

Cats vs Dogs: Crear una CNN para distinguir entre imágenes de perros o gatos.

Julieta Itzel Pichardo Meza | A01369630@tec.mx

1 de diciembre de 2025

ABSTRACT Este documento presenta el desarrollo, evaluación y comparación de tres versiones progresivas de un modelo de clasificación de imágenes para el problema Dogs vs. Cats. A lo largo del estudio se implementaron arquitecturas convolucionales de distinta complejidad, iniciando con un modelo básico diseñado desde cero, seguido de una versión más profunda con mecanismos adicionales de regularización, y concluyendo con una tercera versión basada en transfer learning utilizando ResNet18 preentrenada en ImageNet. Cada versión fue analizada en términos de capacidad de representación, estabilidad durante el entrenamiento, generalización y desempeño en conjuntos de validación y pruebas. Los resultados demuestran una mejora continua entre versiones, destacándose la versión 3 al alcanzar una accuracy cercana al 99% y un 100% de aciertos en imágenes nuevas no utilizadas durante el entrenamiento.

Keywords: Deep Learning, Convolutional Networks (CNN), Image Classification, Dogs vs. Cats Dataset, Pytorch, Computer Vision, Data Augmentation, Regularization, Confusion Matrix, Feature Extraction, Binary Classification, GPU Acceleration (CUDA).

1. INTRODUCCIÓN

La visión computacional con Deep Learning es una rama de la inteligencia artificial que utiliza redes neuronales profundas para que las computadoras interpreten y comprendan imágenes y videos para ejecutar tareas complejas. Las técnicas de Deep Learning permiten a las máquinas aprender patrones directamente de los datos visuales, lo que resulta en aplicaciones como la conducción autónoma, la detección de defectos en producción, el reconocimiento facial y diagnósticos médicos. En este documento, se implementa un modelo de Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) para distinguir entre imágenes de perros y gatos.

El modelo desarrollado fue evaluado a través de tres versiones progresivas, cada

una incorporando mejoras arquitectónicas y metodológicas. La versión final, basada en transfer learning con ResNet18 preentrenada, alcanzó una accuracy cercana al 99% en validación y un desempeño del 100% en un conjunto independiente de imágenes, demostrando una capacidad de generalización superior y consolidándose como la solución más robusta para el problema planteado.

2. DESCRIPCIÓN DEL DATASET

El dataset “Dogs vs. Cats” es el conjunto de datos estándar utilizado para la evaluación de modelos de visión por computadora distribuido a través de una competición de Kaggle.

Este dataset consiste en 25,000 imágenes a color (en una escala RGB) que ya están etiquetadas en dos clases: 12,500

imágenes de perros y 12,500 imágenes de gatos.

Sus características son las siguientes:

- Cada imagen tiene como nombre la etiqueta del animal al que pertenece y su número de imagen.
- Todas las imágenes son tipo jpg.
- Todas las imágenes tienen diferentes dimensiones.

Dentro del dataset, también hay un archivo .csv con nombre “sampleSubmission”, que demuestra el formato en el que se deben subir los resultados del modelo si es que se decide participar en la competencia.

3. PROCESAMIENTO DEL DATASET

El procesamiento de las imágenes consistió en tomar un número fijo de imágenes y dividirlas en los conjuntos de train, validation. Para el conjunto de test, se usan 100 imágenes aleatorias (50 perros y 50 gatos) que no se usaron en los conjuntos anteriores para evaluar el performance de cada modelo.

En la primera versión del modelo, se utilizaron 5,000 imágenes de perros y 5,000 imágenes de gatos. Estas se dividieron en los siguientes conjuntos:

- 4,000 perros/gatos para entrenar (80% train).
- 1,000 perros/gatos para validar el entrenamiento (20% validation).

En esta versión del modelo, sí se hacen transformaciones al conjunto de

imágenes de entrenamiento. Este es un procesamiento básico: se redimensionan las imágenes a un tamaño fijo, se hace la conversión a tensor, se normalizan las intensidades de los píxeles y un aumento de datos básicos que consistió en rotaciones aleatorias suaves y el volteo horizontal aleatorio.

En cambio, en el conjunto de validación, no se deforman las imágenes de manera aleatoria, solo se redimensionan, se convierten a tensor y se normalizan a píxeles.

En la segunda versión del modelo, se buscó ampliar el número de imágenes que se usaron para entrenar, de igual manera siguiendo la estructura del modelo versión 1: 8,000 imágenes de perros y 8,000 imágenes de gatos, divididas en los siguientes conjuntos:

- 7,089 perros/gatos para entrenar (80% train).
- 2,285 perros/gatos para validar el entrenamiento (20% validation).

En esta otra versión del modelo se mantiene el procesamiento de la versión anterior, pero usando diversas técnicas nuevas para el aumento de datos, que consisten en las siguientes:

- Recortes, para que el modelo pueda identificar al animal sin necesariamente estar en el centro de la imagen.
- Aumentos geométricos como rotaciones aleatorias y volteos horizontales aleatorios.
- Aumentos fotométricos que consisten en cambiar aleatoriamente el brillo, contraste

o saturación para que el modelo sea menos sensible a condiciones de iluminación.

Finalmente se hace la conversión a tensor, y la normalización con medias y desviaciones estándar tipo ImageNet. Este estándar permite que las imágenes tengan una distribución compatible con arquitecturas preentrenadas, además de que sí ayudan al modelo a mejorar la consistencia del entrenamiento y reduce la sensibilidad a las variaciones en iluminación y color cuando se trata de procesar imágenes.

En cuanto al conjunto de validación, se mantiene el mismo pipeline menos los aumentos aleatorios: se redimensionan las imágenes, se convierten a tensor y se normalizan los píxeles con los mismos parámetros que los que usa el conjunto de entrenamiento.

En la tercera versión del modelo se utilizó en su totalidad el dataset. Al igual que en los modelos anteriores, se tomó en cuenta los mismos conjuntos de entrenamiento, validación y pruebas. Estos se dividieron de la siguiente manera:

- 8,750 perros/gatos para entrenar (60% train).
- 1,875 perros/gatos para validación (20% validation).
- 1,875 perros/gatos para pruebas (20% test), de las cuales se eligen solo 100 imágenes aleatoriamente para la prueba de matriz de confusión.

En esta versión del modelo sí se aplican transformaciones al conjunto de imágenes de entrenamiento, pero ahora

con un data augmentation más completo y robusto. Aquí las imágenes se someten a un recorte aleatorio redimensionado para variar la escala del objeto que se encuentra en la imagen, en este caso, los animales, después se aplican volteos horizontales y pequeñas rotaciones para generar aumentos geométricos; de la misma manera, se añaden variaciones de brillo, contraste y saturación para simular diferentes condiciones de iluminación. Terminando estas transformaciones, las imágenes se convierten a tensor y se normalizan utilizando los valores estándar de ImageNet.

En cambio, en el conjunto de validación y prueba no se introduce ninguna variación aleatoria, las imágenes únicamente se redimensionan a un tamaño fijo, se convierten a tensor y se normalizan, de esta manera garantizando una evaluación consistente y sin distorsiones artificiales.

4. METODOLOGÍA Y ARQUITECTURA

Las Redes Neuronales Convolucionales (CNNs) son un tipo de arquitectura especializada en el procesamiento de imágenes por su capacidad de extraer patrones visuales mediante operaciones de convolución.

A diferencia de una red neuronal tradicional, las CNNs aprovechan la estructura espacial de la imagen (sus píxeles organizados en altura, anchura y canales) para aprender características como bordes, texturas, formas y finalmente, objetos completos. Esta naturaleza es lo que las hace adecuadas

para la clasificación de imágenes, como lo es la problemática que se busca resolver: identificar correctamente a perros y gatos.

De esta manera, las CNNs pueden identificar diferencias entre ambas clases, más específicamente como proporciones corporales, formas de las orejas, hocico, textura del pelaje y contornos del rostro.

Tomando en cuenta la información anterior, los modelos hechos buscan encontrar la configuración que mejor generalice al clasificar nuevas imágenes de perros y gatos que no se encuentren en el dataset de entrenamiento. Cada modelo hecho incorpora modificaciones en base al anterior para mejorar su capacidad de aprendizaje y performance.

4.1 Arquitectura del modelo VI

La primera versión del modelo se construyó como una CNN diseñada desde cero, siguiendo una estructura típica para tareas de clasificación de imágenes donde no se emplea transfer learning. Esta arquitectura está compuesta por tres bloques convolucionales seguidos por dos capas totalmente conectadas que realizan la clasificación final.

```
CNNPerrosGatos(
    (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fc1): Linear(in_features=41472, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=1, bias=True)
)
```

Imagen 1. Arquitectura de la primera versión del modelo perros vs. gatos.

A continuación, se describe el propósito de cada capa:

- **Capa Convolucional 1 – conv2d (3 → 32):** su entrada es una imagen en escala RGB, su operación consiste en aplicar 32 filtros de tamaño 3x3 con padding=1 para preservar la resolución espacial. Su función es detectar patrones de bajo nivel como bordes, líneas y texturas simples, esto le da al modelo la capacidad de reconocer variaciones iniciales en color y formas básicas.
- **Capa de MaxPooling – MaxPool2d(kernel=2, stride=2):** en esta capa se reducen las dimensiones de la imagen a la mitad, su función es eliminar el ruido y mantener las características más prominentes.
- **Capa Convolucional 2 – Conv2d(32 → 64):** esta capa detecta patrones más complejos combinando las características ya extraídas anteriormente. Su función es aprender texturas, contornos más definidos y partes específicas, en este caso serían las orejas, ojos, forma del hocico tanto en perros como en gatos.
- **Capa Convolucional 3 – Conv2d(64 → 128):** en esta capa se incrementa la profundidad del mapa de características a 128 filtros. Su función es extraer representaciones de alto nivel y reconocer estructuras más complejas de los objetos como la forma completa de los animales, sus patrones de pelaje, sus proporciones, etc.
- **Capa Totalmente Conectada 1 — Linear(41472 → 512):**

después de aplanar todos los mapas de características, esta capa integra la información aprendida en las convoluciones, reduce la dimensionalidad hacia una representación compacta de 512 neuronas. Esto actúa como “resumen” de todo lo que el modelo aprendió sobre la imagen.

- **Capa Totalmente Conectada 2**
 - **Linear(512 → 1):** en esta capa se produce la salida final del modelo. Como es un valor binario, la tarea se resuelve como clasificación binaria. Este valor luego pasa por una función de activación sigmoide para obtener una probabilidad entre 0 y 1, los valores binarios que representan las etiquetas de gato y perro.

La arquitectura de esta primera versión se diseñó para ser simple, interpretable y al mismo tiempo eficiente.

En este modelo, el uso de Adam como optimizador permite que este modelo pequeño tenga un desempeño competitivo sin requerir experimentación extensa con hiperparámetros.

Uno de los puntos clave fue poner tres capas convolucionales que permiten extraer características desde patrones simples hasta representaciones más abstractas sin generar un modelo demasiado grande.

Otro punto clave es el único bloque de MaxPooling que reduce la dimensionalidad y previene el overfitting sin perder demasiada información espacial.

Por último, las capas fully connected convierten las características encontradas en una decisión final de clasificación.

Esta arquitectura proporciona una base sólida y ligera para pruebas iniciales; además de que sirve como punto de comparación para modelos posteriores con arquitecturas diferentes.

4.2 Arquitectura del modelo V2

Tomando en cuenta los resultados de la arquitectura de la versión 1 del modelo, se le hicieron modificaciones para mejorar su performance (ver punto 5).

Estas modificaciones fueron hechas a la CNN original, añadiendo mejoras para controlar el overfitting y la extracción de las características de los animales de manera más robusta. Esta red está compuesta por tres bloques convolucionales, cada uno seguido por max pooling y dos capas conectadas con dropout intermedio.

```
Usando dispositivo: cuda
DogsCatsCNN
(conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(dropout_conv): Dropout(p=0.25, inplace=False)
(fc1): Linear(in_features=41472, out_features=512, bias=True)
(dropout_fc): Dropout(p=0.5, inplace=False)
(fc2): Linear(in_features=512, out_features=1, bias=True)
)
```

Imagen 2. Arquitectura de la segunda versión del modelo perros vs. gatos.

A continuación, se describe el propósito de cada capa:

- **Capa Convolucional 1 — Conv2d(3 → 32):** recibe imágenes RGB con sus 3 respectivos canales. Su operación es aplicar 32 filtros de tamaño 3x3 con padding=1 para preservar la resolución espacial.

Su función es detectar patrones de bajo nivel como bordes, líneas y texturas simples, esto le da al modelo la capacidad de reconocer variaciones iniciales en color y formas básicas.

- **Capa de MaxPooling 1 — MaxPool2d(kernel=2, stride=2):** aquí se reduce el tamaño espacial a la mitad, y tiene como función resumir la información más relevante en cada región.
- **Capa Convolucional 2 — Conv2d(32 → 64):** aquí se incrementa el número de filtros a 64. La función de esta capa es aprender los patrones más complejos que combinan las características de la primera capa y detectar partes más definidas de los animales.
- **Capa de MaxPooling 2 — MaxPool2d(kernel=2, stride=2):** en esta capa se vuelve a reducir a la mitad las dimensiones espaciales, su función es comprimir aún más la información relevante y preparar mapas de características más compactos.
- **Capa Convolucional 3 — Conv2d(64 → 128):** aquí se aumenta la profundidad a 128 filtros, esto con la función de extraer representaciones de alto nivel y reconocer configuraciones más globales como las formas del cuerpo, las proporciones de cada animal y las combinaciones de texturas.
- **Capa de MaxPooling 3 — MaxPool2d(kernel=2, stride=2):** este es el tercer nivel de reducción espacial con la función de obtener mapas de características pequeños pero más informativos, así como ayudar a las capas finales a trabajar con una representación más abstracta.
- **Dropout en convoluciones — Dropout(p=0.25):** en esta capa se aplica después de los bloques convolucionales y su función es “apagar” aleatoriamente un porcentaje de neuronas (en este caso el 25%) durante el entrenamiento, esto es para prevenir el overfitting, obligando al modelo a no aprender de un subconjunto específico de filtros.
- **Capa Totalmente Conectada 1 — Linear(41472 → 512):** aquí se aplana los mapas de características y los proyecta a un vector de 512 neuronas, esto es para integrar toda la información de alto nivel extraída por las convoluciones y actúa como una representación compacta de alto nivel de la imagen.
- **Dropout en fully connected — Dropout(p=0.5):** aquí se apaga aleatoriamente el 50% de las neuronas de la capa fully connected 1 durante el entrenamiento, su función es evitar que el modelo memorice patrones específicos para forzar a la red a aprender características más robustas.
- **Capa Totalmente Conectada 2 — Linear(512 → 1):** esta es la salida final como un único valor, se interpreta como una probabilidad de pertenecer a una

de las dos clases: perros o gatos tras pasar por una función de activación sigmoide.

En este modelo es más profundo, más regularizado y más propenso a tener gradientes más variables por su arquitectura, usar Adam como optimizador ayuda a mantener la estabilidad del entrenamiento incluso cuando los gradientes se vuelven más dispersos al usar dropout; además de acelerar la convergencia compensando la mayor complejidad del modelo con respecto a la versión 1.

4.3 Arquitectura de la tercera versión del modelo

La tercera versión del modelo utiliza ResNet18 con Transfer Learning. Esta es una arquitectura de red neuronal profunda que ha sido ganadora del premio a mejor desempeño en múltiples benchmarks de visión por computadora. Una de sus características más importantes es el uso de bloques residuales (o residual blocks en inglés) que facilitan el entrenamiento de redes profundas evitando la desaparición del gradiente (vanishing gradient en inglés).

A continuación, se presenta la arquitectura de ResNet18:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

Imagen 3. Arquitectura de la tercera versión del modelo perros vs. gatos usando ResNet18 usando Transfer Learning.

En breve, se presenta en detalle la arquitectura de la versión 3 del modelo:

- **Capa Convolucional Inicial — Conv2d(3 → 64, kernel=7×7, stride=2):** recibe imágenes RGB y extrae patrones de bajo nivel con un filtro grande de 7x7. Su función es capturar bordes y texturas generales de manera más amplia que un kernel estándar de 3x3.
- **BatchNorm + ReLU:** aquí se normaliza las activaciones para estabilizar el entrenamiento y aplica una función de activación no lineal que permite aprender representaciones más complejas de los animales.
- **MaxPooling — kernel=3, stride=2:** aquí se reduce la resolución espacial de los mapas de características y concentra la información relevante para las capas profundas.

Bloques residuales (BasicBlock) — Capas layer1 a layer4: cada layer contiene dos bloques residuales. Cada uno de estos bloques tiene dos convoluciones 3x3 con BatchNorm y ReLU; además de incorporar una conexión residual (skip connection) que suma la entrada directamente con la salida del bloque.

El propósito de estos bloques es permitir que la red sea más profunda sin perder información y evitar problemas de

gradientes, lo que termina facilitando el aprendizaje de características más ricas para el modelo.

A continuación, se describe cada layer y la función de los bloques residuales:

- **Layer1 — (64 → 64):** aquí son dos bloques residuales que mantienen el mismo número de canales (64), su función es refinar y estabilizar las características básicas del primer nivel, como bordes y texturas simples.
- **Layer2 — (64 → 128, primer bloque con stride=2):** el primer bloque incrementa los canales de 64 a 128 y reduce resolución espacial, el downsample ajusta la dimensión del tensor para poder realizar la suma residual. Su función es aprender representaciones más complejas como patrones de pelaje, contornos y formas distintivas.
- **Layer3 — (128 → 256, primer bloque con stride=2):** aquí se incrementan los canales a 256 y reduce nuevamente la resolución; su función es extraer las características de nivel medio-alto como partes del cuerpo, orientación y composición de la imagen.
- **Layer4 — (256 → 512, primer bloque con stride=2):** este es el último incremento de profundidad hacia 512 canales, y su función es capturar la información de alto nivel con la que la red aprende patrones que ya diferencian entre perros y gatos, como las formas, las proporciones y estructuras completas.
- **Adaptive Average Pooling — output_size=1×1:** aquí se resume cada uno de los 512 mapas de características a un único valor promedio que va a generar una representación compacta y robusta, esto reduce el riesgo de sobreajuste y reemplaza a las capas densas excesivas de modelos tradicionales.
- **Capa Totalmente Conectada Final — Linear(512 → 2):** aquí se generan dos valores de salida (perro o gato) para clasificación binaria. En esta capa se entrena desde cero en este modelo, adaptándose directamente al dataset Dogs vs Cats.

La elección de ResNet18 como base del modelo de la versión 3 se justifica por su capacidad comprobada para aprender representaciones visuales altamente complejas gracias a el uso de los bloques residuales, los cuales facilitan el flujo del gradiente y permiten entrenar redes profundas de manera estable.

A diferencia de una CNN entrenada desde cero, ResNet18 aporta filtros preentrenados en millones de imágenes (Transfer Learning), lo que permite reutilizar características para detectar bordes, textura, formas y estructuras complejas de los animales desde las primeras capas, lo que acelera la velocidad de convergencia, reduce el sobreajuste y aumenta la capacidad de generalización.

Así mismo, el uso de Adaptive Average Pooling y la progresiva expansión de canales hasta 512 permiten obtener una representación compacta pero rica en información, ideal para un problema de clasificación.

De la misma manera que en los dos modelos anteriores, se hizo uso del optimizador Adam debido a su alta capacidad de adaptarse dinámicamente a los gradientes durante el entrenamiento, lo que resulta especialmente adecuado para arquitecturas profundas como ResNet18.

En resumen, esta arquitectura ofrece mayor estabilidad, mejor rendimiento y una extracción de características más profunda y eficiente que las versiones previas del modelo.

4.4 Hiperparámetros en común y únicos entre versiones

Para los primeros dos modelos, hubo hiperparámetros que permanecieron igual, uno de ellos fue el número de épocas de entrenamiento, que fueron 30 para ambas versiones. Para la tercera versión solo se definieron 5 épocas gracias al performance proporcionado usando Transfer Learning además del uso de early stopping, esto se aplicó como mecanismo para detener el entrenamiento cuando la pérdida de validación dejara de mejorar. Esta técnica fue especialmente adecuada al usar Transfer Learning con ResNet18 preentrenada ya que el modelo aprende patrones útiles desde las primeras épocas y logra converger rápidamente, lo que haría inconveniente seguir con más etapas de entrenamiento, aumentando el

riesgo de que el modelo empiece a ajustarse de forma demasiado específica al conjunto de entrenamiento, lo que causaría overfitting. El early stopping ayuda a identificar automáticamente cuando el modelo ha alcanzado su mejor performance en la etapa de validación, evitando el entrenamiento innecesario.

Uno de los hiperparámetros que permaneció igual en todas las versiones fue el learning rate, que se definió en 1e-4, o 0.0001 en forma decimal.

Tanto el batch size como el tamaño estático de las imágenes permaneció igual para los tres modelos, siendo el primero de 32 imágenes, y el segundo valor de 150 x 150 píxeles.

Para los primeros dos modelos, se empleó la función de pérdida BCEWithLogitsLoss, que es una función recomendada para problemas de clasificación binaria como la del dataset perros vs. gatos. Esta función combina dos componentes esenciales: Binary Cross-Entropy (BCE) y una función sigmoide interna aplicada a la salida del modelo, lo que le permite al modelo no incluir explícitamente una sigmoide en la última capa, ya que BCEWithLogitsLoss ya la aplica.

Esta función calcula la diferencia entre la probabilidad predicha por el modelo (después de aplicar la sigmoide) y la etiqueta real, ya sea 0 = gato o 1 = perro. Mientras más lejos esté la predicción del valor real, más alta será la pérdida, causando que el modelo ajuste los pesos necesarios durante el entrenamiento.

La manera en la que se manejan los pesos es la siguiente:

- Si el modelo asigna alta probabilidad a la clase correcta, la pérdida es baja.
- Si asigna una alta probabilidad a la clase incorrecta, la pérdida será alta.

De la misma manera, esta función de pérdida evita desbordamientos numéricos, aumenta la estabilidad de los gradientes y mejora la convergencia del entrenamiento, lo que la hace buena para aplicar en CNNs cuando las salidas pueden variar ampliamente al inicio del entrenamiento.

En cambio, para la tercera versión del modelo se utilizó la función de pérdida CrossEntropyLoss, que se utiliza de forma recurrente para la clasificación binaria con dos logits de salida.

Esta función combina dos pasos fundamentales dentro de una sola operación:

- **Softmax:** convierte los logits de salida del modelo en probabilidades.
- **Negative Log-Likelihood (NLL):** calcula qué tan mala fue la probabilidad asignada a la clase correcta.

Estas dos operaciones permiten entrenar redes profundas de manera estable y eficiente.

CrossEntropyLoss mide qué tan bien coincide la probabilidad asignada por el modelo a la clase correcta. Si la probabilidad es alta, la pérdida es baja; si es baja, la pérdida crece rápidamente. Esta penalización guía a la red a ajustar sus pesos para aumentar la probabilidad de la clase correcta en cada ejemplo; además de facilitar el entrenamiento

estable del modelo que está basado en ResNet18.

Aunque el problema es de clasificación binaria (perros vs. gatos), la versión 3 del modelo utiliza CrossEntropyLoss porque la arquitectura basada en ResNet18 produce una salida con dos logits en lugar de un solo valor. En este caso de configuración CrossEntropyLoss es la función adecuada, ya que está diseñada para manejar directamente múltiples logits y aplicar internamente Softmax y la pérdida logarítmica que corresponde a la clase correcta. Esto permite que el modelo aprenda de manera más estable y consistente dentro del flujo de entrenamiento de redes de clasificación multiclas.

Por otro lado, BCEWithLogitsLoss se usa principalmente cuando la red produce un único logit y se requiere modelar la probabilidad de “clase positiva” en problemas binarios o con diversas etiquetas. En este caso, la ResNet18 modificada no genera una salida escalar, sino un vector de dimensión 2, por lo que BCEWithLogitsLoss no coincide de manera natural con la forma de la salida y requeriría una reconfiguración innecesaria del modelo.

Además, CrossEntropyLoss tiene ventaja al permitir a la red competir entre ambas clases a través de Softmax, lo cual es más adecuado cuando las clases son mutuamente excluyentes (decidir entre perro o gato) y hay una clara relación de competencia entre ellas.

En resumen, CrossEntropyLoss se adapta mejor a la forma de salida de

ResNet, ya que evita transformaciones adicionales a la arquitectura y proporciona un entrenamiento más estable usando múltiples logits.

5. RESULTADOS POR MODELO

En esta sección del documento se analizan los resultados del desempeño de cada modelo de acuerdo con tres diferentes indicadores: curva de Accuracy, curva de Loss y una Matriz de Confusión.

A continuación, se habla de los resultados de cada modelo de manera individual.

5.1 Resultados del Modelo V1

Curva de Accuracy

En la gráfica de la izquierda, se logra observar que el accuracy de entrenamiento sube de manera continua casi al 100%, además de que el accuracy de validación alcanza un tope alrededor de 78% a 80% y después logra estabilizarse en ese rango.

Esto indica que el modelo aprende bien sobre los datos de entrenamiento pero su capacidad de generalización hacia datos nuevos sí es limitada.

Curva de Loss

La gráfica muestra que Training loss desciende de forma constante durante las 30 épocas del entrenamiento, pero Validation loss baja solo al inicio, a partir de la época 10 u 11 empieza a oscilar, y después de la época 20 sube de manera progresiva.

Esto indica que el modelo tiene overfitting, es decir, que la red de este modelo está memorizando los patrones del conjunto de entrenamiento, pero no aprende representaciones suficientemente robustas para datos nuevos.

A continuación, se muestran ambas gráficas de las dos curvas antes mencionadas:

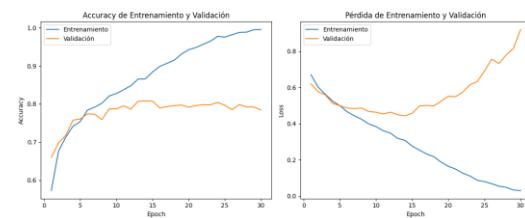


Imagen 4. Curva de Accuracy (izquierda) y Curva de Loss (derecha) del modelo V1.

De la misma manera, se hace la interpretación para la matriz de confusión de la primera versión del modelo.

Matriz de Confusión

La matriz de confusión toma 100 imágenes no usadas por el modelo a lo largo de su entrenamiento para probar su desempeño de predicción.

La exactitud general del modelo sobre estas 100 imágenes consiste en que clasificó 39 gatos y 44 perros de manera correcta, sumando 83 aciertos y 17 errores.

Dando un Accuracy aproximado de 83% para el modelo en general.

Analizando el comportamiento de acuerdo con cada clase, se puede dar el siguiente análisis.

Para la clase de gatos, se hicieron 39 predicciones correctas con 11 errores, lo que equivale a un accuracy de 78% para la clase de gatos.

En cambio, en la clase de perros se hicieron 44 predicciones correctas con 6 errores, lo que equivale a un accuracy de 88% para la clase de perros.

Este comportamiento asimétrico muestra que comete más errores al clasificar gatos como perros. Esto afirma la detección del overfitting en el modelo, ya que las características distintivas de los gatos no están siendo capturadas con suficiente robustez por esta arquitectura actual.

En conjunto, los resultados muestran que el modelo V1 logra aprender los patrones básicos del dataset, pero su capacidad de generalización es limitada debido a sobreajuste.

A continuación, se muestra la matriz de confusión de la primera versión del modelo:

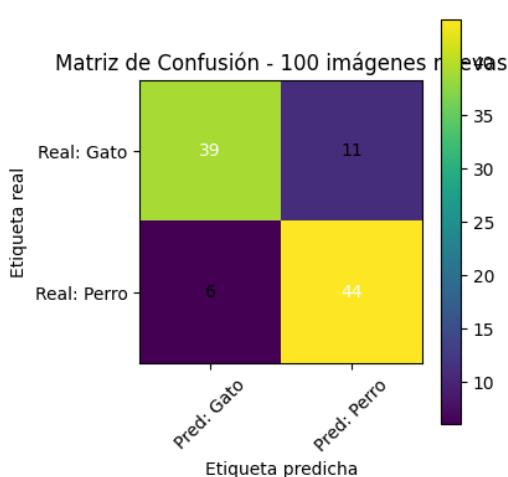


Imagen 5. Matriz de Confusión del modelo V1.

5.2 Resultados del Modelo V2

Curva de Accuracy

En la gráfica de la izquierda, se muestra un comportamiento saludable en la mayor parte del entrenamiento, ya que el accuracy del entrenamiento sube de manera constante hasta llegar cerca del 95%. De la misma manera, el accuracy de validation también crece de forma estable y se mantiene en el rango de 90% al final de las 30 épocas; además, las dos curvas de mantienen relativamente cercanas, sin que se abran brechas amplias entre los dos conjuntos.

El modelo sí está aprendiendo patrones reales del dataset y hasta ahora, no muestra un problema visible de overfitting.

Curva de Loss

En la gráfica de la derecha se muestra que el loss del entrenamiento desciende de forma continua a lo largo de las épocas; además de que el loss de validation baja de buena manera alrededor de la época 10, y después oscila para luego subir de manera preocupante.

El loss en validation no diverge, lo que confirma que el modelo generaliza razonablemente bien, con una ligera fluctuación típica de conjuntos de imágenes con alta variabilidad (como en este caso, perros y gatos).

A continuación, se muestran las dos gráficas de las dos curvas antes mencionadas:

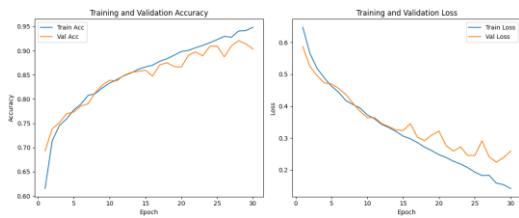


Imagen 6. Curva de Accuracy (izquierda) y Curva de Loss (derecha) del modelo V2.

Matriz de Confusión

En el caso de la versión 2 del modelo, su matriz de confusión muestra que sobre 100 imágenes no incluidas en el entrenamiento, clasificó de manera correcta a 46 gatos y 37 perros, sumando así 83 aciertos y 17 errores nuevamente.

Esto da un 83% de accuracy para este modelo en general.

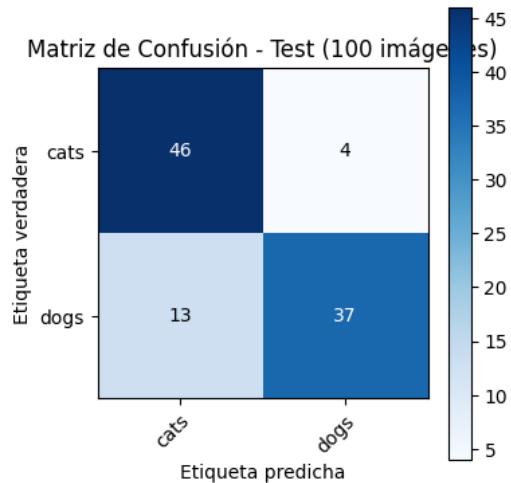
Analizando el comportamiento por clase, se puede dar el siguiente análisis:

Para la clase de gatos, esta cuenta con 46 aciertos y 4 errores, demostrando así un accuracy del 92% para esta clase, dando a entender que el modelo reconoce muy bien las características de esta clase.

Por otro lado, la clase de perros muestra en total 37 aciertos y 13 errores, demostrando que esta clase tiene un accuracy del 74%.

Estos resultados indican que el modelo tiene una mayor dificultad para reconocer perros, especialmente aquellos que visualmente se parecen a gatos, con colores claros, tamaños pequeños o fondos similares.

A continuación, se muestra la matriz de confusión de la segunda versión del modelo:



5.3 Resultados del Modelo V3

Curva de Accuracy

En la gráfica de la derecha, se observa que el accuracy presenta un comportamiento muy bueno: el Train Accuracy sube de 0.97 a casi 0.99 a lo largo de las 5 épocas de entrenamiento; además de que el Validation Accuracy inicia en un valor alto (aproximadamente en 0.98), sube a un valor aproximado de 0.99 y se vuelve a estabilizar en 0.98.

Esto indica que el accuracy de validation es prácticamente el mismo que el de entrenamiento, lo que significa que el modelo ha generalizado bien, no muestra que haya memorizado el dataset y está clasificando correctamente las imágenes, aunque no las haya visto antes.

Cabe mencionar que la combinación de ambas de estas curvas es difícil de lograr con modelos entrenados desde cero, y es una de las ventajas del uso de ResNet18 que ya fue previamente entrenada.

Curva de Loss

En la gráfica izquierda se observa un comportamiento estable y consistente con la buena generalización que hace este modelo: el Train Loss desciende de forma continua desde aproximadamente 0.075 hasta un aproximado de 0.026 de manera rápida, lo que indica que el modelo aprende progresivamente patrones útiles sin estancarse.

En cuanto a la curva de Validation Loss se mantiene baja durante todo el entrenamiento, balanceándose entre unos valores aproximados de 0.040 y 0.051. Esto muestra que no hay un incremento abrupto, lo que indica que no hay sobreajuste.

Estos patrones indican que hay un aprendizaje estable, sin síntomas de overfitting, gracias al uso de Transfer Learning, la regulación implícita de ResNet y el data augmentation que usa esta versión del modelo.

A continuación, se muestran las dos gráficas de las dos curvas antes mencionadas:

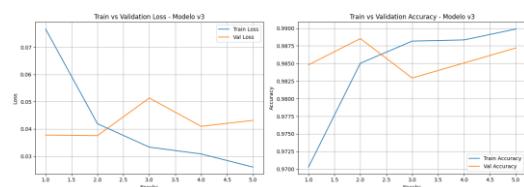


Imagen 8. Curva de Accuracy (derecha) y Curva de Loss (izquierda) del modelo V3.

Matriz de Confusión

Para la versión 3 del modelo, se puede observar que sobre las 100 imágenes que no se usaron en etapas anteriores del modelo, este logró acertar 43 perros, sin errores y 57 gatos, sin errores.

Esto le da un accuracy al modelo de 100% de manera general.

Tanto la clase de perros como de gatos cuentan con un 100% de accuracy, por lo que el modelo ha generalizado características de ambas clases de manera óptima.

Estos hechos confirman que no hay evidencia de sobreajuste a los datos de entrenamiento y que esta versión del modelo maneja correctamente variaciones reales del dataset, como la orientación de las fotografías, las diferencias entre fondos, las variaciones en la iluminación, tamaños diferentes entre los animales, entre otras características.

A continuación, se muestra la matriz de confusión de la tercera versión del modelo:

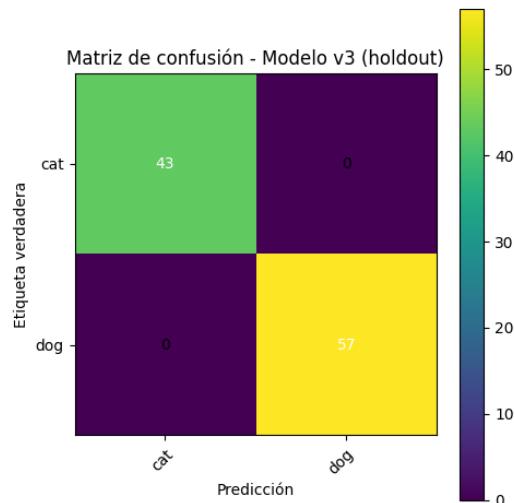


Imagen 9. Matriz de Confusión del modelo V3.

6. DISCUSIÓN DE RESULTADOS

6.1 Estabilidad del entrenamiento

La estabilidad de las curvas de pérdida y precisión también evidencia progresión entre las tres versiones. En la versión 1,

el train loss disminuía rápidamente, pero la validación no era por completo estable, lo que indica overfitting. La versión 2 logró una reducción más suave del train loss y un acercamiento más gradual de la curva de validación, aunque aún había diferencias notables entre ambas, especialmente en las primeras épocas. La versión 3 mostró el comportamiento más estable: tanto train loss como validation loss descendieron casi de manera paralela en las 5 épocas de entrenamiento, con diferencias mínimas entre estas dos curvas al final: train loss con 0.026 y validation loss con 0.038, demostrando que sí hay un equilibrio óptimo entre aprendizaje y generalización.

6.2 Sensibilidad al dataset y robustez ante variaciones

Al evaluar cómo cada modelo reaccionaba a variaciones en el fondo, iluminación, orientación y escala, se observaron diferencias claras. Las versiones 1 y 2 cometían errores comunes en imágenes con ángulos no convencionales o fondos con distracciones; en particular la versión 1 tenía dificultades significativas con imágenes donde el animal era pequeño o poco contrastante. La versión 3, en cambio, mostró una robustez notable debido a los filtros preentrenados en ImageNet, ya que capturan texturas, estructuras y contextos variados. Esto se corroboró al evaluar un conjunto externo de 100 imágenes no utilizadas anteriormente: mientras la v2 aún cometía confusiones aisladas, la versión 3 alcanzó 100% de aciertos, demostrando una generalización

excelente en comparación con versiones pasadas.

6.3 Análisis de Overfitting

Las primeras dos versiones mostraron distintos niveles de overfitting, en la versión 1 el overfitting pasa pocas después de iniciar debido a su capacidad para representar patrones complejos y la falta de mecanismos de regularización. La versión 2, con dropout y mayor profundidad, logró reducir el overfitting, pero seguía mostrando una diferencia significativa entre train accuracy y val accuracy que estaba entre 4 y 7 puntos porcentuales en algunas partes del entrenamiento. La versión 3, en cambio, pudo eliminar este comportamiento: la diferencia entre train accuracy (aproximadamente 0.99) y validation accuracy (aproximadamente 0.987) fue mínima, indicando que el modelo sí aprendió patrones generalizables sin memorizar ejemplos específicos del dataset.

6.4 Procesamiento y dependencia del aumento de datos

La dependencia del procesamiento también varó entre versiones. Las primeras arquitecturas dependían fuertemente de aumento de datos y transformaciones para suplir la falta de profundidad del modelo. La versión 2 se benefició del uso de rotaciones, flips y normalización, lo que ayudó a estabilizar el aprendizaje, aunque necesitaba un conjunto más extenso para alcanzar un rendimiento óptimo. La versión 3 requirió menos dependencia del preprocesamiento externo debido a que

sus filtros preentrenados ya poseen representaciones útiles de colores, texturas, bordes y estructuras, lo que permitió que incluso son aumentos ligeros, el modelo alcanza métricas de validación sobresalientes.

6.5 Esfuerzo de optimización e ingeniería del modelo

El esfuerzo requerido para ajustar hiperparámetros disminuyó progresivamente gracias al uso de arquitecturas mucho más robustas. Las versiones 1 y 2 necesitaron ajustes manuales de número de filtros, tasas de aprendizaje, funciones de activación y dropout para lograr un rendimiento aceptable. La versión 3, por el contrario, funcionó adecuadamente con configuraciones estándar: una tasa de aprendizaje baja ($1e-4$), un optimizador Adam y un entrenamiento de apenas cinco épocas fueron suficientes para alcanzar buenos resultados. Esto demuestra una ventaja clave del transfer learning, reduce la necesidad de optimización extensiva y acelera el desarrollo de un modelo confiable.

6.6 Discusión comparativa entre las versiones del modelo

Las tres versiones del modelo progresan en complejidad, capacidad de representación y rendimiento. Cada versión logra incorporar mejoras de manera gradual que abordan directamente las limitaciones que se observan en la versión anterior, esto permite visualizar cómo decisiones arqueológicas y metodológicas influyen en el desempeño influyen en el

desempeño final del sistema de clasificación.

En la versión 1 se implementó una arquitectura convolucional sencilla diseñada de manera manual. Aunque logró aprender patrones visuales útiles, la capacidad de generalización fue limitada debido a la poca profundidad del modelo y a la ausencia de técnicas de regularización más avanzadas. Esta versión del modelo tenía overfitting, sobre todo cuando se exponía a nuevas imágenes fuera del conjunto de entrenamiento. Las métricas obtenidas mostraron que la arquitectura sí era funcional, pero insuficiente para capturar variabilidad compleja del dataset Dogs vs. Cats.

La versión 2 profundizó la arquitectura e incorporó técnicas como dropout, se añadieron más capas convolucionales y un pipeline de procesamiento más completo. Estos cambios lograron reducir el overfitting y mejorar la estabilidad del entrenamiento. El modelo consiguió tener métricas más altas en accuracy y una matriz de confusión más equilibrada que el de la versión 1. Sin embargo, al seguir entrenando desde cero, la red aún tenía dificultades para generalizar patrones y por lo mismo requería más épocas y muchos más datos para converger. Aunque esta versión es claramente superior a la versión 1, seguía limitada frente a los beneficios de usar arquitecturas modernas preentrenadas.

La tercera versión representó el salto más significativo entre las tres. Al integrar

ResNet18 preentrenada, el modelo aprovechó filtros ya optimizados para detectar texturas, bordes, formas y estructuras complejas aprendidas a partir de más de un millón de imágenes en ImageNet. Esto permite que la red comience desde una representación visual altamente informativa, en lugar de aprender desde cero. Además, los bloques residuales evitan el desvanecimiento del gradiente, permitiendo entrenar modelos profundos de forma estable. Como resultado, el modelo no solo alcanzó una pérdida baja y un accuracy cercana a 0.99 en validación, si no que logró un performance perfecto de 100% al usar imágenes nuevas no usadas anteriormente en el entrenamiento, mostrando una generalización significativamente superior a las versiones anteriores.

En conjunto, la evolución de las tres versiones muestra cómo el uso de arquitecturas más profundas, mecanismo de regularización y finalmente el uso de técnicas de Transfer Learning permiten mejorar progresivamente la estabilidad, precisión y robustez del modelo. La versión 3 no solo supera a las anteriores en su performance; además presenta una arquitectura con mejoras prácticas actuales en visión por computadora, así siendo la versión más efectiva para el problema de clasificación entre perros y gatos.

evidencian un progreso consistente hacia arquitecturas más robustas, estables y capaces de generalizar eficazmente frente a variaciones reales del dataset. La transición desde modelos convolucionales diseñados desde cero hacia una arquitectura de transfer learning basada en ResNet18 permitió aprovechar representaciones visuales previamente aprendidas, acelerando la convergencia y reduciendo de manera significativa el overfitting. Este cambio metodológico no solo elevó las métricas de desempeño, así alcanzando un accuracy cercana al 0.99 en validación y un 100% en el conjunto final de pruebas, además de que también disminuyó la complejidad del proceso de optimización, así como también disminuyó la complejidad del sistema para adaptarse a condiciones diversas. En consecuencia, la versión 3 se consolida como la aproximación más efectiva y madura dentro del proyecto, demostrando que el uso de arquitecturas modernas preentrenadas ofrece ventajas sustanciales en tareas de clasificación de imágenes con alta variabilidad como el del dataset de Dogs vs. Cats.

7. CONCLUSIÓN

En conjunto, los resultados obtenidos a lo largo de las tres versiones del modelo

