Ecuaciones diferenciales y reconocimiento facial

Iván Guerrón

Abstract*—En la era digital actual, el reconocimiento facial tiene varias aplicaciones, desde la seguridad hasta la autenticación y la inteligencia artificial. Sin embargo, enfrenta desafíos significativos, especialmente en entornos variables donde las condiciones de iluminación, expresiones faciales y fondos pueden afectar drásticamente la precisión del sistema. De esta manera, se propuso una solución basada en ecuaciones diferenciales ordinarias neuronales (NODEs). Estas redes ofrecen una alternativa para las muy utlizadas redes neuronales residuales, aunque ofrecen un acercamiento con menos pérdida de información, lo que resulta en la mejora sustancial en la generalización del modelo a través de diversas condiciones. Este estudio buscó avanzar en la eficacia y robustez del reconocimiento facial, contribuyendo así al desarrollo continuo de tecnologías cruciales en el ámbito de la visión por computadora, además de poner a prueba la arquitectura propuesta para resolver este tipo de problemas. La arquitectura NeuralODECNN fue entrenada en un conjunto de datos con imágenes de 105 personas. Después del entrenamiento y de la selección del mejor modelo bajo la métrica de exactitud, dicho modelo fue evaluado en un conjunto de prueba, y se obtuvo un 0.96 de exactitud, demostrando su alto nivel de generalización y su capacidad para cumplir con su principal tarea, el reconocimiento facial.

I. Introducción

La búsqueda de la mejora de los sistemas de reconocimiento facial sigue siendo un punto central de la investigación y el desarrollo. La evolución de las metodologías de aprendizaje profundo ha impulsado significativamente la precisión de los algoritmos de reconocimiento facial, pero persisten los desafíos, especialmente en la captura de la dinámica temporal inherente a las expresiones y transformaciones faciales. Este artículo explora un enfoque novedoso para abordar estos desafíos aprovechando el poder de las ecuaciones diferenciales ordinarias neuronales (NODEs) en el contexto del reconocimiento facial [1].

Este artículo profundiza en los fundamentos teóricos de las NODE y su aplicación a tareas de reconocimiento facial. Al tratar las capas de las redes neuronales como sistemas dinámicos continuos [1], los NODEs ofrecen una forma de integrar perfectamente la información temporal en el proceso de aprendizaje, mantienen un costo de memoria constante, y además ofrecen una alternativa para el uso de redes convolucionales residuales, que ya se han establecido como modelos importantes para la clasificación de imágenes. De esta manera, se pretende demostrar la eficacia de incorporar NODEs en arquitecturas de reconocimiento facial. Se presentan resultados experimentales sobre conjuntos de datos de referencia y se discuten las aplicaciones potenciales y las implicaciones de los NODEs en escenarios de reconocimiento facial del mundo real, incluyendo la precisión, la robustez del modelo propuesto y la mejora en términos de coste de memoria.

En resumen, este artículo presenta una exploración de aplicación de las ecuaciones diferenciales ordinarias neuronales para el avance de los sistemas de reconocimiento facial. Al reemplazar las capas residuales con NODEs se pretende lograr capacidades de reconocimiento facial más robustas y precisas, con implicaciones potenciales para una amplia gama de aplicaciones, como lo es la seguridad, entre otras.

II. Materiales y Métodos

A. Conjunto de datos

El conjunto de datos fue recuperado de Kaggle. Este mismo es llamado Pins Face Recognition, en donde existen 17534 imágenes de distintos tamaños de 105 personas [2]. En la Fig. 1 se encuentran algunos ejemplos de las imágenes del conjunto de datos

B. Ecuaciones diferenciales ordinarias neuronales

Tomemos una red neuronal residual. Al ver su comportamiento de la manera más simple, vemos que una red neuronal residual es obtener el siguiente estado oculto de una red sumando el estado actual y la capa neuronal (función) aplicada en el estado actual y los pesos de la red. Es decir, lo podemos ver de la siguiente manera:

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \theta_t) \tag{1}$$

Ahora, si prestamos atención podemos encontrar que existe una relación entre el método de Euler para el análisis de ecuaciones diferenciales ordinarias, y la definición de redes neuronales residuales. El método de Euler se lo puede ver como:

$$y_{k+1} = y_k + hf(x_k, y_k) \tag{2}$$

en cada paso, en donde y_k significa $y(x_k)$ y tenemos x_0 y y_0 dados, además de la longitud del paso dado por h. Si a h lo definimos como 1, entonces tenemos exactamente las mismas ecuaciones en 1 y en 2.

Pero, ¿y si queremos en vez de discretizar la función, hacerla continua? Tomemos pasos mucho más pequeños (en teoría llevemos su longitud al límite 0) y entonces tendríamos una capa de estados ocultos continua en vez de discreta. De esa manera tenemos:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta) \tag{3}$$

, y lo que queremos hacer es resolver la ecuación diferencial de la red neuronal para cada capa oculta [1].

Pero, ¿cómo entrenamos este tipo de redes?

Fig. 1: Ejemplos de imágenes del conjunto de datos.

$$L(\mathbf{z}(t_1)) = L\Big(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\Big) \\ = L\big(\text{ODESolver}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\big) \tag{4}$$

De esta manera, tenemos que obtener el gradiente a través de la solución de la ecuación diferencial. El primer método que viene a la cabeza es utilizar "backpropagation" a través de las operaciones del ODESolver. Sin embargo, esto provocaría problemas de memoria, al tener que guardar las operaciones intermedias de la resolución de la ecuación diferencial. Además, añade error numérico al gradiente.

Es de aquí que nace la propuesta de definir a lo que se conoce como "adjoint state", o estado adjunto, de la siguiente manera:

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)} \tag{5}$$

Sabemos que en redes neuronales estándares, el gradiente de una capa oculta depende del gradiente de la siguiente capa oculta bajo la regla de la cadena descrita a continuación:

$$\frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial \mathbf{z}_{t+1}} \frac{\partial \mathbf{z}_{t+1}}{\partial \mathbf{z}_t} \tag{6}$$

Usando lo anterior, y definiendo ε como el cambio de tiempo, podemos reescribir la ecuación 6 de la siguiente forma:

$$\frac{\partial L}{\partial \mathbf{z}(t)} = \frac{\partial L}{\partial \mathbf{z}(t+\varepsilon)} \frac{\partial \mathbf{z}(t+\varepsilon)}{\partial \mathbf{z}(t)} \tag{7}$$

Aplicando la definición de la derivada, la ecuación antes descrita y series de Taylor, se puede demostrar que:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} \tag{8}$$

Asimismo, podemos extendernos y demostrar que:

$$\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} \mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt \tag{9}$$

Una vez obtenidas nuestras ecuaciones, podemos differenciar automáticamente para definir estas dinámicas adjuntas ("adjoint dynamics"), y después llamar a nuestro ODESolver para poder integrar hacia atrás en el tiempo nuestra función. Sorpresivamente, esto significa un coste constante de memoria, ya que solo reconstruimos las capas ocultas al paso que corremos el sistema original atrás en el tiempo.

Finalmente, podemos reemplazar cualquier bloque residual con nuestro nuevo sistema.

C. Procesamiento de datos y creación del conjunto de datos experimental

Se redujo el conjunto de datos experimental a 86 imágenes por persona (105 personas en total) con el objetivo de lograr un equilibrio y evitar sesgos en el entrenamiento. Posteriormente, se dividió el conjunto en entrenamiento (70%), validación (20%) y pruebas (10%), resultando en 60 imágenes para entrenamiento, 17 para validación y 9 para pruebas. Después de esta división, las imágenes se ajustaron a un tamaño de 16×16 y se normalizaron para facilitar un entrenamiento más eficiente.

D. Configuracion de hiper parámetros e implementación

Para la configuración de hiperparámetros, el tamaño del lote seleccionado fue de 16 y el número de épocas fue 60. Además, se utilizó el optimizador Adamw [3], el cual se inicializó con una tasa de aprendizaje de 5×10^4, y con una tasa de decaimiento y un decaimiento de peso de 0.1 y 0.05, respectivamente. Este último fue seleccionado, ya que es computacionalmente eficiente y consume poca memoria, además de mejorar el redimiento de generalización del optimizador Adam. La implementación se realizó a través de Python 3.9 [4].

III. Resultados y discusión

A. Evaluación en el conjunto de entrenamiento y validación

Durante el entrenamiento y validación del modelo se pudieron obtener resultados que están resumidos en la Tabla I. En esta última se puede observar la exactitud, la precisión y otras métricas a lo largo de las épocas en las que se hizo un punto de chequeo. Como se puede observar, el modelo entrenado con 35 épocas fue el que mejores resultados arrojó (0.99 acc). Sin embargo, a lo largo del entrenamiento se fue perdiendo la exactitud del modelo.

TABLE I: Rendimiento del modelo.

Arquitectura	Lote	Época	Acc	Prec	Recall	F1
NeuralODECNN	32	5	0.98	0.98	0.98	0.98
		10	0.98	0.98	0.98	0.98
		15	0.87	0.88	0.87	0.87
		20	0.82	0.82	0.82	0.82
		25	0.90	0.90	0.91	0.90
		30	0.97	0.97	0.97	0.97
		35	0.99	1.00	1.00	1.00
		40	0.98	0.98	0.98	0.98
		45	0.93	0.93	0.93	0.93
		50	0.96	0.96	0.96	0.96
		55	0.88	0.88	0.88	0.88
		60	0.94	0.94	0.95	0.94

Los valores de las métricas son los obtenidos en la validación del modelo.

Fig. 2: Función de pérdida del modelo NeuralODECNN.

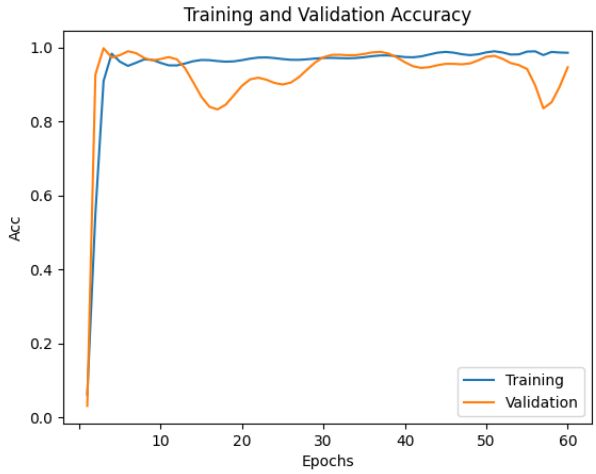

Fig. 3: Exactitud del modelo NeuralODECNN.

Así mismo podemos ver en las figuras presentadas ciertos comportamientos que se dieron a la hora del entrenamiento. En la Fig. 3 podemos ver como la métrica de evaluación (exactitud) converge rápidamente a valores cercanos a 1 para el conjunto de validación y para el conjunto de entrenamiento. Sin embargo, una vez llegado a dicho punto, la curva de validación empieza a tener más inestabilidad que aquella de entrenamiento, aunque ambas se mantienen en cierto rango de exactitud.

Por otro lado, en la Fig. 2 tenemos las curvas de la funcion de pérdida durante el entrenamiento con las 60 épocas. En esta gráfica podemos ver como antes de las 40 épocas de entrenamiento las curvas para entrenamiento y validación se separan y la pérdida para el conjunto de validación empieza a crecer gradualmente.

Estos comportamientos en ambas figuras y en la tabla presentada, muestran que el modelo es capaz de ejecutar su objetivo con éxito (reconocimiento facial) y tiene un alto nivel de generalización (siempre y cuando el modelo haya sido entrenado con un número de épocas limitado). No obstante, también se demuestra la falta de estabilidad en la arquitectura.

Para comprobar el nivel de generalización de la arquitectura, se escogió el mejor modelo en términos de epocas de entrenamiento. Esta selección dió como mejor modelo a aquel entrenado con 35 épocas, ya que este alcanzó una exactitud del 0.99 al ser sometido a reconocimiento facial. Asimismo, este modelo fue seleccionado para ser evaluado en el conjunto de prueba.

B. Evaluación en el conjunto de pruebas

El mejor modelo seleccionado, entrenado con 35 épocas, fue sometido a evaluación bajo el conjunto de pruebas. Este conjunto es un subconjunto del conjunto de datos totalmente desconocido para el modelo. De esta manera, el modelo dió a

Fig. 4: Ejemplos de reconocimiento en el conjunto de prueba.

conocer su nivel de generalización y su habilidad para realizar reconocimiento facial.

El modelo, al ser evaluado de esta manera, obtuvo como resultado un 0.98 de exactitud y demostró su capacidad de abstracción y aprendizaje. Además en la Fig. 4 se pueden ver ejemplos de imágenes que fueron reconocidas de manera adecuada por el modelo, con la predicción como título de cada imagen.

IV. Conclusiones

Este estudio propuso una solución para el reconocimiento facial basada en las ecuaciones diferenciales ordinarias neuronales (NODEs). La arquitectura NeuralODECNN, entrenada con un conjunto de datos de 105 personas, demostró un alto nivel de generalización, alcanzando una exactitud del 0.98 en el conjunto de prueba. Sin embargo, se observó que la estabilidad de la arquitectura fue un área de preocupación, ya que se evidenció poca estabilidad y sobreajuste en épocas posteriores durante el entrenamiento. Como trabajo futuro, se realizará una introspección de la arquitectura para intentar dar con una solución más estable.

References

[1] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[2] Burak, "Pins face recognition," 2020 [Online]. [Online]. Available: https://www.kaggle.com/datasets/hereisburak/pins-face-recognition

[3] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[4] Python Core Team, *Python 3.9: A dynamic, open source programming language.*, Python Software Foundation, 2020. [Online]. Available: https://www.python.org/.