Aprendiendo a leer la hora

Daniel Ortiz Romero

Resumen. — Este proyecto se engloba dentro del ámbito de la Inteligencia Artificial, tiene como objetivo la realización de un caso práctico utilizando herramientas de Deep Learning o aprendizaje profundo, desarrollando una red neuronal que nos permita a partir de una imagen de un reloj analógico que marca una hora concreta, que nuestro programa sea capaz de adivinar la hora que marca ese reloj.

A nivel de programación se ha elegido el lenguaje de programación Python, utilizando la librería Pytorch para desarrollar la red neuronal.

Para entrenar el sistema se generan aleatoriamente un gran número de relojes analógicos, para este caso de estudio en concreto son imágenes que cada una de ellas tiene dos agujas correspondientes a la hora y minutos del reloj, por cada reloj se guarda un registro que etiquetara la hora exacta que muestra.

Palabras clave— Inteligencia Artificial, Deep Learning, redes neuronales, Machine Learning, Python, PyTorch, predicción de datos.

Abstract— This project is included within the field of Artificial Intelligence, its objective is to carry out a practical case using Deep Learning or deep learning tools, developing a neural network that allows us from an image of an analog clock that marks one hour concrete, that our program is able to guess the time that that clock marks.

At the programming level, the Python programming system has been chosen, using the Pytorch library to develop the neural network.

To train the system, a large number of analog clocks are randomly generated, for this particular case study they are images that each have two hands corresponding to the hour and minute of the clock, for each clock a record is kept that will label the exact time showing.

Index Terms— Artificial Intelligence, Deep Learning, neural networks, machine learning, Python, PyTorch, data prediction.

1 Introducción

La Inteligencia Artificial en estos últimos años ha desarrollado un crecimiento considerable, pasando a ser un campo esencial en la transformación de la sociedad en el presente y futuro. Hemos podido ver su aplicación en numerosos ámbitos de sociedad, ya sea industrial, militar, comercial o personal. Hemos visto como se ha integrado en la vida de las personas, empresas e instituciones, poniendo al alcance del gran público herramientas que nos permiten trabajar con gran cantidad de datos, creando sistemas de aprendizaje automático que van mejorando a medida que alimentamos el nivel de información y que pueden predecir con cada vez mejor acierto el comportamiento de la información.

Todo esto está ligado a la mejora en el poder de cálculo en la computación, y concretamente en la mejora y optimización del rendimiento de CPUs y GPUs, este rendimiento ha ido aumentando considerablemente, siendo las GPUs las que han tenido un papel principal en proyectos de Deep Learning.

Una de las áreas que más ha evolucionado es en el reconocimiento de imágenes, ya que tiene numerosas utilidades en multitud de campos diferenciados, por otro lado, esto también ha hecho que surjan críticas a estos sistemas ya que pueden ser usados para objetivos éticamente cuestionables y que exista un debate ético como esta tecnología debe ser utilizada.

En este proyecto se quiere desarrollar un caso práctico utilizando estas herramientas, concretamente queremos ser capaces de que dada una imagen de un reloj analógico el sistema sea capaz de adivinar qué hora es.

Para ello deberemos disponer de un gran número de imágenes que contengan relojes analógicos, que proyecten una hora concreta debidamente etiquetada, a partir de ahí, generar un sistema de aprendizaje profundo, que una vez habiendo entrenado con un gran número de imágenes, este sea capaz de que, dada una nueva imagen, el sistema devuelva la hora que marca de una manera acertada.

Estas imágenes las hemos generado nosotros desde Python para controlar sus características ya que podemos controlar su peso a nivel de resolución, manteniendo una homogeneidad en las imágenes, que nos ayudará a desarrollar nuestra prueba de concepto.

1.2 Objetivos:

- Dado que es un caso práctico en un proyecto de Grado, este proyecto tiene como objetivo principal la obtención de los conocimientos necesarios para poder utilizar con éxito este tipo de técnicas y demostrar que es posible llegar a resultados acertados.
- El objetivo específico del proyecto es crear una red neuronal que sea capaz a partir de imágenes de relojes analógicos, saber leer que hora es exactamente.
- También tiene como objetivo evaluar las posibilidades de estas herramientas de reconociendo de imágenes para poder ayudar a personas con problemas de visión. Este sería el primer paso, para comprobar que dada una imagen un sistema es capaz de leer una hora especifica.

1.3 Etapas y Organización del Proyecto

Recopilación de información y organización.

Esta etapa ha consistido en introducirse en las técnicas de aprendizaje profundo, para ello se han seguido tutoriales, apuntes y videos de información de internet, así como documentación y material de clases pertenecientes a la propia universidad de asignaturas relacionadas con Deep Leearning suministradas por el tutor del proyecto.

Esta recopilación de información ha permitido tener una idea conceptual del proyecto y de generar una organización de tareas para llegar a la realización de los objetivos a nivel de programación, así como a nivel de comprensión de las tareas que se estaban realizando en cada momento.

Formación en Técnicas de Deep Learning.

Para ello se han seguido cursos de la plataforma Coursera en concreto el curso.

Neural Networks and Deep Learning por la deeplearning.ai, impartido por el profesor Andrew Ng.

También ha estado disponible apuntes i materiales de la asignatura “Redes Neuronales y Aprendizaje Profundo” impartido en la propia escuela de ingeniería de la UAB correspondiente al Grado de Matemática Computacional.

Por otro lado, se ha recopilado información de canales oficiales de sitios de internet que trabajan con lenguaje Python y la librería PyTorch con puede ser https://pytorch.org/.

Obtención de Datos de trabajo.

Generación del Datase correspondiente a las imágenes de relojes para el entrenamiento y validación del sistema, en esta etapa se han generado diez mil relojes que nos ayudarán a entrenar y validar nuestro modelo.

Recopilación de modelos similares de redes neuronales capaces de aprender a través de imágenes tanto en tareas de clasificación como de regresión, aunque teniendo en cuenta que nuestro trabajo es un sistema que utiliza la regresión también nos serán útiles los modelos de clasificación ya que hay múltiples ejemplos.

Modelaje.

En esta etapa de diseño se han ido probando diferentes modelos hasta encontrar resultados que han sido satisfactorios, en los cuales se ha ido mejorando la eficiencia del modelo, para ello hay que recordar que al utilizar una plataforma compartida como es Colab de Google, los recursos de computación que se disponían eran altamente limitados por lo que los modelos a implementar tenían que cumplir que fueran óptimos para utilizar sus recursos de una manera eficiente y no saturar la plataforma de ejecución.

Análisis de Resultados.

Aquí analizamos los resultados de los diferentes modelos e intentaremos desglosar que posibilidades tiene cada uno, analizando las predicciones obtenidas comparándolos con las reales veremos el grado de exactitud de estos resultados, así como también podremos ver su eficiencia y complejidad.

Conclusiones y documentación.

En esta etapa se plasmará en este documento un informe del trabajo realizado, así como una conclusión general del proyecto.

1.4. Recursos Utilizados

Colaboratory de Google.

Colaboratory, o "Colab", es un producto de Google Research. Permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación. Desde un punto de vista más técnico, Colab es un servicio alojado de Jupyter Notebook que no requiere configuración y que ofrece acceso gratuito a recursos informáticos, como GPUs.

Los recursos de Colaboratory no están garantizados ni son ilimitados, y los límites de uso a veces varían.

Google Drive.

Google Drive es el servicio de almacenamiento de datos en internet que provee Google, en este caso necesario para guardar los cuadernos de Colab que es donde los almacenaremos, también almacenaremos aquí todo el Dataset con las diez mil imágenes.

Lenguaje de Programación Python.

Entre los lenguajes de programación orientados en el ámbito de la inteligencia artificial Python es uno de los más utilizados. Sobre todo, si nos centramos en deep learning disponiendo de un gran número de bibliotecas como son PyTorch, Caffe, TensorFlow, o Keras, en este caso se ha utilizado la versión correspondiente a la suministrada por colab, por defecto utiliza Python 3. Librerías utilizadas:

- Pytorch: Se ha utilizado esta libreria para ejecutar

todas las operaciones relacionadas con deep learning. Pytorh es un paquete dieseñado para realizar cálculos numéricos haciendo uso de la programación de tensores, por otro lado, permite la ejecución sobre GPU con lo que se consiguen acelerar los procesos de entrenamiento.

- Matplotlib: Se ha utilizado esta librería para la creación de gráficos, en nuestro caso concreto para ver cómo evoluciona el entrenamiento del modelo de una forma gráfica, para ayudarnos a tomar conclusiones.
- Numpy: Está enfocada a la computación científica, dispone de arrays o vectores como potentes herramientas matemáticas que se pueden aplicar sobre estos, se ha utilizado para cargar los datos de ficheros como una estructura de datos.

Fig. 1. Imagenes coorporativas de pyhon, colab y GoogleDrive. Fuente: google fotos. Site colab: https://colab.research.google.com/

2 DISEÑO.

2.1 Dataset.

El dataset se ha construido generando un gran número de imágenes de relojes analógicos de manera aleatoria, que marcan una hora concreta. En nuestro caso concreto hemos generado un total de diez mil imágenes de relojes analógicos.

Estas imágenes son en dos dimensiones y tres canales RGB, estando formadas por un círculo exterior, las doce marcas horarias de un reloj y dos agujas correspondientes a la hora y minutos.

Cada imagen esta descrita con un identificador único, ejemplo: img0, img22, img344, por otro lado, existe un archivo que hace de base de datos que tiene la hora exacta correspondiente a cada una de estas imágenes.

Ejemplos de imágenes.

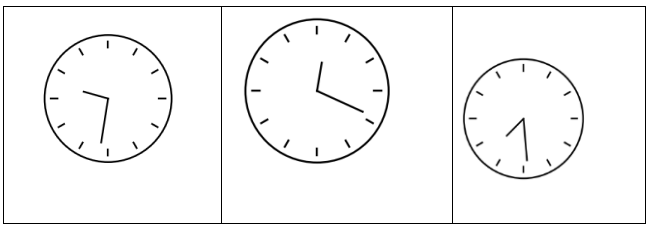

Fig. 2. Ejemplos de relojes generados. fuente propia

Hemos elegido estas imágenes ya que han sido creadas a partir de un algoritmo el cual podemos gestionar nosotros mismo haciendo que aleatoriamente se generen tantas imágenes de relojes como queramos, así también se ha decidido que estas tengan una resolución de 64*64 pixeles que nos genera una resolución manejable para poder realizar los cálculos necesarios en nuestra red neuronal sin saturar el sistema, por otra parte hemos de decir que esto nos puede llegar a limitar en la precisión de los resultados dada esta resolución, es así por lo que se ha llegado a un equilibrio entre exactitud y operatividad creyendo que esta resolución es suficiente para el cometido de este trabajo.

Este tipo de imágenes guardan una similitud en el aspecto que no encontraríamos en la vida real, por lo que siendo este un trabajo de iniciación a la investigación a las posibilidades de una red neuronal se ha creído conveniente que fuera una forma simple de probar la potencia de aprendizaje de nuestra red neuronal, tratando imágenes muy similares, pudiendo cambiar el tamaño y posición, pero siempre desde un diseño único.

Aunque se podrían realizar mejoras en la generación de las imágenes, como podrían ser imágenes más reales en 3D o completar el proceso con imágenes reales, esto haría más complejo el trabajo y quedaría fuera del propósito de este TFG.

Por otro lado, las imágenes generadas pueden ser modificadas para poder así dar más posibilidades nuestro sistema, ya que con el mismo número de imágenes podemos generar modificaciones que nos ayudaran a dar más recursos a nuestro sistema para poder utilizar imágenes modificadas como si fueran diferentes imágenes haciendo modificaciones como podría ser ruido, saturación u otras técnicas posibles.

Hemos de tener en cuenta que hay algunas modificaciones que podrían generar una posible variabilidad de las imágenes a nivel de contexto, por ejemplo, si rotamos una imagen, esta nos podría indicar una hora diferente de la imagen original, perdiendo así su coherencia con la etiqueta correspondiente que especifica la hora concreta, así que se tiene que valorar que modificaciones se pueden hacer y cuáles no.

Transformaciones realizadas(Data Aumentation). :

En proyectos de Deep learning se necesita una gran cantidad de datos para entrenar el modelo pudiendo disponer de más datos con datos aumentados.

Es una práctica común utilizada en el tratamiento de imágenes a través de redes neuronales, para poner aumentar los datos presentes en el dataset.

Data Augmentation, nos permite aumentar nuestro dataset de dos formas, introduciendo perturbaciones en los datos originales, por ejemplo, tomando una imagen original centrada, la replicamos descentrada, invirtiendo ejes, etc. o utilizando distintas distribuciones. Por ejemplo, añadimos imágenes borrosas, con ruido, etc.

Si una imagen es modificada ligeramente, para la red es interpretada por una imagen completamente diferente

pero que mantendrá sus valores de salida sin alterar.

Para poder aumentar el número de datos disponibles se pueden realizar diferentes transformaciones de la imagen original, por la que se pueden obtener imágenes derivadas lo importante de estas transformaciones es que, aunque la imagen cambie su visualización esta siga manteniendo su lógica con su valor de salida.

Hay Transformaciones que cambiarían el contexto de la imagen, ya que dadas sus características cambiarían la lectura contextual de la imagen.

Por ejemplo, si rotamos una imagen, como en la imagen no se especifican los números de cada hora podemos hacer que el sentido contextual de la imagen cambie, es decir que, si rotamos una imagen, la hora que veamos en la nueva imagen esta también cambiará por lo que debemos tener en cuenta que hay transformaciones que no se pueden hacer, para no perjudicar al modelo, ya que aportaríamos información no coherente.

Ejemplo de imagen rotada en 90 grados

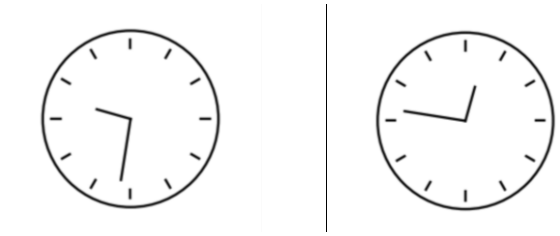

Fig. 3. Resultado de girar una imagen 90 grados. Fuente propia

Las siguientes transformaciones son aleatorias, lo que significa que la misma instancia de transformador producirá un resultado diferente cada vez que transforme una imagen determinada.

```
transforms.Compose([transforms.ToTensor(),
            transforms.ColorJitter(brightness = 0.4),
            transforms.GaussianBlur(11, sigma=(0.1, 2.0)), #pruebas
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           ])
```

Fig. 4. Transformaciones realizadas en PyTorch. Fuente propia

ColorJitter

La transformación ColorJitter cambia aleatoriamente el brillo, la saturación y otras propiedades de una imagen.

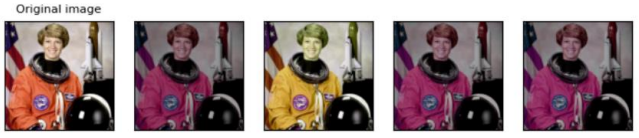

*Fig. 5. Efectos de la transformación ColorJitter. Fuente: https://pytorch.org/ (*Illustration of Transforms — Torchvision 0.10.0 Documentation*, s. f.)*

GaussianBlur

La transformación GaussianBlur realiza una transformación de desenfoque gaussiano en una imagen.

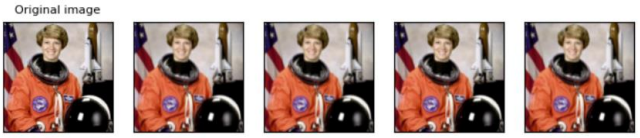

Fig. 6. Efector de la transformación GaussianBlur. Fuente: https://pytorch.org/ (Illustration of Transforms — Torchvision 0.10.0 Documentation, s. f.)

Normalize

Proporciona una media (media) y una varianza (estándar) de todos los canales y normaliza los datos originales. El formato de datos de la operación es Tensor

2.2 Modelo

Para la realización del modelo se han ido probando modelos desde uno base básico, que en un inicio tenía solo 2 convoluciones ejecutándose 2 Maxpool y 2 fully connected layers. Los resultados han ido mejorando desde resultados no muy ajustados y con curva de aprendizaje no satisfactoria a los resultados del modelo final.

```
# definir nuesro CNN model.

class CNN(nn.Module):
    def __init__(self, n_feature, output_size):
        super(CNN, self).__init__()
        self.n_feature = n_feature
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=n_feature, kernel_size = 3)
        self.conv2 = nn.Conv2d(n_feature, n_feature, kernel_size=4)
        self.fc1 = nn.Linear(n_feature*62*62, 50)
        self.fc2 = nn.Linear(50, output_size)
        self.maxpool = nn.MaxPool2d(kernel_size=2)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = x.view(-1, self.n_feature*62*62)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.sigmoid (x)    #CAMBIO Fijamos entre 0-1
        return x
```

Fig. 7. Definición en código de nuestro modelo base utilizado. Fuente propia

Modelo Final.

El modelo final desarrollado, define 4 convoluciones diferentes, así como 3 fully Conected layers y se ejecutan 3 MaxPool en las primeras convoluciones, ejecutando un Adaptive Average Pool en la última.

Este modelo nos da un total de 4096 valores de salida que es una cantidad muy razonable para poder obtener buenos resultados sin sobrecargar el sistema.

Los tres FullConects finales se hacen pasando de 4096 a 300 de 300 a 20 y de 20 a dos, que son las salidas necesarias para predecir la hora y minutos respectivamente.

Utilizamos 4 funciones de activación 3 relu, y una ultima de tipo sigmoide para poder fijar los resultados entre 0 y 1.

Definición.

```
# definir nuestro CNN model.

class CNN(nn.Module):
    def __init__(self, n_feature, output_size):
        super(CNN, self).__init__()
        self.n_feature = n_feature
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size = 3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3) #
        self.fc1 = nn.Linear(256*4*4, 300) #
        self.fc2 = nn.Linear(300, 20)
        self.fc3 = nn.Linear(20, output_size)
        self.maxpool = nn.MaxPool2d(kernel_size=2)
        self.avgpool = nn.AdaptiveAvgPool2d (output_size=(4, 4))

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = self.conv4(x) #
        x = F.relu(x)
        x = self.avgpool(x)
        x = x.view(-1, 256*4*4) #
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.sigmoid (x)    #
        return x
```

Fig. 8. Definición en código de nuestro modelo final de red neuronal. Fuente propia

Diseño

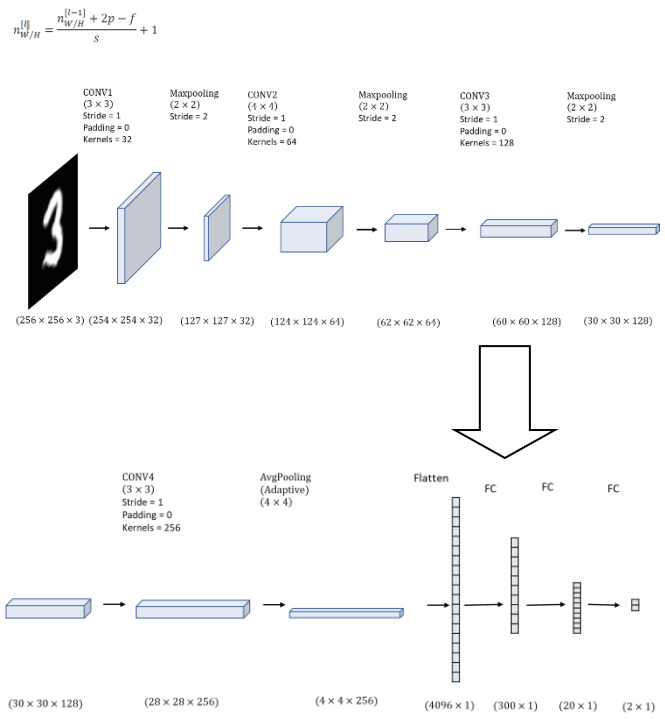

Fig. 9. Diseño conceptual de dimensiones de nuestro modelo. Fuente propia

Cálculos.

En este apartado se realizarán los cálculos para evaluar las diferentes operaciones de la red neuronal.

Recordamos que la fórmula para calcular dimensiones de una convolución también funciona para calcular el tamaño de MaxPool:

$$n_{W/H}^{[l]} = \frac{n_{W/H}^{[l-1]} + 2p - f}{s} + 1$$

Convolución 1:
(3×3) Stride = 1 Padding = 0 Kernels = 32

$$n_{W/H}^{[l]} = \frac{256 + 2.0 - 3}{1} + 1 = 254$$

(254×254×3)

MaxPool 1:
(2x2) Stride = 2

$$n_{W/H}^{[l]} = \frac{254 + 2.0 - 2}{2} + 1 = 127$$

(127×127×32)

Convolución 2:
(4×4) Stride = 1 Padding = 0 Kernels = 64

$$n_{W/H}^{[l]} = \frac{127 + 2.0 - 4}{1} + 1 = 124$$

(124×124×64)

MaxPool:
(2x2) stride = 2

$$n_{W/H}^{[l]} = \frac{124 + 2.0 - 2}{2} + 1 = 62$$

(62×62×64)

Convolución 3:
(3×3) Stride = 1 Padding = 0 Kernels = 128

$$n_{W/H}^{[l]} = \frac{62 + 2.0 - 3}{1} + 1 = 60$$

(60×60×128)

MaxPool:
(2x2) stride= 2

$$n_{W/H}^{[l]} = \frac{60 + 2.0 - 2}{2} + 1 = 30$$

(30×30×128)

Convolución 4

(3×3) Stride = 1 Padding = 0 Kernels = 256

$$n_{W/H}^{[l]} = \frac{30 + 2.0 - 3}{1} + 1 = 28$$

(28×28×256)

Adaptive Average Pool :
Esta operación permite fijar la salida específicamente.
Se fija una salida 4x4
(4x4x256)

Flatten:
Se fijan salidas:
4096 (4096×1)
300 (300×1)
20 (20×1)
2 (2×1)

Pooling.

Las técnicas de pooling nos proporcionan una capa de agrupación que resume una región de neuronas de la capa anterior. Pooling utilizados:

Max pooling.
Pregunta cuál es la mejor detección de una característica en particular en la región. es lo mismo que preguntar si una característica en particular ha sido detectada en cualquier parte del campo receptivo.
Todo gradiente fluye a través del ganador

Average Pooling.
Calcula la respuesta promedio a una característica.
Resume partes iguales a través de todas las unidades.

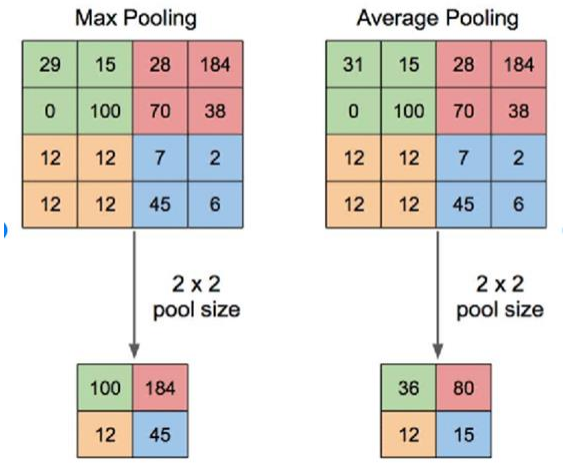

Fig. 10. Funcionamiento de Max Pooling y Average Pooling. Se muestra un ejemplo de operación de agrupación máxima y agrupación promedio con un tamaño de filtro de 2x2 píxeles a partir de una entrada de 4x4 píxeles. En la agrupación máxima, cada filtro se toma el valor máximo y luego se organiza en una nueva salida con un tamaño de 2x2 píxeles. Mientras que el valor de agrupación promedio tomado es el valor promedio del tamaño del filtro Fuente:(Yani et al., 2019).

Adaptive Average Pooling.
Aplica una agrupación promedio adaptativa 2D sobre una señal de entrada compuesta por varios planos de entrada.

La salida es de tamaño H x W, para cualquier tamaño de entrada, que podemos especificar nosotros, así podemos adaptar la salida al tamaño que queramos.

Detalles técnicos.

Funciones de Perdida (Loss fuctions).
"MSE Loss" o "Cross Entropy Loss", en las primeras aproximaciones al modelo se ha utilizado Entropy Loss ya que después de revisar la documentación disponible era muy utilizado, pero sobre todo en modelos de clasificación, dado que el proyecto trabajaba con un modelo orientado a la regresión se decidió cambiar a MSE Loss dando mejores resultados al modelo.

La entropía cruzada "Cross Entropy" funciona mejor que MSE para la clasificación, porque el límite de decisión en una tarea de clasificación es grande (en comparación con la regresión). MSE es la pérdida correcta para la regresión, donde la distancia entre dos valores que se pueden predecir es pequeña.

Es decir, por norma general MSE Loss es mejor para regresión y Cross Entropy Loss es mejor para clasificación.

Learning rate (tasa de aprendizaje)
Es el cambio con el que se actualizan los pesos en cada iteración, cada vez que se realiza una iteración en el proceso de entrenamiento se deben actualizar los pesos de la entrada para poder dar cada vez una mejor aproximación.

Learning rate actúa en como actualizamos los pesos en cada iteración, en un rango de 0 a 1. Poner un valor más cercano a uno podría cometer errores.

En nuestro caso hemos obtenido mejores valores con Learning_rate = 1e-3 que con learning_rate = 1e-2 por lo que ese learning rate más grande creemos que puede cometer más errores que uno más pequeño, aunque necesitemos más interacciones para el entrenamiento.

Algoritmo de Optimización
Adam o stochastic gradient descent (SGD)
SDG mantiene una única tasa de aprendizaje (denominada alfa) para todas las actualizaciones de peso y la tasa de aprendizaje no cambia durante el entrenamiento. En cambio, Adam es como una combinación de las ventajas de otras dos extensiones del descenso de gradiente estocástico AdaGrad y RMSProp.

En nuestro caso se mejoró cambiando la optimización de SGD a Adam.

Motramos las dos maneras de utilizar los dos optimizadores diferentes en pytorch.

optimizer=torch.optim.SGD(model.parameters(), lr=learning_rate,momentum=momentum, weight_decay=lambda_l2)

optimizer=torch.optim.Adam(model.parameters(), lr=learning_rate)

Adam es un algoritmo popular en el campo del aprendizaje profundo.

2.3 Resultados.

En el modelo final realizado podemos observar que se consiguen resultados buenos desde de las primeras epochs, que tenemos que recordar que es el paso sobre el dataset entero.
Sobre todo, se consigue de una forma muy rápida y tan solo con poco más de 10 epochs mejoramos mucho nuestra eficiencia en nuestra red neuronal.
A partir de ahí vemos como la red neuronal sigue mejorando, pero a partir de unas 20 epochs, nuestro sistema converge. Podemos decir que comprobamos como utilizando Adam obtenemos mejores resultados de una manera más rápida.

En las siguientes graficas vamos a ver la evolución de los datos de validación y entrenamiento.

Para ello vamos a comparar cómo evoluciona la función de perdida (los), la función de pérdida, (Loss function), es una función que evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje.
Por lo que calcula la distancia euclidiana entre los datos predichos y los datos verdaderos. Cuanto menor es el resultado de esta función, más eficiente es la red neuronal.
También vamos a mostrar 6 ejemplos de predicciones por cada situación estudiada por el modelo para ver la exactitud de las predicciones en casos reales.
Para poder ver la hora real, especificar que se han convertido las horas y minutos en un rango de 0 a 1 por lo que si queremos saber la predicción de la hora hay que multiplicar el primer dato de salida por 12 y en el caso de los minutos habría que multiplicar el segundo dato de salida por 60.

Primeras 100 epochs.

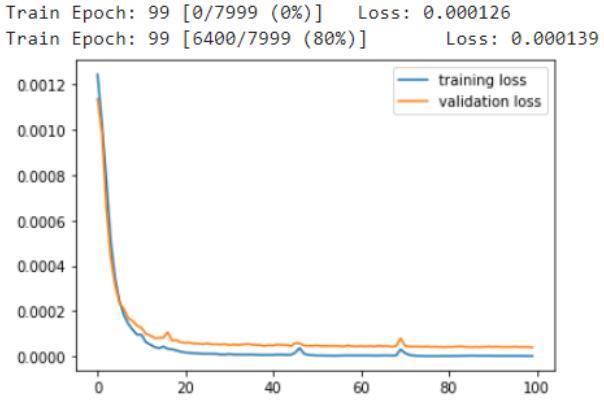

Fig. 11. Gráfico de evolución de la función de pérdida primeras 100 epochs. Fuente Propia.

En estos resultados calculamos el rendimiento de la predicción y vemos que con un 0.05 de margen de error que correspondería a un margen de 3 minutos, logramos que se consiga un 100 %.

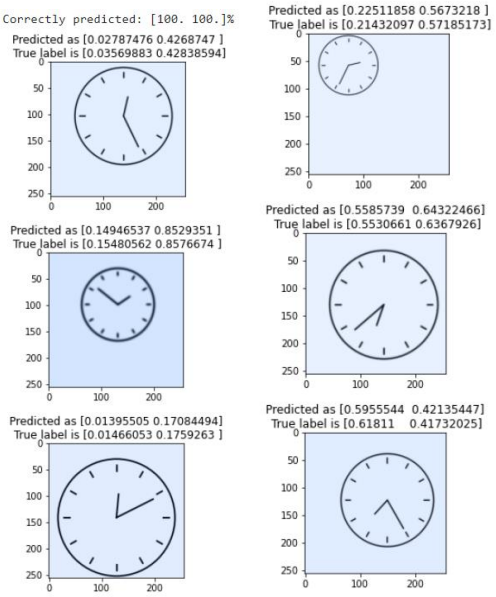

Fig. 12. Predicciones después de las primeras 100 epochs, con margen de error 0,05. Fuente Propia

Con un margen de error 0.01 equivalentes a un margen de 36 segundos en lo que se refiere a los minutos estaríamos en un 65 % de efectividad.

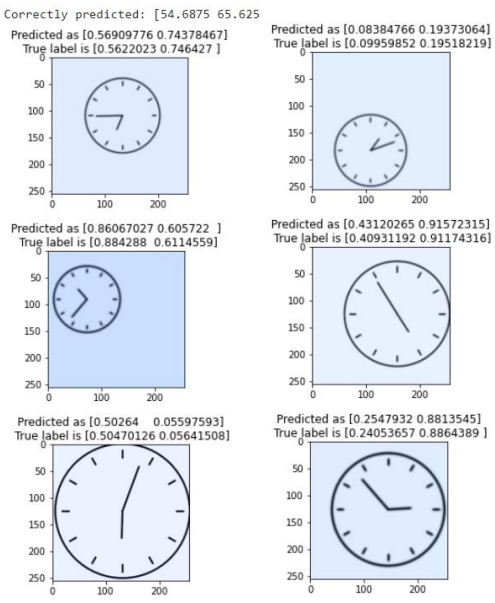

Fig. 13. Predicciones después de las primeras 100 epochs, con margen de error 0,01. Fuente Propia

Para ver cómo evoluciona el sistema se realizan 100 iteraciones más es y vemos cómo evoluciona.

Iteraciones de 100 a 200 (siguientes 100 Epoch).

En este caso vemos que la tendencia a la mejora de los datos de validación se mantiene, pero continua de una manera lenta, pero se valida que mejoran los resultados en lo que se refiere a exactitud y podemos obtener resultados algo más precisos aunque bajemos el margen de error.

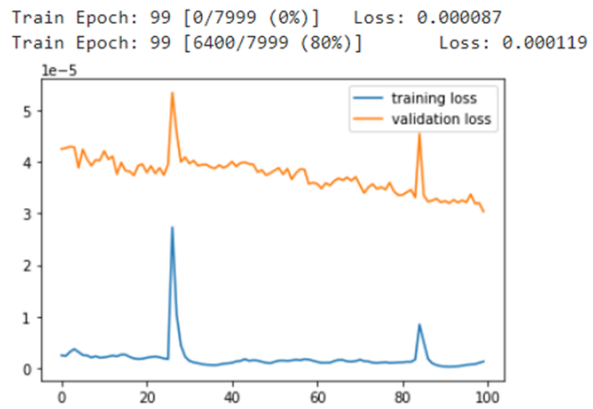

Fig. 14. Gráfico de evolución de la función de pérdida de 100 a 200 epochs. Fuente Propia.

Con un margen de 0.5 seguimos logrando el 100 % de efectividad.

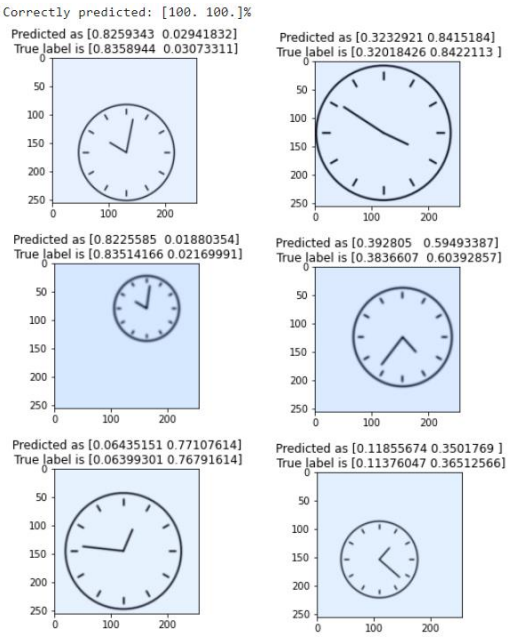

Fig. 15. Predicciones después de 100 a 200 epochs, con margen de error 0,05. Fuente Propia

Con una margen del 0.03 mantenemos el 100 % de exactitud de las predicciones.

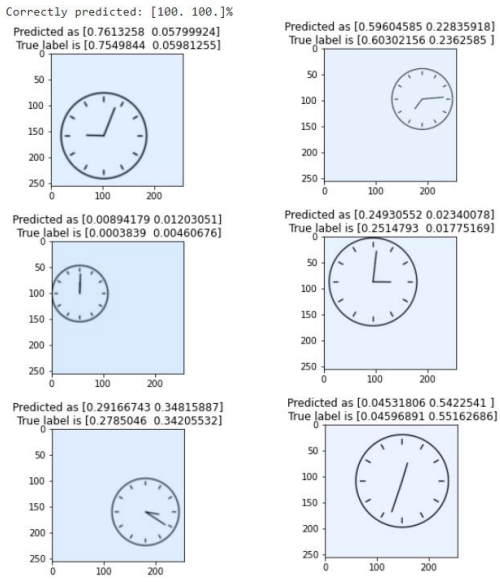

Fig. 16. Predicciones después de 100 a 200 epochs, con margen de error 0,03. Fuente Propia

Con un margen del 0.01 (error máximo de precisión de 36 segundos) no llegamos al 100 % pero estamos muy cerca acertando en más de un 90 % de las predicciones de los minutos, superior al 65 % que habíamos obtenido en las primeras 100 epoch.

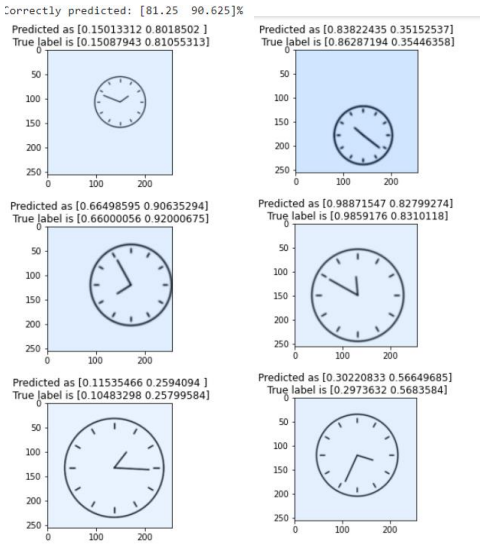

Fig. 17. Predicciones después de 100 a 200 epochs, con margen de error 0,01. Fuente Propia

Iteraciones de 200 a 300 (siguientes 100 Epoch).

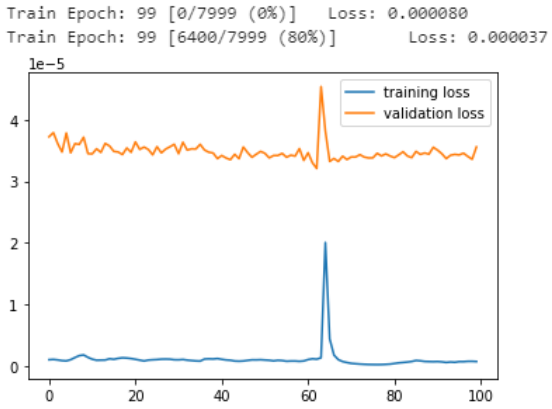

Fig. 18. Gráfico de evolución de la función de pérdida de 200 a 300 iteraciones. Fuente Propia

En esta serie de epoch vemos que realmente la mejora es mínima o simplemente el sistema ha dejado de aprender, los resultados de predicción también nos confirman que no se produce mejora en la exactitud de los resultados. Por lo que vemos que se acaba produciendo un overfitting en nuestro modelo, es decir un sobreajuste teniendo demasiado en cuenta los datos de entrenamiento sin llegar a mejorar el modelo.
Por lo que con precisiones del 0,01 se mantiene muy cerca del 90 % de precesión sin mejorar considerablemente los resultados.

3 Conclusión

Como conclusiones vamos a dividirlas en 4 secciones para así especificar las conclusiones de los objetivos, problemas encontrados, mejoras de futuro y conclusión general.

3.1 Objetivos.

A nivel de objetivos se han ido cumpliendo en su totalidad, aunque a nivel básico ya que en el ámbito que estaba enfocado este TFG simplemente pretendía ser un acercamiento al Deep learning.
El primer objetivo que es la obtención de conocimientos para utilizar este tipo de técnicas, este TFG me ha hecho ser consciente de que es un campo en el que se requiere una gran formación y que con un simple proyecto de TFG solo podemos aspirar a tener unos conocimientos limitados que aunque nos sirvan para implementar el trabajo realizado, si se quiere profundizar en el campo del Deep learning, es necesario dedicarse más a fondo y adquirir muchos más conocimientos para poder así llegar a mejores resultados en proyectos futuros.

En el objetivo de crear la red neuronal que fuera capaz de leer la hora se ha conseguido siempre teniendo en cuenta que se ha realizado simplificando su complejidad como se muestra en este trabajo.

El último objetivo que era evaluar las posibilidades de estas herramientas de reconocimiento de imágenes para ayudar a personas con problemas de visión, si que se llega a la conclusión que este tipo de técnicas o este trabajo puede ser la puerta de entrada para desarrollar herramientas que ayuden a estas personas, siempre y cuando tengamos en cuenta que necesitaran imágenes de la vida real con lo que eso sumara complejidad a los proyectos

3.2 Problemas encontrados.

En este trabajo tengo que decir que el principal problema, ha sido poder tener una base en la comprensión en materia de Deep learning, creo que este TFG hubiese sido más acorde con otro tipo de grado, más orientado al tratamiento de datos o con más orientación matemática, ya que se necesita una base de conocimientos adecuada para poder afrontar los objetivos y llegar a entender que está pasando en cada una de las operaciones que se han realizado.
Aun así, ha servido para tener una idea de las posibilidades que tienen estas técnicas de aprendizaje y poder aplicarlas.
Por lo que, si tuviera que recomendar a un estudiante, que realizase un trabajo similar, le recomendaría tener una base solida en Deep Learning antes de introducirse en este tipo de proyectos para que su realización sea más comoda.

3.3 Mejoras de Futuro.

En este trabajo que se enfoca en aprender la hora queda claro que en la vida real hay multitud de relojes de formas y modelos diferentes, por lo que nuestro modelo tendría que aprender a trabajar con imágenes reales y no solo generadas por nosotros mismos.
Otra de las posibles mejoras es incluir en esta predicción no solo a las imágenes de relojes analógicos sino que también incluir imágenes de relojes digitales, esto nos pondría en una complejidad añadida ya que antes de comenzar el proceso de predicción de la hora primero habría que decidir si se trata de una hora en formato digital o en formato analógico, posiblemente en este caso el modelo para el formato digital tendría que ser cambiado, ya no seria un modelo de regresión y este debería ser un modelo de clasificación de los diferentes dígitos de la imagen, que se deberían clasificar entre los posibles valores de 0 a 9.
Una vez se sepan leer las diferentes imágenes relacionadas con horas en la vida real, se abriría un campo para explorar las posibilidades de este trabajo y si puede llegar a ser útil esta información pensando que cada vez vivimos en un mundo más digital y conectado.

3.4 Conclusión general

Como hemos podido ver en este trabajo, ha quedado claro la potencia de las técnicas de deep learning en el aprendizaje y comprensión de imágenes, este es un caso particular simplemente con imágenes de relojes, pero podemos extrapolar las opciones a otro tipo de imágenes que queramos tratar siempre y cuando tengamos una fuente de imágenes que nos sirvan de aprendizaje y que estas estén correctamente identificadas y etiquetadas, aunque nos podemos encontrar con limitaciones como el que el número adecuado de imágenes no sea tratable o tengamos que necesitar modelos mucho más complejos para llegar a buenos resultados.
Hemos podido comprobar que para resultados correctos necesitamos una cantidad de imágenes adecuada, intentando procesar el mayor número posible, aunque siempre tendremos que obtener un compromiso entre la eficiencia del sistema y la precisión de los resultados, ya que, aunque dispongamos de un número infinito de ellas, tenemos que pensar como esto va a afectar al rendimiento del sistema, ya sea por el gran número de imágenes a utilizar o por que tengan una resolución demasiado alta. Hay que pensar que los diferentes cálculos que realiza nuestro modelo en cada iteración necesitan recursos y estos nunca son ilimitados.
En el diseño de nuestro modelo hay que realizar los diferentes cálculos para saber cuántos parámetros tiene nuestro sistema, un número demasiado grande podría no ser viable de gestionar por falta de recursos, y un número demasiado pequeño podría no ser suficiente para encontrar soluciones optimas, así que también hay que encontrar un modelo con un número de parámetros equilibrado.
Como conclusión general, he de decir que este caso práctico deja ver las grandes posibilidades que podemos obtener con técnicas de Deep learning y de la multitud de campos en la que estas soluciones pueden ser aplicadas.

AGRADECIMIENTOS

En primer lugar, agradecer la dedicación de mi tutor Dimosthenis Karatzas profesor de la Universidad Autónoma de Barcelona, por su entusiasmo a la hora de explicar una materia muchas veces complicada como es la IA a alguien como yo que intenta descubrirla.
Agradecer también como no, a mi familia, que tanta paciencia tienen conmigo, i me permiten compaginar estos estudios con la vida familiar.
Por último, dar las gracias a la vida, que me ha permitido tener esta segunda oportunidad de disfrutar de esta experiencia universitaria que ya no esperaba.

BIBLIOGRAFÍA

[1] Torra I Reventos, V. (2007). fonaments d'intel.ligència artificial (primera edicion ed.). Editorial UOC.Referència 2
[2] Torres I Viñals, J. (2020). Python deep learning - Introducción practica con Keras y TensorFlow 2. Marcombo..
[3] Illustration of transforms — Torchvision 0.10.0 documentation. (s. f.). Pytorch.Org. Recuperado 30 de agosto de 2021, de https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py
[4] Illustration of transforms — Torchvision 0.10.0 documentation. (s. f.). Pytorch.Org. Recuperado 30 de agosto de 2021, de https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py
[5] Platzi: Cursos online profesionales de tecnología. (s. f.). Platzi.Com. Recuperado 16 de julio de 2021, de https://platzi.com/tutoriales/1687-machine-learning/9832-que-es-el-learning-rate/
[6] Yani, M., Budhi Irawan, S. S. M., & Casi Setiningsih, S. M. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. Journal of Physics: Conference Series, 1201, 012052. https://doi.org/10.1088/1742-6596/1201/1/012052
[7] Conceptos básicos sobre redes neuronales. (s. f.). Vicente Rodríguez blog. Recuperado 16 de julio de 2021, de https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales#:%7E:text=Epoch,pasaran%20por%20la%20red%20neuronal
[8] M. (2021, 18 febrero). Overfitting. Qué es, causas, consecuencias y cómo solucionarlo. Grupo Atico34. https://protecciondatos-lopd.com/empresas/overfitting/