CLASIFICACIÓN: Monedas Romanas



UNIVERSIDAD
COMPLUTENSE
MADRID

Miembros:
Pablo Tejero Delgado

Sara Medina

Iria Barrios

Mireia Suso

Grupo Finis Terrae

Mayo 2026

1. Introducción y Objetivos..... 3
2. Metodología y Dataset.....3
3. Modelos Implementados.....3
4. Resultados y Discusión.....6
5. Conclusiones y Trabajo Futuro..... 11
6. Aplicación *Clasificador de Monedas*……..…13
7. Bibliografía. Artículo de referencia..……...…….16

1. Introducción y Objetivos

La numismática antigua, en particular la romana, presenta grandes desafíos para su clasificación manual. El desgaste de las piezas a lo largo de los siglos, las variaciones de los bordes y la complejidad de sus relieves hacen que la identificación sea una tarea dificil y reservada a expertos. Este proyecto se centra en buscar automatizar el reconocimiento de estas monedas.

El **objetivo general** es desarrollar y evaluar empíricamente diferentes arquitecturas de Deep Learning para clasificar monedas romanas. Como **objetivos específicos**, se plantea:

1. Comparar el rendimiento de clasificadores tradicionales (MLP) frente a arquitecturas recientemente nuevas(KAN).
2. Evaluar distintos extractores de características: convolucionales desde cero, modelos de reconstrucción (Autoencoders) y aprendizaje por transferencia (ResNet18 con Atención Espacial).

2. Metodología y Dataset

Para el entrenamiento se ha utilizado un subconjunto del **Dataset RRCD** (Roman Republican Coin Dataset), seleccionando específicamente **10 clases** de monedas.

- **Preprocesamiento:** Las imágenes se han redimensionado a 128x128 píxeles para optimizar la carga computacional sin perder los detalles clave del relieve.
- **Distribución:** El conjunto de datos se dividió en carpetas de entrenamiento (Train), validación (Val) y test. Las imágenes se procesaron en lotes (*batch size*) de 16.
- **Configuración del entrenamiento:** Se estableció un total de 15 épocas para todos los modelos, utilizando el optimizador Adam con una tasa de aprendizaje de 1e-3, un decaimiento de pesos de 1e-4, y un planificador de tasa de aprendizaje que reduce el LR cada 5 épocas. El entrenamiento se aceleró mediante el uso de GPU.

3. Modelos Implementados

El núcleo de la investigación consistió en desarrollar y comparar cinco vías de experimentación:

3.1.1. Arquitectura CNN + MLP

Esta variante representa el paradigma clásico del aprendizaje profundo para clasificación de imágenes. Recibe como entrada tensores de tamaño 128 x 128 x 3 (imágenes RGB normalizadas) y se divide en dos bloques principales:

- **Extractor de Características:** Compuesto por una secuencia de cuatro bloques convolucionales. Cada bloque aplica de forma sucesiva:
    - Una capa de convolución bidimensional (Conv2D) con un tamaño de kernel de 3 x 3 y *padding* de 1 para mantener las dimensiones espaciales.

    - Normalización por lotes (BatchNorm2D) para estabilizar y acelerar el entrenamiento.
    - Una función de activación no lineal ReLU.
    - Una operación de submuestreo MaxPooling2D con kernel y *stride* de 2 x 2, encargada de reducir a la mitad la resolución espacial de los mapas de características.
- A medida que la imagen atraviesa estos cuatro bloques, su dimensionalidad espacial se reduce progresivamente (128 x 64 x 32 x 16 x 8), mientras que la profundidad de los canales de extracción aumenta (32 x 64 x 128 x 256). El tensor resultante a la salida de este módulo tiene unas dimensiones de 256 x 8 x 8.
- **Clasificador Final (MLP):** El tensor tridimensional extraído se aplana (*flatten*) en un vector unidimensional de 16384 características (256 x 8 x 8). Este vector alimenta una red neuronal densa tradicional compuesta por:
    - Una capa lineal oculta que reduce la dimensionalidad de 16384 a 512 neuronas, seguida de una activación ReLU.
    - Una capa de regularización Dropout con una probabilidad de desconexión del **50%** ($p = 0.5$) para mitigar el sobreajuste (*overfitting*).
    - Una capa lineal de salida que proyecta las 512 características al número total de clases del conjunto de datos, generando los *logits* finales.

3.1.2. Arquitectura CNN + KAN

Para evaluar la viabilidad de las redes Kolmogorov-Arnold en el procesamiento de características puras de imagen, se ha diseñado una segunda arquitectura que sustituye el clasificador denso tradicional por capas KAN.

- **Extractor de Características Modificado (CNN + Global Pooling):** Se mantiene la base convolucional de cuatro bloques descrita en el modelo anterior (Línea Base), la cual transforma la imagen original en un mapa de características espaciales de dimensiones 256 x 8 x 8. Sin embargo, para evitar una carga enorme de parámetros en la capa KAN, se introduce una operación de *Global Average Pooling* mediante la capa AdaptiveAvgPool2d((1, 1)). Esta operación comprime la información espacial de cada canal promediando sus valores, transformando el tensor de 256 x 8 x 8 en un tensor de 256 x 1 x 1$. Tras aplanar el resultado (*flatten*), se obtiene un vector denso de **256 características** (en contraposición a las 16384 del modelo anterior), logrando una representación compacta y manejable computacionalmente.
- **Clasificador Basado en KAN:**
  El vector de 256 características es procesado por un bloque de clasificación secuencial compuesto íntegramente por capas basadas en el teorema de representación de Kolmogorov-Arnold (KANLinear):
    1. **Capa KAN Oculta:** Proyecta el vector de entrada de 256 dimensiones a un espacio latente de 64 dimensiones.
    2. **Normalización de Capa (LayerNorm):** A diferencia de las redes perceptrón tradicionales que suelen emplear normalización por lotes (BatchNorm), se ha demostrado que las representaciones en las redes KAN se estabilizan mejor

mediante *Layer Normalization*, aplicándose a las 64 características intermedias.

3. **Capa KAN de Salida:** Proyecta las 64 características al número final de clases del conjunto de datos para generar los *logits* de clasificación..

3.2. Modelos basados en Autoencoders (Autoencoder + MLP)

Esta variante emplea el aprendizaje multitarea (*Multi-Task Learning*) obligando a la red a resolver dos problemas simultáneamente: clasificar la moneda y reconstruir su imagen. Esto fuerza al modelo a aprender representaciones más robustas. El sistema se divide en tres módulos:

- **Codificador (*Encoder*):** Extrae las características reduciendo la imagen mediante convoluciones con un paso de 2, prescindiendo del MaxPooling. Comprime las imágenes de entrada de 128 x 128 x 3 hasta un espacio latente de 256 x 8 x 8.
- **Clasificador (*MLP*):** Aplana el tensor latente y lo procesa mediante una capa densa de 512 neuronas con un Dropout del 40% para evitar el sobreajuste, generando finalmente las predicciones de clase.
- **Decodificador (*Decoder*):** Toma el mismo espacio latente y reconstruye la imagen original mediante convoluciones transpuestas (ConvTranspose2d). Su capa final usa una activación Sigmoid para mantener los píxeles generados en un rango de [0, 1].

Función de Pérdida Múltiple

La innovación clave de este modelo es su función de coste, que combina la Entropía Cruzada para la clasificación L_cls y el Error Cuadrático Medio para la reconstrucción L_rec:

L_total = L_cls * 10 + L_rec

Al multiplicar el error de clasificación por 10, se fuerza al optimizador a priorizar estrictamente la exactitud predictiva. La reconstrucción visual actúa únicamente como una tarea secundaria que guía a la red para que no pierda información estructural y topológica relevante durante la extracción de características.

4.2.2. Arquitectura Autoencoder + KAN

Esta variante combina el aprendizaje de dos tareas a la vez (clasificar y reconstruir la moneda) con la flexibilidad de las nuevas redes KAN. El sistema se divide en tres partes principales:

- **Codificador y Decodificador:** Funcionan exactamente igual que en el modelo anterior (Autoencoder + MLP). El codificador comprime la imagen de la moneda y el decodificador intenta redibujarla. Se mantiene la misma fórmula que da mucha más importancia a clasificar bien la moneda que a dibujarla perfectamente:
  L_total = L_cls * 10 + L_rec

- **Reducción de datos (Global Pooling):** Si pasáramos toda la información gráfica directamente a la red KAN, el modelo colapsaría por tener demasiados parámetros. Por eso, usamos una capa AdaptiveAvgPool2d((1, 1)) que resume y comprime los mapas de características en un vector pequeño y manejable de solo 256 valores.
- **Clasificador KAN:** En lugar de usar una red tradicional (MLP), pasamos esos 256 valores por un bloque de clasificación basado en KAN (KANLinear):
    - **Capa KAN Oculta:** Reduce la información de 256 a 64 características más precisas.
    - **Normalización (LayerNorm):** A diferencia de otras redes, las capas KAN funcionan mucho mejor y son más estables usando *Layer Normalization* en lugar de *Dropout*.
    - **Capa KAN de Salida:** Coge esas 64 características finales y decide a cuál de las 10 clases pertenece la moneda.

Objetivo de este modelo: Con este diseño buscamos comprobar si obligar a la IA a "entender" la forma 3D de la moneda (mediante la reconstrucción) ayuda a darle a la red KAN la estabilidad y seguridad que le falta cuando trabaja solo con imágenes normales.

3.3. Modelo de Vanguardia: ResNet 18 (CoinNet)

Para tener un modelo potente con el que comparar, utilizamos una red neuronal llamada ResNet18, que ya venía entrenada previamente con millones de imágenes (ImageNet). Decidimos "congelar" la primera parte de esta red; es decir, no la volvimos a entrenar. Esto nos sirvió para ahorrar tiempo de ordenador y aprovechar las formas básicas que la red ya sabía reconocer.

Además, le añadimos un módulo de "Atención Espacial". Básicamente, es un mecanismo matemático que calcula a qué partes de la imagen hay que darles más importancia. De esta forma, ayudamos a la red a centrarse en los detalles útiles de la moneda y a ignorar el fondo.

Finalmente, usamos una técnica de regularización (*Dropout* del 30%) para evitar que el modelo se aprenda las fotos de memoria, y pasamos la información por una última capa sencilla (capa lineal) que es la que decide a qué clase de moneda pertenece la imagen.

4. Resultados y Discusión

Tras el entrenamiento durante 15 épocas, los modelos mostraron resultados extraordinarios, evidenciando que las redes neuronales profundas pueden manejar la complejidad visual y el desgaste de la numismática romana con gran soltura.

4.1. Resultados (Precisión / Accuracy)

El rendimiento global de las arquitecturas implementadas desde cero alcanzó métricas muy competitivas:

- Autoencoder + MLP: 98,77% (Rendimiento superior global).

- CNN + KAN: 97,69%
- CNN + MLP: 97,61%
- Autoencoder + KAN: 96,89%

Por otro lado, usamos la red ResNet18 como modelo de referencia. Al compararla con nuestros resultados, pudimos confirmar que los modelos que hemos creado nosotros mismos desde cero son igual de buenos y precisos que una red profesional.

4.2. Análisis de Resultados e Interpretación Profunda

- **El Análisis de Confianza (MLP vs KAN)**: El hallazgo más importante de este trabajo no radica únicamente en la precisión global, sino en la "seguridad" o confianza matemática de las predicciones. Al evaluar clasificaciones exitosas (por ejemplo, identificando el *Denario republicano tipo cautivo + trofeo* o el de *L. Valerius Flaccus*), se observó una divergencia crítica:
    - El modelo con clasificador tradicional (CNN + MLP) demostró una altísima seguridad, prediciendo la clase correcta con un 91,85% de confianza y dejando a las demás clases con probabilidades casi nulas.
    - Por el contrario, la red que integraba el clasificador matemático novedoso (CNN + KAN) logró acertar la moneda, pero lo hizo con una confianza de apenas el 57,36%, mostrando grandes dudas internas y asignando un significativo 31,8% de probabilidad a que la moneda fuera un Denario de C. Fonteius.
    - *Interpretación:* Las redes KAN son extremadamente flexibles gracias a sus funciones *spline* aprendibles. Sin embargo, en dominios de alta dimensionalidad y ruido como las imágenes de monedas antiguas, esta misma flexibilidad provoca que sus límites de decisión sean menos precisos y tajantes que los hiperplanos rígidos que traza una capa densa (MLP).
- **El poder de la reconstrucción (Autoencoders):** El éxito rotundo del modelo Autoencoder + MLP (98,77%) demuestra un concepto clave en Deep Learning. Obligar a la red a reconstruir visualmente la moneda (aunque sea con cierta pérdida de calidad) actúa como un poderoso regularizador. Fuerza al "espacio latente" de la red a no memorizar atajos inútiles, sino a retener la topografía y los volúmenes reales de la pieza, facilitando enormemente el trabajo del clasificador MLP final.
- **El rol de ResNet18 y la Atención Espacial:** La evaluación de la arquitectura ResNet18 confirmó que utilizar un extractor de características genérico e insertarle un mapa de Atención Espacial es sumamente efectivo para ignorar el fondo y centrarse en el relieve de la acuñación. No obstante, este experimento demostró que el verdadero cuello de botella decisivo está en la "cabeza" de la red (el clasificador final): por muy potente que sea el cuerpo extractor (como ResNet18), el sistema necesita un clasificador final robusto (MLP) para ofrecer la certidumbre que requiere un análisis numismático riguroso.

4.3. Análisis de la Matriz de Confusión

Para comprender en profundidad el comportamiento de los clasificadores más allá de la precisión global (*accuracy*), se calculó y analizó la matriz de confusión de los modelos. Esta herramienta gráfica resulta fundamental en numismática, ya que nos permite visualizar no solo los aciertos (representados en la diagonal principal), sino también la naturaleza exacta de los errores (falsos positivos y falsos negativos) y entre qué clases específicas se producen.

Al observar la matriz, se extraen las siguientes conclusiones clave:

- **Alta densidad en la diagonal principal:** Consistente con las métricas de precisión superiores al 96%, la inmensa mayoría de las predicciones se concentran en la diagonal, confirmando que las redes (especialmente la combinación Autoencoder + MLP) logran separar correctamente las 10 clases de denarios sin sesgos graves hacia una clase mayoritaria.
- **Errores no aleatorios (Similitud iconográfica):** Los valores residuales que caen fuera de la diagonal no se distribuyen al azar. Las confusiones tienden a concentrarse entre monedas que comparten patrones iconográficos similares, relieves parecidos o que sufren tipos de desgaste análogos por la acuñación.
- **Reflejo gráfico de la falta de confianza:** La matriz de confusión ilustra perfectamente las dudas matemáticas detectadas en la red KAN. Mientras que la matriz del modelo basado en MLP muestra celdas externas casi a cero (decisiones tajantes), la matriz de los modelos menos seguros refleja una ligera dispersión. Un claro ejemplo de esto es la frontera de decisión entre el *Denario de L. Valerius Flaccus* y el *Denario de C. Fonteius*; al presentar características latentes cercanas, los modelos con menor confianza (o sin la regularización del autoencoder) "reparten" sus predicciones erróneas entre estas celdas adyacentes.

En definitiva, la matriz de confusión demuestra que cuando el modelo falla, lo hace con una "lógica visual" comprensible para un experto numismático, confundiendo piezas objetivamente similares, lo cual valida que la red neuronal está extrayendo características reales de las monedas y no basándose en artefactos aleatorios del fondo de la imagen.

4.4. Prueba de Estrés: Evaluación con una moneda al azar

Para comprobar la robustez de los modelos más allá de las métricas estandarizadas del conjunto de *Test*, se introdujo un experimento adicional: probar las redes con una imagen de una moneda extraída al azar (un caso de uso real, no visto durante el entrenamiento y con condiciones de desgaste e iluminación variables).

Al pasar esta misma moneda por las arquitecturas convolucionales básicas, **tanto el modelo CNN + MLP como el CNN + KAN fallaron en su predicción**. Sin embargo, el análisis de *cómo* fallaron resulta tremendamente revelador y confirma la hipótesis principal del estudio sobre el comportamiento de los clasificadores:

1. **El fallo por exceso de confianza (CNN + MLP):** La red tradicional falló, prediciendo que se trataba de un *Denario republicano tipo cautivo + trofeo*. Lo grave

desde el punto de vista de implementación no es el fallo en sí, sino que lo hizo con una confianza altísima (**91,85%**). Las redes con MLP tienden a ser "sobreconfiadas" (overconfident); cuando se equivocan ante datos anómalos o ruidosos, lo hacen con total seguridad, lo que en una aplicación real daría por válida una clasificación errónea sin pedir revisión humana.

2. **El fallo con incertidumbre (CNN + KAN):** Por su parte, la red con el clasificador KAN también falló, creyendo que la moneda era un *Denario de L. Valerius Flaccus*. Sin embargo, la ventaja matemática de su flexibilidad quedó patente en su matriz de probabilidad: predijo este error con una confianza muy baja (**57,36%**) , repartiendo casi un 32% de probabilidad a otra clase totalmente distinta (*Denario de C. Fonteius*).

Interpretación del fallo:

El hecho de que ambas CNN básicas fallaran ante una moneda aleatoria demuestra sus **limitaciones de generalización**. Las convoluciones simples son susceptibles de confundirse ante cambios de rotación, iluminación o grados de corrosión imprevistos en la pieza.

Es precisamente la constatación de este tipo de fallos la que **justifica y da valor a las arquitecturas superiores** desarrolladas en este trabajo:

- Explica por qué el uso de **Autoencoders** (que obligan a la red a entender la forma 3D de la moneda) es necesario para que el sistema no se confunda ante el ruido.
- Demuestra la necesidad de recurrir a redes de vanguardia como **ResNet 18**, capaces de extraer mapas de características mucho más profundos y aplicar **Atención Espacial** para aislar únicamente el relieve útil de la moneda, ignorando la degradación o el fondo.

4.5. El límite de la reconstrucción: Análisis del error en los Autoencoders

Aunque las arquitecturas basadas en Autoencoders demostraron ser las más robustas del estudio (alcanzando hasta un 98,77% de precisión en el conjunto de test), someterlas a la "prueba de estrés" con una imagen de una moneda aleatoria —fuera de la distribución original de entrenamiento (*out-of-distribution*)— revela que tampoco son infalibles.

Como se observa en los resultados empíricos, al menos una de las variantes del Autoencoder también incurrió en un error de clasificación ante esta pieza desconocida. Sin embargo, a diferencia de las redes CNN puras (donde el motivo del error permanece oculto en las capas convolucionales), el modelo Autoencoder nos proporciona una herramienta diagnóstica visual invaluable: **la imagen reconstruida**.

Al analizar el fallo del Autoencoder, se extraen las siguientes conclusiones:

1. **Colapso del espacio latente:** Cuando la red se enfrenta a una moneda con un desgaste, pátina, centrado o condiciones de iluminación radicalmente distintas a las de su conjunto de entrenamiento, el *Encoder* es incapaz de mapear esas características correctamente en el espacio latente.

2. **Reconstrucción deficiente:** Como consecuencia matemática directa, el *Decoder* no tiene información válida para "redibujar" la moneda. La imagen que reconstruye resulta ser una mezcla borrosa de características genéricas o distorsionadas que intenta encajar a la fuerza en los patrones que sí conoce.
3. **Efecto dominó en la clasificación:** Dado que la pérdida del modelo (*Loss*) obliga a la red a basar su clasificación en la topografía latente de la moneda, si la reconstrucción visual falla estrepitosamente, el clasificador final (sea MLP o KAN) recibe información defectuosa y, por ende, toma una decisión equivocada.

Interpretación: Este fenómeno es fascinante desde el punto de vista del Machine Learning, ya que demuestra que la Inteligencia Artificial no está simplemente "memorizando" píxeles, sino intentando comprender la geometría de la moneda. Si la pieza es demasiado anómala, el Autoencoder falla en su intento de comprensión visual, lo que arrastra al clasificador al error. Esto subraya la necesidad futura de aplicar técnicas de *Data Augmentation* (aumento de datos) mucho más agresivas durante el entrenamiento para enseñar a la red a reconstruir piezas bajo cualquier condición de degradación posible.

4.5. Prueba de Estrés y el Triunfo Definitivo del Autoencoder + MLP

Para comprobar la verdadera robustez de los modelos más allá de las métricas de laboratorio, se sometieron las arquitecturas a una "prueba de estrés" utilizando una imagen de una moneda aleatoria (un dato *out-of-distribution*, con condiciones de iluminación, fondo y desgaste no vistas en el entrenamiento). Los resultados de este experimento revelaron la verdadera jerarquía de los modelos desarrollados.

Como se analizó en el apartado anterior, las arquitecturas convolucionales simples (CNN + MLP y CNN + KAN) fracasaron en esta prueba al ser engañadas por el ruido visual. Asimismo, al observar los modelos de reconstrucción, la variante **Autoencoder + KAN** también se equivocó; ante un dato tan anómalo, la flexibilidad matemática de KAN jugó en su contra, produciendo una reconstrucción deficiente y arrastrando al clasificador al error.

Sin embargo, el hallazgo más destacable de todo el experimento es que **el modelo Autoencoder + MLP logró clasificar la moneda aleatoria de forma correcta**. Este éxito empírico ante un dato adverso no es casualidad y demuestra teóricamente por qué esta ha sido la mejor arquitectura del estudio:

El modelo combina lo mejor de ambos mundos. Al estar entrenado con una fuerte penalización en la pérdida de clasificación apoyada por la reconstrucción, el *Encoder* actúa como un filtro implacable. Ignora el ruido del fondo o la corrosión superficial y extrae únicamente el "mapa topográfico" o volumen real de la moneda en su espacio latente.

Mientras que KAN se desestabilizó ante esta moneda desconocida, el clasificador tradicional **MLP** hizo valer la rigidez y seguridad de sus capas densas. Al recibir características geométricas puras y libres de ruido por parte del Autoencoder, el MLP trazó una frontera de decisión impecable y segura, logrando el acierto.

Interpretación final: Esta prueba de estrés valida de manera irrefutable la conclusión central del proyecto. Para resolver problemas visuales complejos en la numismática antigua, no basta con redes de clasificación directa (como las CNN puras). Es estrictamente necesario obligar a la IA a "entender" la geometría del objeto (mediante Autoencoders) y, hoy por hoy, confiar esa decisión final a clasificadores matemáticamente sólidos, rígidos y seguros como el Perceptrón Multicapa (MLP).

5. Conclusiones y Trabajo Futuro

5.1. Resultados

Modelo	**Confianza TRAIN**	**Confianza TEST**
CNN-KAN	97,69 %	40,66 %
CNN-MLP	97,61 %	75,56 %
Autoencoder-KAN	96,89 %	33,65 %
Autoencoder-MLP	98,77 %	51 %
Resnet18	95,58 %	99,20 %

5.2. Conclusiones Principales

1. **Viabilidad Absoluta en Humanidades Digitales:** El Deep Learning ha demostrado ser una herramienta excepcionalmente madura para la numismática clásica. Haber superado el 96% de precisión global en todos los enfoques estudiados (con solo 15 épocas) confirma que las redes neuronales pueden automatizar tareas de clasificación que tradicionalmente requerían décadas de especialización humana.
2. **La Superioridad del Autoencoder + MLP:** El experimento de la moneda aleatoria (prueba de estrés) coronó a esta arquitectura como la ganadora indiscutible (98,77% de precisión). Al combinar la reconstrucción (que actúa como un filtro que ignora el ruido y comprende la topografía 3D de la pieza) con la robustez matemática del MLP, se obtiene un modelo capaz de generalizar y acertar incluso ante datos fuera de su distribución original (*out-of-distribution*).
3. **MLP vs KAN (Certeza frente a Incertidumbre):** Aunque las redes KAN son matemáticamente innovadoras, su aplicación directa como clasificadores de imágenes numismáticas genera inestabilidad. Como reflejó la matriz de confusión y la prueba de estrés, KAN duda entre clases visualmente similares, ofreciendo distribuciones de probabilidad erráticas (57,36% de confianza). Por el contrario, las capas densas clásicas (MLP), con su contundente 91,85% de seguridad, siguen siendo el estándar indiscutible para tomar decisiones certeras en la Inteligencia Artificial visual.

4. **Eficiencia y Transfer Learning:** El uso de arquitecturas de vanguardia preentrenadas como ResNet18, combinadas con mecanismos de *Atención Espacial*, resulta ser una estrategia altamente eficiente. Permite alcanzar el estado del arte aprovechando el conocimiento previo de la red, mitigando en parte la necesidad de diseñar extractores de características desde cero.

5.3. Limitaciones del Estudio y Líneas Futuras

- **Limitaciones:**
    - **Escalabilidad y Volumen de Clases:** El estudio se ha centrado exitosamente en un subconjunto de 10 clases de denarios para comparar arquitecturas. Sin embargo, una aplicación numismática universal requeriría escalar el número a cientos de variantes, emperadores y cecas distintas, lo que introduciría un severo problema de desbalanceo de clases y complejidad.
    - **Coste Computacional y Temporal:** A la limitación anterior se suma el alto coste de procesamiento. El entrenamiento de estos modelos desde cero en hardware convencional requirió **más de 5 horas** de cómputo ininterrumpido. Escalar el dataset a miles de imágenes y cientos de clases haría inviable el entrenamiento en equipos domésticos, requiriendo infraestructura especializada.
- **Líneas Futuras:**
    - **Data Augmentation Extremo:** Se propone aplicar técnicas de aumento de datos más agresivas durante el entrenamiento para simular digitalmente distintos grados de corrosión, pátina, descentrado e iluminación asimétrica.
    - **Arquitecturas Híbridas:** Sería de gran interés investigar modelos que no descarten la innovación de KAN, sino que la reubiquen. Por ejemplo, utilizando funciones *spline* aprendibles en las capas intermedias para mejorar la interpretabilidad de las características, pero manteniendo un MLP rígido al final para garantizar la seguridad de la clasificación.
    - **Migración a Cloud Computing:** Para superar la barrera de las 5 horas de entrenamiento y permitir la escalabilidad del proyecto, el siguiente paso lógico sería la migración del entrenamiento a clústeres en la nube (ej. Google Colab Pro o AWS) con aceleración multi-GPU.

6. Aplicación *Clasificador de Monedas*

Link de la aplicación: https://5jypwixzs8whku5wpr9jyf.streamlit.app

6.1 Desarrollo de la aplicación

Para la implementación de la aplicación de clasificación de monedas romanas se optó por una solución basada en tecnologías web, con el objetivo de facilitar su accesibilidad multiplataforma (iOS y Android) sin necesidad de desarrollar aplicaciones nativas específicas para cada sistema operativo.

Inicialmente valoramos el desarrollo de una aplicación móvil nativa. Sin embargo, esta opción implicaba una mayor complejidad técnica, especialmente en la integración del modelo de redes neuronales entrenado en Python. Por este motivo, se decidió utilizar **Streamlit**, una herramienta que permite crear aplicaciones web interactivas directamente a partir de código Python.

Esta elección permitió:

- reutilizar directamente el modelo entrenado (.pth)
- simplificar la interfaz de usuario
- evitar el desarrollo de un backend adicional
- compatibilidad con cualquier dispositivo con navegador

6.2 Modelo de inteligencia artificial

El núcleo de la aplicación está constituido por una red neuronal convolucional basada en la arquitectura ResNet18. Esta arquitectura, perteneciente a la familia de las *Residual Networks*, introduce conexiones residuales que permiten el flujo directo de la información entre capas, lo que facilita el entrenamiento de redes profundas y reduce problemas como el desvanecimiento del gradiente.

En el presente proyecto, la arquitectura ResNet18 se ha empleado como base para la extracción de características visuales de las imágenes de monedas. A partir de esta estructura inicial, se han incorporado modificaciones con el objetivo de mejorar el rendimiento del modelo en la tarea específica de clasificación. En concreto, se han añadido bloques residuales adicionales que incrementan la capacidad de representación del modelo, así como un mecanismo de atención espacial que permite al sistema centrarse en las regiones más relevantes de la imagen, ignorando información irrelevante del fondo.

El modelo resultante fue entrenado previamente utilizando el conjunto de datos generado, y posteriormente almacenado en un archivo en formato .pth, que contiene tanto los pesos aprendidos como la correspondencia entre las clases y sus índices. Este archivo constituye el elemento central que se carga en la aplicación para realizar las predicciones en tiempo real.

6.3 Estructura del proyecto

Para facilitar la gestión del código y su posterior despliegue, el proyecto se organizó en un repositorio de GitHub. En dicho repositorio se incluyeron los archivos necesarios para la ejecución de la aplicación, destacando principalmente el archivo app_streamlit.py, que contiene la lógica de la interfaz y la integración con el modelo de inteligencia artificial.

Asimismo, se incluyó un archivo requirements.txt con las dependencias necesarias para la ejecución del proyecto, tales como Streamlit, PyTorch y otras librerías auxiliares. También se incorporó un archivo README.md con una descripción general del proyecto y un archivo .gitignore destinado a excluir del repositorio elementos innecesarios, como entornos virtuales o conjuntos de datos de gran tamaño.

El control de versiones del proyecto se llevó a cabo mediante el uso de Git. En primer lugar, se inicializó un repositorio local en la carpeta del proyecto y se añadieron todos los archivos relevantes. Posteriormente, se realizó un primer commit que recogía el estado inicial de la aplicación.

A continuación, se creó un repositorio remoto en GitHub, al cual se vinculó el repositorio local mediante la correspondiente URL. Finalmente, se realizó la subida del código al repositorio remoto, lo que permitió almacenar el proyecto de forma centralizada y facilitar su posterior despliegue en la nube.

6.4 Despliegue de la aplicación con Streamlit Cloud

Una vez disponible el repositorio en GitHub, se procedió al despliegue de la aplicación utilizando **Streamlit Cloud**. Esta plataforma permite publicar aplicaciones desarrolladas con Streamlit directamente a partir de un repositorio remoto, automatizando el proceso de instalación de dependencias y ejecución del código.

El proceso de despliegue consistió en conectar la cuenta de GitHub con Streamlit Cloud, seleccionar el repositorio correspondiente y especificar el archivo principal de la aplicación. Tras completar estos pasos, la plataforma generó automáticamente una URL pública desde la cual se puede acceder a la aplicación.

De este modo, la aplicación pasó de estar disponible únicamente en un entorno local (localhost) a ser accesible desde cualquier dispositivo con conexión a internet.

6.5 Funcionamiento de la aplicación

La aplicación desarrollada permite al usuario interactuar de forma sencilla con el modelo de inteligencia artificial. El usuario puede subir o tomar una imagen de una moneda romana a través de la interfaz web, tras lo cual la imagen es procesada y transformada para adaptarse a los requisitos del modelo.

Una vez realizado este preprocesamiento, la imagen se introduce en la red neuronal basada en ResNet18, que genera una predicción correspondiente a una de las clases de monedas previamente definidas. Además de la clase predicha, la aplicación muestra la probabilidad

asociada a dicha predicción, así como las alternativas más probables, lo que permite al usuario evaluar el grado de confianza del modelo.

Cabe destacar que el modelo ha sido entrenado exclusivamente con un conjunto cerrado de 10 clases de monedas. Como consecuencia, cuando se introduce una imagen correspondiente a una moneda que no pertenece a ninguna de estas clases, el modelo no dispone de una categoría adecuada para clasificarla correctamente. En estos casos, el sistema sigue generando una predicción hacia alguna de las clases conocidas, pero la probabilidad asociada suele situarse en valores moderados, típicamente entre el 40% y el 65%. Este comportamiento es coherente con el funcionamiento del modelo, ya que refleja una menor confianza debido a la falta de correspondencia real con las clases aprendidas.

Por el contrario, cuando la imagen pertenece a una de las clases incluidas en el entrenamiento, el modelo suele asignar probabilidades significativamente más altas, generalmente entre el 80% y el 100%, lo que indica una mayor seguridad en la predicción. Este patrón permite interpretar la probabilidad no sólo como una medida de clasificación, sino también como un indicador indirecto de si la moneda introducida pertenece o no al conjunto de clases conocidas por el modelo.

Todo este proceso se realiza en tiempo real, proporcionando una experiencia interactiva e inmediata.

El flujo interno es el siguiente:

Usuario → subida de imagen → modelo IA → predicción → visualización

Aunque la aplicación desarrollada no es una aplicación nativa, su carácter web permite su utilización en dispositivos móviles de forma sencilla. Para ello, basta con acceder a la URL generada mediante un navegador web.

Además, tanto en iOS como en Android existe la posibilidad de añadir la aplicación a la pantalla de inicio del dispositivo, creando un acceso directo que funciona de manera similar a una aplicación instalada. Esta funcionalidad mejora significativamente la experiencia de usuario, ya que permite acceder a la aplicación de forma rápida y directa.

6.6 Ventajas y limitaciones de la aplicación

La solución implementada presenta diversas ventajas. En primer lugar, permite una integración directa del modelo de inteligencia artificial sin necesidad de realizar adaptaciones a otros lenguajes o plataformas. En segundo lugar, reduce significativamente el tiempo de desarrollo, al evitar la creación de una interfaz desde cero mediante tecnologías frontend tradicionales.

Asimismo, la aplicación es accesible desde cualquier dispositivo, lo que la convierte en una solución altamente flexible y adaptable. Por último, el uso de GitHub y Streamlit Cloud

facilita tanto la gestión del código como su despliegue, permitiendo realizar actualizaciones de forma sencilla.

No obstante, la solución también presenta ciertas limitaciones. En primer lugar, al tratarse de una aplicación web, depende de la disponibilidad de conexión a internet para su funcionamiento. Además, no ofrece todas las capacidades de una aplicación nativa, especialmente en lo relativo al acceso directo al hardware del dispositivo.

6.7 Conclusiones

En conclusión, la combinación de un modelo basado en ResNet18 con la plataforma Streamlit ha permitido desarrollar una aplicación funcional, accesible y fácilmente desplegable. Esta resulta especialmente adecuada en el contexto de proyectos, por ejemplo, de carácter histórico donde es interesante integrar modelos de inteligencia artificial de forma eficiente, manteniendo un equilibrio entre funcionalidad y complejidad de desarrollo.

7. Bibliografía. Artículo de referencia.

- Anwar, H., Anwar, S., Zambanini, S., & Porikli, F. (s. f.). *Deep Ancient Roman Republican Coin Classification via Feature Fusion and Attention*.