	SoftMax	Resultado
005271	004343	0.60594964	FALSE
002303	004206	0.6572935	FALSE
000075	000075	0.7361178	TRUE
017068	017068	0.79818016	TRUE
018789	020167	0.9042065	FALSE

Tabla 4.2: Clases con menor salida

Si se dejan pasar mas épocas durante el entrenamiento, podemos llegar a un 100 % de acierto sobre el set de validación. Aun así, al ejecutar el mismo test se observa que sigue etiquetando el tornillo 018789 como 006805 con una salida de la función Softmax de 0.9996276, por lo que no sería capaz de detectar el error mediante un umbral en ese valor. Además, un 100 % de acierto sobre validación suele ser señal de que el modelo presenta sobreajuste, por lo que es interesante acabar el entrenamiento un poco antes y preservar la capacidad de la red de generalizar sobre los datos presentados.

Tras realizar diferentes experimentos, la arquitectura de AlexNet demuestra ser insuficiente, por lo que se implementan dos nuevas arquitecturas de red más profundas, que deberían ser capaces de generalizar con un menor volumen de datos. Estas son VGG (Simonyan y Zisserman 2014) y ResNet (He y col. 2015). Ambas se han implementado con diferentes niveles de profundidad. A más profundidad obtendremos teóricamente una menor tasa de error, pero el coste computacional y el uso de memoria aumenta de manera considerable, por lo que se ha de encontrar una solución de compromiso.

ResNet es fruto del equipo de investigación en inteligencia artificial de Microsoft, en el cual se introduce el uso de bloques de conexión residual (Figura 4.19). En estos bloques se realizan las operaciones de convolución y *batch normalization*, y tras ellas se le suma la entrada original al bloque. Mediante este procedimiento se resuelve el problema del *desvanecimiento del gradiente*, el cual provoca la "muerte" de algunas neuronas durante el entrenamiento, perdiendo información en el proceso. ResNet 18 utiliza 4 de estos bloques en serie, finalizando en una capa de neuronas completamente conectadas encargadas de realizar la tarea de clasificación.

La arquitectura de red VGG fue desarrollada por el grupo de visión artificial de la universidad de Oxford para el ILSVRC-2014, ganando con una precisión top-5 del 92.7 %. Esta arquitectura reemplaza los filtros 11x11 de AlexNet por filtros más pequeños 3x3, haciendo la red más profunda.

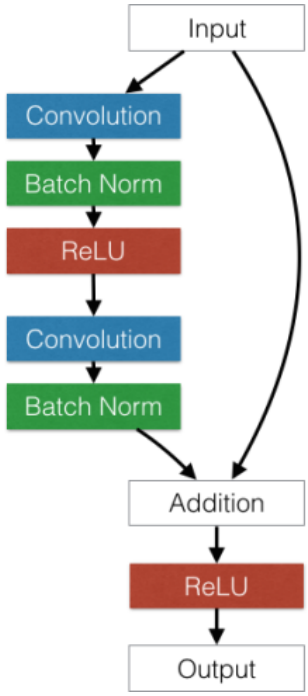


Figura 4.19: Bloque ResNet (He y col. 2015)

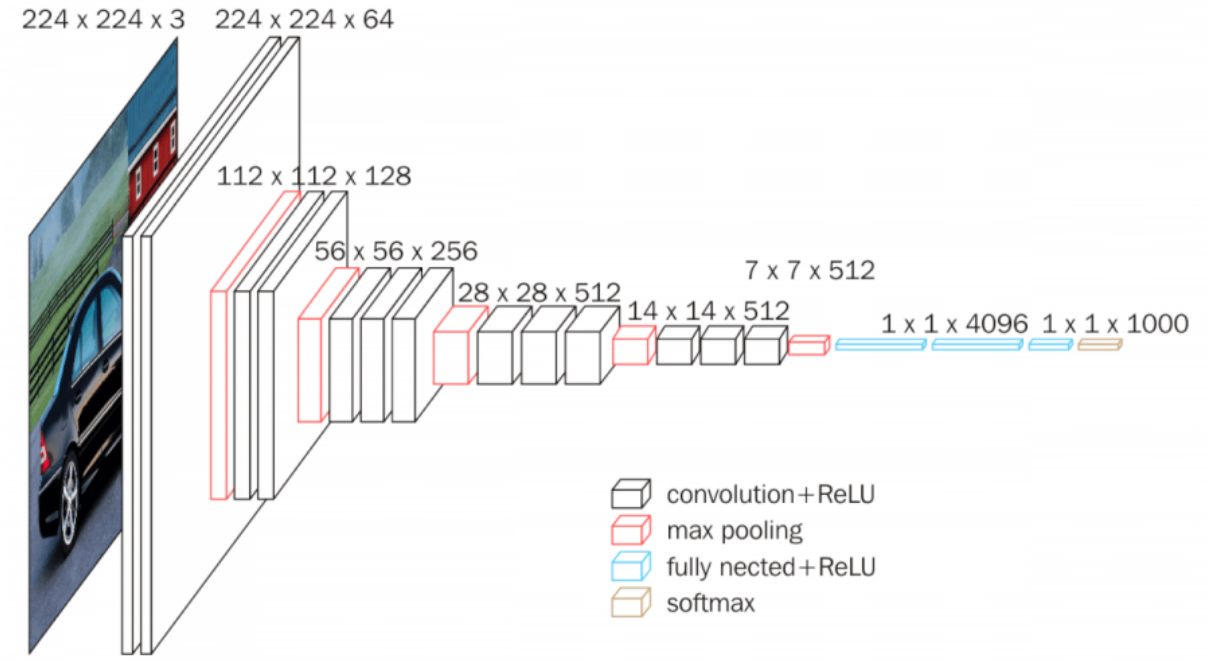


Figura 4.20: Arquitectura VGG16 (Simonyan y Zisserman 2014)

VGG necesita grandes cantidades de memoria debido a la cantidad de parámetros que extrae, por lo que el batch size se ve limitado a 2, usando para ello 6GB de VRAM en la GPU. ResNet permite un batch size de hasta 16 sin agotar memoria, por lo que converge mucho más rápido. Además el modelo creado es mucho más ligero y, aún teniendo mayor nivel de profundidad, necesita menos parámetros para clasificar.

Con el mismo dataset se entrena una red con la arquitectura ResNet con una profundidad de 18 capas. Esta clasifica todos correctamente en tan solo 13 épocas en 1 hora y 40 minutos. En una etapa de testeo donde se pasa un ejemplo de cada clase todas han sido clasificadas correctamente, con una salida de SoftMax siempre superior a 0.99. Demostrado el buen funcionamiento de la arquitectura ResNet se pasa a completar el modelo utilizando el dataset con todas las clases mencionadas en el apartado 3.3.

4.5 Implementación del modelo con el dataset completo

En esta sección se aplicará las redes planteadas en el apartado anterior para clasificar todos las clases de objetos de `www.mootio-components.com` detallados en la sección 3.3, evaluando así el rendimiento de la red, además de poder realizar mediciones de tiempos de entrenamiento y ejecución en un caso que se asemejaría a un entorno industrial real.

El nuevo dataset es proporcionado en forma de imágenes *'.jpg'* con fondo rojo, debido a que se va a desarrollar en paralelo otro tipo de clasificador k-vecinos, el cual necesita este fondo de color vivo para funcionar correctamente. El primer paso que se plantea es eliminar el fondo rojo.

Para realizar esta operación se transforma la imagen en color a espacio HSV (del inglés Hue, Saturation, Value – Matiz, Saturación, Valor). Se utiliza esta codificación de color debido a que es más útil para trabajar con las diferentes tonalidades de un color específico, ya que su codificación es más parecida a la percepción que tenemos los humanos sobre el color, permitiendo determinar un rango donde el color sea similar.

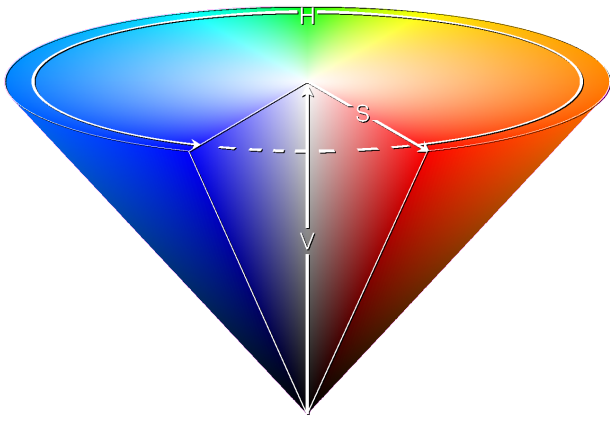


Figura 4.21: Cono de valores HSV

Se obtiene una suma de dos máscaras debido a que le rojo se encuentra a los extremos del rango H. Hay que tener en cuenta que, sobre todo en las objetos más pequeños, debido a la compresión JPEG hay píxeles con valores H correspondiente al rojo dentro de lo que sería el objeto, por lo que hay que afinar mediante los parámetros S y V, ya que estos son más *oscuros* que el fondo de la imagen. En la figura 4.22 podemos observar este proceso en la arandela más pequeña del dataset, por lo que algunos píxeles en la frontera entre el objeto y el fondo permanecen, y al reali-

zar la transformación a escala de grises estos pasan a formar parte del objeto de manera aparente.

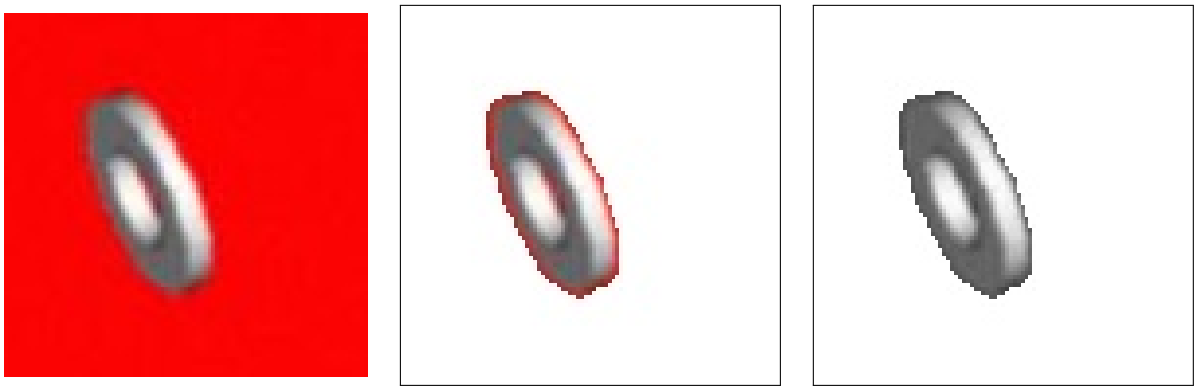


Figura 4.22: Preprocesado arandela 000005 (recortado)

Para coronas, muelles y rodamientos se proporcionan inicialmente solo 20 lanzamientos por clase, dividiendo en 15 para train, 4 para test y uno para validación. Realizando el entrenamiento se observa claramente como la menor cantidad de imágenes disponibles para entrenamiento da como resultado una menor convergencia, presentando mayor confusión para estas clases, por lo que se generan más capturas para obtener los 100 lanzamientos al igual que en las otras clases.

Otros 10 lanzamientos adicionales de cada pieza se han simulado para realizar un testeo externo de la clasificación de estas piezas y realizando un estudio para determinar si un umbral en el valor softmax podría ser útil para detectar objetos clasificados de manera errónea.

Realizados los primeros tests se llega a la conclusión de que es interesante de dejar el entrenamiento unas cuantas épocas más tras llegar a un acierto del 99 %, y que se detecta que en ese momento todavía la red no utiliza el parámetro de resolución con el peso que nos gustaría, por lo que aunque piezas iguales de diferentes tamaños se clasifican correctamente, dan una salida softmax baja, y dificultarían la tarea de umbralizado para detectar objetos mal clasificados.

Realizando un entrenamiento de 30 épocas se llega a un mínimo en la función de perdidas, por lo que se considera un entrenamiento bueno para realizar el testeo de la red con este set adicional de lanzamientos, obteniendo los siguientes resultados.

Total lanzamientos	2240	100.00 %
Correctos	2236	99.82 %
Incorrectos	4	0.18 %

Tabla 4.3: Resultados validación (30 épocas)

Ground Truth	Tirada	Salida	Resultado	SoftMax	Umbralizado
muelles_025525	102	muelles_025526	FALSE	0.80714	FALSE
muelles_025526	109	muelles_025525	FALSE	0.98332	TRUE
muelles_025526	102	muelles_025525	FALSE	0.98365	TRUE
muelles_025526	104	muelles_025525	FALSE	0.98706	TRUE
arandelas_000339	105	arandelas_000339	TRUE	0.69321	FALSE
muelles_025526	106	muelles_025526	TRUE	0.74878	FALSE
arandelas_019541	107	arandelas_019541	TRUE	0.92241	FALSE
tornillos_001495	104	tornillos_001495	TRUE	0.96448	FALSE
muelles_025526	107	muelles_025526	TRUE	0.97675	FALSE
muelles_000964	104	muelles_000964	TRUE	0.98027	TRUE

Tabla 4.4: Fuente de errores

En la tabla 4.4 se encuentran los objetos clasificados erróneamente y los objetos clasificados correctamente con menor salida SoftMax del set de validación, de manera que podemos obtener una primera aproximación de un valor de umbral SoftMax de 0.98 determinado empíricamente para detectar errores de clasificación. La tirada 102 del muelle 025525 se ha clasificado erróneamente, pero el sistema la rechazaría por estar por debajo del umbral. Los valores en color rojo significa que el error no sería detectado ya que están por encima del umbral. Los valores naranja son aquellos que se rechazarían a pesar de estar clasificados correctamente.

Los objetos clasificados de manera incorrecta se corresponden con los muelles 025525 y 025526, los cuales tienen ambos la misma longitud y número de espiras, pero con diámetros de 15 y 14 milímetros respectivamente, por lo que para algunos lanzamientos puede que la información extraída no sería suficiente para una diferencia tan sutil.

4.6 Comportamiento ante piezas defectuosas

Se ha generado una serie de piezas defectuosas para comprobar la respuesta de una red entrenada solo con piezas correctas. Los defectos se consideran ligeros, como pequeñas muescas o dobleces, y graves, realizando cortes en la pieza o deformaciones considerables.

Las piezas defectuosas se han generado mediante dos métodos. En el primero se ha utilizado el software libre Blender, donde partiendo del archivo de malla *.ply* se aplican una serie de transformaciones al objeto según los criterios mencionados anteriormente y se generan las capturas sintéticas a partir de estos modelos, intentando obtener defectos de fabricación coherentes al tipo de pieza. El segundo método que se ha implementado es un script escrito en python el cual aplica fuerzas en puntos aleatorios de la malla en la misma dirección que la suma de las normales de los planos a los que pertenece el punto. Esta fuerza es propagada al resto de los puntos de la pieza, siendo la magnitud de esta inversamente proporcional a su distancia al punto de origen, consiguiendo así la deformación del objeto.

Se han simulado cinco lanzamientos de cada objeto defectuoso, descartando así errores de clasificación debido a una orientación particular del objeto ante las cámaras.

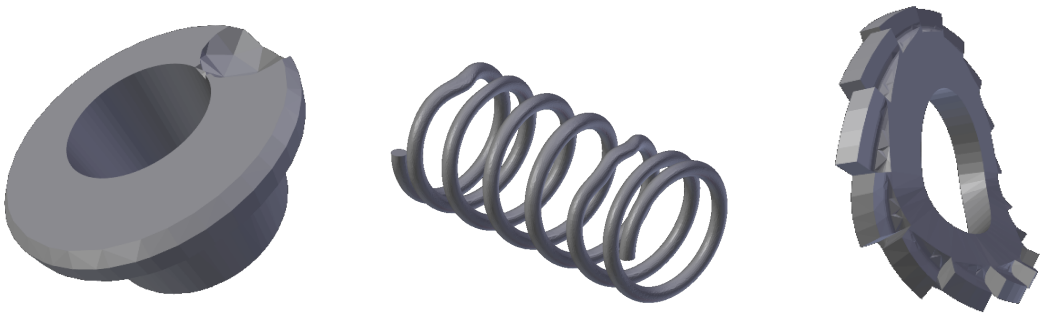


Figura 4.23: Ejemplos defectos ligeros

Figura 4.24: Ejemplos defectos extremos

El objetivo de estos experimentos es comprobar la siguiente premisa: si el defecto es ligero, la red clasificará el objeto correctamente con un índice de confianza alto, y serán las posteriores etapas de verificación las que se encarguen de decidir si el objeto cumple con las especificaciones. Si el defecto es grave, la red clasificará el objeto con una salida softmax más baja, por lo que se puede descartar directamente la pieza en esta etapa.

Tras realizar los ensayos consecuentes, se observa que el comportamiento de la red ante las piezas defectuosas no es exactamente el esperado, aunque se le aproxima, así que el uso de un valor umbral no descartaría piezas defectuosas, pero sí puede evitar clasificaciones erróneas, algunos de ellos debido a piezas con deformaciones considerables. Se ha determinado de manera empírica un valor umbral de 0.98 en la salida *softmax*, el cual minimiza las piezas mal etiquetadas que quedan por encima de ese valor, sin que ninguna pieza correcta obtenida en el apartado 4.5 quede fuera de ese valor. El método de obtención del valor de umbral dependería del tipo de tarea en la que se quiera emplear la red, pudiendo utilizar métodos de aprendizaje automático para determinar un valor que cumpla con las especificaciones deseadas.

Umbral SoftMax	0.98
Total lanzamientos	8218

Correctas	8109	98.67 %
Incorrectas	109	1.33 %
Pasa Corte	**7928**	**96.47 %**
Correctas Descartadas	200	2.43 %
Incorrectas Pasan	39	0.47 %
Correctas Pasan	7889	96.00 %

Tabla 4.5: Resultados set defectos

También se ha observado que pueden ocurrir casos particulares de deformaciones que hacen que el objeto deformado entre dentro de la categoría de otro. Esto se ha dado en el rodamiento defectuoso (7)14546, el cual se generó eliminando la sección perteneciente al anillo interior del rodamiento, dando así un rodamiento con un diámetro interior mayor, por lo que se confunde con el rodamiento 025401, de igual diámetro externo pero mayor diámetro interno, por lo que el defecto cae dentro de la zona de otra clase. Otro ejemplo viene dado por el tornillo (3)00863, el cual se le ha cortado una sección por la punta, por lo que entra dentro de la clase 001957, el cual tiene la misma cabeza pero es más corto.

Umbral SoftMax	0.98
Total lanzamientos	10458

Correctas	10345	98.92 %
Incorrectas	113	1.08 %
Pasa Corte	**10162**	**97.44 %**
Correctas Descartadas	205	1.96 %
Incorrectas Pasan	42	0.40 %
Correctas Pasan	10120	96.77 %

Tabla 4.6: Resultados set completo

Los resultados obtenidos son realmente buenos, con solo un 0.4 % de piezas mal etiquetadas que no se han detectado es un coste asumible, al igual que las correctas descartadas, ya que al ser objetos de bajo valor unitario no es crítico, y se pueden recircular para que pasen una segunda vez por si la tirada anterior hubiese dado una orientación singular ante las cámaras que dificulte la correcta identificación del objeto.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

El objetivo principal de este trabajo es plantear y diseñar un clasificador basado en redes neuronales artificiales orientado a la clasificación de objetos 3D a partir de una representación multivista.

La principal ventaja del clasificador planteado en este trabajo es que se ha trabajado con las imágenes directamente, mientras que el clasificador basado en *k-vecinos* empleado actualmente trabaja con la reconstrucción 3D obtenida a partir de las imágenes, en la cual se pierden detalles concretos que pueden ser determinantes a la hora de diferenciar entre las clases planteadas, como el tipo de cabeza de un tornillo o el tipo de acabado de una arandela. Además, esta clasificación pre-reconstrucción es interesante a la hora de aportar información interesante a la hora de realizar la reconstrucción, facilitando así tareas como el alineado de la pieza al sistema de referencia.

La adaptación realizada a las arquitecturas de redes neuronales, utilizando la resolución de las imágenes recortadas como parámetro de escala del objeto, ha permitido a las redes extraer la información óptima para cada tipo de objeto, manteniendo el tamaño del objeto como parámetro a tener en cuenta, haciendo el sistema más flexible para diferentes entornos industriales. Esta adaptación es posible gracias a que la disposición de las cámaras es fija, por lo que los objetos siempre estarán a la misma distancia de las cámaras.

Los resultados obtenidos son satisfactorios, cumplen ampliamente lo marcado en los objetivos, e incluso se ha ido un paso más adelante al testear la red con el conjunto de piezas deformadas, comprobando la capacidad de generalizar de la red y los límites de esta.

El uso de la generación de imágenes sintéticas permite obtener un dataset de tamaño significativo que permita entrenar y validar las redes neuronales profundas correctamente, ahorrando gran cantidad de tiempo que se debería emplear para conseguir el mismo dataset de imágenes reales correctamente etiquetadas.

Una ventaja al clasificar con imágenes es que la obtención de estas son el primer paso realizado por ZG3D, por lo que si el mismo clasificador detecta que la pieza introducida es una pieza

no deseada, el sistema puede directamente rechazarla, ahorrando así los siguientes pasos de reconstrucción y medición, los cuales son computacionalmente mucho más costosos, permitiendo un ahorro en tiempos.

Aun así el sistema presenta ciertos inconvenientes que cabe destacar. Las redes neuronales funcionan como un sistema de caja negra, por lo que es muy complicado detectar un etiquetado falso que dé una salida *softmax* alta. Además, a la hora de añadir una nueva categoría nueva usualmente se ha de entrenar el modelo de nuevo, por lo que hay que conservar el dataset original, aunque se están desarrollando métodos para evitar este inconveniente que se comenta en la sección siguiente.

El último inconveniente a solucionar sería ejecutar el software en el hardware adecuado. Aunque las redes neuronales se pueden ejecutar en diferentes dispositivos, lo más eficiente es utilizar un hardware del tipo GPU. Esta limitación pronto dejará de ser importante ya que están proliferando los dispositivos de bajo coste con unidades de procesamiento orientadas a las redes neuronales, por lo que no sería necesario adquirir una GPU para tener una aplicación eficiente de las redes neuronales.

5.2 Líneas futuras

El software desarrollado durante la realización de este trabajo será integrado dentro de las herramientas utilizadas por ZG3D para su funcionamiento, pudiendo ofertar el método de clasificación desarrollado al cliente.

Sería interesante ver como un modelo entrenado mediante las imágenes generadas sintéticamente responde ante imágenes obtenidas del dispositivo real. Esto presentaría un gran ahorro en tiempo y costos, ya que el uso de redes neuronales requiere de una cantidad de datos considerables, y generar estos datos a partir de objetos reales se ha de invertir mucho tiempo, tiempo además el cual el dispositivo no esta disponible para realizar tareas de inspección en la linea.

Se plantea desarrollar un sistema híbrido (Ren y col. 2014), el cual a partir de la red neuronal previamente entrenada, tomar las salidas de la etapa convolucional de la red tras el *vew-pooling* para alimentar las características de un clasificador basado en k-vecinos, pudiendo así añadir nuevas categorías de manera más flexible y rápida.

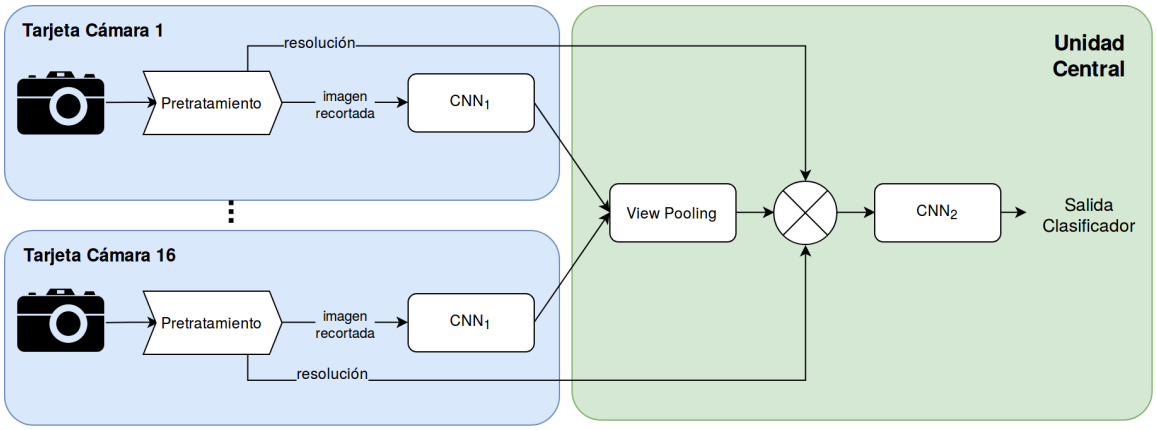


Figura 5.1: Esquema de funcionamiento distribuido

En la siguiente versión de ZG3D, cada cámara o grupo de cámaras llevarán una tarjeta dedicada a realizar el tratamiento de imagen de manera distribuida, por lo que se podría aprovechar y dividir el modelo utilizado para realizar las operaciones convolucionales en cada una de estas tarjetas, y pasar los resultados a la unidad central donde se realizaría la clasificación final. Este planteamiento podría reducir los tiempos de ejecución gracias a que las 16 imágenes se procesarían de manera simultánea en cada procesador de las distintas cámaras, por lo que podría compensar la menor potencia de estas tarjetas.

Otro aspecto a mejorar sobre las redes neuronales, es como poder añadir una nueva categoría al clasificador sin tener que entrenar el modelo por completo de nuevo. Una de las técnicas desarrolladas más prometedoras es la *Learning without forgetting* (Li y Hoiem 2017), donde se puede reentrenar el modelo con nuevas categorías sin mantener el dataset original.

Bibliografía

Aleksander, I. y H. Morton (1990). *An Introduction to Neural Computing*. New York, NY, USA: Van Nostrand Reinhold Co. ISBN: 0-442-31218-0 (vid. pág. 9).

Fergus, Rob (2015). *Neural Netwoks. MLSS 2015 Summer School* (vid. págs. 12, 13).

Fukushima, Kunihiko (feb. de 1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". En: *Biological cybernetics* 36, págs. 193-202. DOI: 10.1007/BF00344251 (vid. pág. 11).

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (vid. pág. 13).

He, Kaiming y col. (2015). "Deep Residual Learning for Image Recognition". En: *CoRR* abs/1512.03385. arXiv: 1512.03385 (vid. págs. 36, 37).

Hebb, Donald O. (jun. de 1949). *The organization of behavior: A neuropsychological theory*. ISBN: 0-8058-4300-0 (vid. pág. 8).

Kingma, Diederik P. y Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". En: *CoRR* abs/1412.6980. arXiv: 1412.6980 (vid. págs. 13, 14).

Krizhevsky, Alex, Ilya Sutskever y Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". En: *Advances in Neural Information Processing Systems 25*. Ed. por F. Pereira y col. Curran Associates, Inc., págs. 1097-1105 (vid. pág. 25).

LeCun, Y. y col. (1990). "Handwritten digit recognition with a back-propagation network"". En: *Advances in Neural Information Processing Systems (NIPS 1989)*. Ed. por David Touretzky. Vol. 2. Denver, CO: Morgan Kaufman (vid. pág. 11).

Li, Zhizhong y Derek Hoiem (nov. de 2017). "Learning without Forgetting". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP, págs. 1-1. DOI: 10.1109/TPAMI.2017.2773081 (vid. pág. 45).

Mei, Song, Andrea Montanari y Phan-Minh Nguyen (2018). "A mean field view of the landscape of two-layer neural networks". En: *Proceedings of the National Academy of Sciences* 115.33, E7665-E7671. ISSN: 0027-8424. DOI: 10.1073/pnas.1806579115. eprint: https://www.pnas.org/content/115/33/E7665.full.pdf (vid. pág. 13).

Perez-Cortes, Juan-Carlos y col. (2018). "A System for In-Line 3D Inspection without Hidden Surfaces". En: *Sensors* 18.9. ISSN: 1424-8220. DOI: 10.3390/s18092993 (vid. pág. 7).

Ren, W. y col. (ago. de 2014). "Learning Convolutional Nonlinear Features for K Nearest Neighbor Image Classification". En: *2014 22nd International Conference on Pattern Recognition*, págs. 4358-4363. DOI: 10.1109/ICPR.2014.746 (vid. pág. 44).

Rumelhart, David E., Geoffrey E. Hinton y Ronald J. Williams (1986). "Learning representations by back-propagating errors". En: *Nature*. DOI: https://doi.org/10.1038/323533a0 (vid. pág. 11).

Russakovsky, Olga y col. (2015). "ImageNet Large Scale Visual Recognition Challenge". En: *International Journal of Computer Vision (IJCV)* 115.3, págs. 211-252. DOI: 10.1007/s11263-015-0816-y (vid. pág. 22).

Samuel, Arthur L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". En: *IBM Journal of Research and Development* (vid. pág. 7).

Simonyan, Karen y Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". En: *CoRR* abs/1409.1556. arXiv: 1409.1556 (vid. págs. 36, 37).

Srivastava, Nitish y col. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". En: *Journal of Machine Learning Research* 15, págs. 1929-1958 (vid. pág. 26).

Su, Hang y col. (2015). "Multi-view convolutional neural networks for 3d shape recognition". En: *Proc. ICCV* (vid. págs. 24, 25).

Szegedy, Christian, Wei Liu y col. (jun. de 2015). "Going deeper with convolutions". En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2015.7298594 (vid. págs. 21, 22).

Szegedy, Christian, Vincent Vanhoucke y col. (jun. de 2016). "Rethinking the Inception Architecture for Computer Vision". En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2016.308 (vid. págs. 21, 22).

Índice de figuras

Índice de tablas

Parte II

Presupuesto

Capítulo 6

Presupuesto

En este capítulo se presentará el presupuesto necesario para la elaboración del presente trabajo, desglosado en costes de recursos humanos y costes materiales. Se pretende que este sirva de guía para la estimación de proyectos futuros similares.

6.1 Costes recursos humanos

Para el calculo del coste en recursos humanos se ha dividido el trabajo en diferentes bloques. El único recurso que se contempla es el del ingeniero industrial recién titulado (I), con un salario de 30.00€/hora. Se añade también las tareas realizadas por el tutor (T) de dirección y revisión del trabajo, con un coste de 60.00€/hora.

CONCEPTO	**TIPO**	**UD**	**Nº**	**PRECIO UNITARIO (€/hora)**	**TOTAL sin IVA (€)**
Formación y documentación	I	h	160	30.00	2,400.00
Diseño e implementación de las CNNs	I	h	120	30.00	6,000.00
Evaluación de las diferentes alternativas	I	h	40	30.00	1,200.00
Análisis de los resultados	I	h	20	30.00	600.00
Redacción de la memoria	I	h	50	30.00	1,500.00
Revisión de la memoria	I	h	10	30.00	300.00
Revisión y dirección	T	h	20	60.00	1,200.00
TOTAL			**420**		**13,200.00**

Tabla 6.1: Desglose costes recursos humanos

El coste asociado a los recursos humanos asciende a **trece mil doscientos euros**.

6.2 Costes de equipamiento

En cuanto al equipamiento se ha tenido en cuenta el *software* y el *hardware* utilizado para la realización de los trabajos comentados en esta memoria, incluido la redacción de la misma.

Todo el software utilizado en este trabajo ha sido *open source*. La libreria de PyTorch™ se distribuye bajo la licencia de software libre permisiva BSD[1]. El interprete de Python es distribuida bajo la licencia *Python Software Foundatio License*[2] (PSFL), compatible con la *GNU General Public License* (GPL). El software de diseño 3D Blender utiliza licencia GNU GPL[3]. La memoria ha sido redactada utilizando LaTeX, distribuido bajo licencia *LaTeX Project Public License*[4], siendo esta una licencia de software libre sin *Copyleft*. Todo esto se ha ejecutado sobre el sistema operativo Ubuntu 16.04, un sistema operativo basado en Linux distribuido bajo licencia GPL[5]. Ningún elemento de software utilizado tiene un coste directo imputable.

El hardware utilizado ha sido un equipo personal donde se han realizado la mayoría de las tareas, y un segundo equipo equipado con una GPU NVIDIA GTX 1080 para realizar los entrenamientos. Como de este segundo equipo solo es interesante por su unidad gráfica, solo se ha tenido en cuenta este elemento en la elaboración del presupuesto.

El coste imputable de los materiales C se ha calculado a partir de la ecuación 6.1, siendo c el coste unitario, t el tiempo de uso y T la vida útil.

$$C = \frac{c \cdot t}{T} \tag{6.1}$$

CONCEPTO	**UD**	**Nº**	**VIDA ÚTIL (años)**	**PERIODO DE USO (años)**	**PRECIO UNITARIO sin IVA (€)**	**COSTE IMPUTABLE sin IVA(€)**
Ordenador Personal (core i7, 10Gb de RAM)	u	1	5	0.3	1500.00	90.00
GPU NVIDIA GTX 1080	u	1	5	0.3	1200.00	72.00
TOTAL						**162.00**

Tabla 6.2: Desglose costes materiales

El coste del equipamiento asciende a **ciento seseinta y dos euros**.

[1] Licencia PyTorch: `https://github.com/pytorch/pytorch/blob/master/LICENSE`
[2] Licencia Python: `https://docs.python.org/3/license.html`
[3] Licencia Blender: `https://www.blender.org/about/license/`
[4] Licencia LaTeX: `https://www.latex-project.org/lppl/lppl-1-3c/`
[5] Licencia Ubuntu: `https://www.ubuntu.com/licensing`

6.3 Presupuesto del proyecto

El importe total del proyecto se calcula como la suma de los costes de recursos humanos y materiales, se calculan los gastos generales (16 %) y el beneficio industrial (6 %), dando el presupuesto de ejecución por contrata, al cual aplicamos el IVA (21 %), obteniendo el importe total del proyecto.

CONCEPTO	IMPORTE (€)
Presupuesto de ejecución material	13,362.00
Gastos generales (16 %)	2,137.92
Beneficio industrial (6 %)	801.72
Presupuesto de ejecución por contrata	16,301.64
IVA (21 %)	3,423.34
TOTAL	**19,724.98**

Tabla 6.3: Desglose del presupuesto total

El coste total del proyecto asciende a **DIECINUEVE MIL SETECIENTOS VEINTICUATRO EUROS CON NOVENTA Y OCHO CÉNTIMOS**.