



AEC3 - Competición Kaggle

Introducción

Esta actividad ha consistido en medir y hacer competitivo un modelo creado desde una plantilla en con el dataset de perros y gatos en la plataforma Kaggle. Más que simplemente buscar la precisión más alta, mi objetivo personal fue trazar una ruta clara de optimización: comenzar con una arquitectura simple sacada de la plantilla que se nos proporcionó y escalar el modelo, probando técnicas de la unidad sobre Deep Learning para encontrar el mayor rendimiento.

La memoria está estructurada en 10 iteraciones, cada una documentada en su propio notebook y resumida aquí, mostrando los cambios, hipótesis, resultados y aprendizajes de cada paso. El modelo final se encuentra separado de los demás, todos los notebooks intermedios se encuentra en la carpeta de MODELOS DEPRECATED. Así se puede seguir la evolución completa, desde los primeros experimentos hasta lo que ha sido el modelo final.

Iteración 1: CNN Básica

Descripción del Cambio:

Implementación de una CNN básica desde cero con 3 bloques convolucionales (32, 64, 128 filtros), cada uno seguido de MaxPooling2D. La arquitectura se completa con una capa Dense de 512 neuronas y una salida softmax de 2 clases. Se utilizó un tamaño de imagen de 256×256 píxeles, batch_size de 125 y 12 épocas de entrenamiento con el optimizador RMSprop.

Arquitectura: Conv2D(32) → MaxPool → Conv2D(64) → MaxPool → Conv2D(128) → MaxPool → Flatten → Dense(512) → Dense(2, softmax)

Hipótesis/Justificación:

Como primera aproximación al problema, quería establecer un baseline con una arquitectura convolucional simple pero funcional. Mi hipótesis era que una red con 3 capas convolucionales sería suficiente para capturar características básicas de perros y gatos (bordes, texturas, formas simples), pero probablemente tendría problemas de overfitting al no contar con técnicas de regularización más allá de la arquitectura básica.

Resultado Obtenido:

- **Validation Accuracy:** ~0.70-0.75
- **Kaggle Public Score:** No presentado (prueba local)
- **Observaciones:** El modelo mostraba signos claros de overfitting, con una diferencia significativa entre accuracy de entrenamiento y validación.

Conclusiones y Próximos Pasos:

El modelo baseline funcionó como punto de partida, pero como esperaba, sufría de overfitting severo. La arquitectura captura características básicas pero no generaliza bien. El siguiente paso lógico es implementar data augmentation para aumentar artificialmente el dataset y forzar al modelo a aprender características no variantes a transformaciones, lo que debería mejorar significativamente la capacidad de generalización.

Referencias:

- [Keras Conv2D Layer](#)
 - [Keras: Image Classification from Scratch](#)
 - [YouTube: CNN desde cero en Keras \(Español\)](#)
 - [42 AI School - Deep Learning](#)
-

Iteración 2: CNN con Data Augmentation

Fecha: 04/11/2025

Descripción del Cambio:

Mantuve la misma arquitectura CNN de la iteración 1, pero añadí una capa de data augmentation al inicio del modelo con transformaciones aleatorias: RandomFlip (horizontal), RandomRotation (20%), RandomZoom (20%). También agregué una capa Dropout(0.5) antes de la salida para mayor regularización. Reduje el batch_size a 32 para mejorar la generalización y aumenté las épocas de entrenamiento a 20.

Hipótesis/Justificación:

Mi hipótesis era que el data augmentation obligaría al modelo a aprender características que no variaran en transformaciones geométricas y de iluminación, reduciendo significativamente el overfitting observado en la iteración anterior. Las transformaciones aleatorias simulan diferentes condiciones de captura de las imágenes, aumentando efectivamente el tamaño del dataset de entrenamiento sin necesidad de más datos reales.

Resultado Obtenido:

- **Validation Accuracy:** 0.75 → 0.78
- **Kaggle Public Score:** 0.58768
- **Observaciones:** Aunque la validation accuracy mejoró ligeramente, el Kaggle score fue sorprendentemente bajo, indicando un problema serio con las predicciones.

Conclusiones y Próximos Pasos:

El data augmentation ayudó a reducir el overfitting como esperaba, pero el bajo score de Kaggle. El próximo paso es probar Transfer Learning con VGG16, una arquitectura pre-entrenada que debería capturar características más complejas que mi CNN básica, pero primero necesito corregir el problema de clasificación binaria.

Referencias:

- [Keras: Data Augmentation Layers](#)
 - [TensorFlow: Data Augmentation Tutorial](#)
 - [YouTube: Data Augmentation in Keras](#)
 - [42 AI School - Data Augmentation Notebook](#)
-

Iteración 3: Transfer Learning VGG16 (Modo Categorical - Bug)

Fecha: 04/11/2025

Descripción del Cambio:

Primera implementación de Transfer Learning usando VGG16 pre-entrenado en ImageNet. Congelé todas las capas convolucionales de VGG16 y añadí un cabezal personalizado: Dense(256) + Dropout(0.5) + Dense(2, softmax). Reduje el tamaño de imagen a 224×224 (requerimiento de VGG16) y mantuve data augmentation conservador (flip horizontal, rotation ±10%, zoom ±10%).

Hipótesis/Justificación:

Mi hipótesis era que las características aprendidas por VGG16 en ImageNet (1.4M imágenes, 1000 clases) serían transferibles al problema de perros vs gatos, ya que ImageNet contiene múltiples clases de animales. Transfer Learning debería permitir aprovechar representaciones visuales sofisticadas sin el coste computacional de entrenar desde cero, superando ampliamente el rendimiento de mi CNN básica.

Resultado Obtenido:

- **Validation Accuracy:** ~0.82
- **Kaggle Public Score:** 0.54664
- **Observaciones:** Aunque la validation accuracy era prometedora, el score de Kaggle fue incluso peor que la iteración anterior.

Conclusiones y Próximos Pasos:

El problema de la categorización del label con softmax(2) seguía presente, causando predicciones incorrectas en Kaggle a pesar de una validation accuracy aparentemente buena. VGG16 definitivamente tiene el potencial de extraer mejores características que mi CNN básica, pero necesito corregir urgentemente el bug de clasificación binaria. La próxima iteración debe usar el label mode binario con dense(1, sigmoid para obtener los verdaderos resultados de Transfer Learning.

Referencias:

- [Keras: VGG16 Application](#)
- [Keras: Transfer Learning Guide](#)
- [Paper: Very Deep Convolutional Networks for Large-Scale Image Recognition \(VGG\)](#)
- [YouTube: Transfer Learning with VGG16 in Keras](#)
- [42 AI School - Transfer Learning](#)

Iteración 4: Transfer Learning VGG16 (Corrección Binary)

Fecha: 05/11/2025

Descripción del Cambio:

Corrección crítica del bug anterior: cambié el label a binario y modifiqué la capa de salida a binary cross entropy como función de pérdida. Mantuve VGG16 congelado como base y el mismo cabezal Dense(256) + Dropout(0.5). Entrené durante 15 épocas con early stopping (patience=5) y data augmentation.

Hipótesis/Justificación:

Esta corrección debería alinear perfectamente el modelo con el problema de clasificación binaria real. Mi hipótesis era que con la configuración correcta, VGG16 mostraría su verdadero potencial y superaría ampliamente las iteraciones anteriores, ya que las características pre-entrenadas en ImageNet deberían ser altamente transferibles a la distinción perro/gato.

Resultado Obtenido:

- **Validation Accuracy:** 0.82 → 0.87
- **Kaggle Public Score:** 0.58768 → 0.86380 (Posición #7)
- **Mejora:** +27.6 puntos porcentuales en Kaggle

Conclusiones y Próximos Pasos:

La corrección del bug reveló que el TL sí que estaba funcionando. VGG16 superó ampliamente todas las iteraciones anteriores, confirmando que las características pre-entrenadas son mejores.

Referencias:

- [Keras: BinaryCrossentropy Loss](#)
 - [TensorFlow: Transfer Learning Tutorial](#)
 - [YouTube: Binary Classification with Keras](#)
 - [42 AI School - Fine Tuning](#)
-

Iteración 5: Fine-Tuning VGG16

Fecha: 05/11/2025

Descripción del Cambio:

Implementé Fine-Tuning descongelando las últimas 4 capas convolucionales de VGG16 (de un total de 13). Tras el Transfer Learning inicial, reduje el learning rate a $1e-5$ (100 veces menor) y entrené 10 épocas adicionales permitiendo que estas capas se adapten específicamente al problema de perros vs gatos. Mantuve el resto de la configuración igual.

Hipótesis/Justificación:

Mi hipótesis era que mientras las capas iniciales de VGG16 capturan características genéricas (bordes, texturas) que son universales, las capas finales pueden beneficiarse de especialización para el problema específico.

Resultado Obtenido:

- **Validation Accuracy:** $0.87 \rightarrow 0.89$
- **Kaggle Public Score:** $0.86380 \rightarrow 0.88520$
- **Mejora:** +2.14 puntos porcentuales en Kaggle

Conclusiones y Próximos Pasos:

El Fine-Tuning funcionó como esperaba, logrando una mejora moderada las capas superiores de VGG16 se adaptaron a las características específicas de perros y gatos en nuestro dataset. Sin embargo, VGG16 es computacionalmente costoso y el margen de mejora parece estar llegando a un techo. El siguiente paso es explorar arquitecturas más modernas y eficientes como la familia EfficientNet.

Referencias:

- [CS231n: Transfer Learning](#)
 - [Keras: Fine-tuning a Keras model](#)
 - [YouTube: Fine-Tuning VGG16 in Keras](#)
 - [42 AI School - Fine Tuning Notebook](#)
-

Iteración 6: Ensemble de Modelos

Fecha: 06/11/2025

Descripción del Cambio:

Implementé un ensemble combinando 3 modelos diferentes: VGG16 (Fine-Tuned), ResNet50 (Transfer Learning) y EfficientNetB3 (Transfer Learning). Cada modelo generó predicciones independientes sobre el test set, y promedí las probabilidades de salida para obtener la predicción final. Entrené cada modelo con las mejores configuraciones descubiertas hasta ahora.

Hipótesis/Justificación:

Mi hipótesis era que diferentes arquitecturas aprenden representaciones complementarias del mismo problema. Al promediar sus predicciones, los errores individuales deberían cancelarse, mejorando la robustez general. Esta técnica es ampliamente utilizada en competiciones de ML.

Resultado Obtenido:

- **Validation Accuracy (Individual):** 0.89 (VGG16), 0.87 (ResNet50), 0.88 (EfficientNetB3)
- **Kaggle Public Score:** 0.88520 → 0.90230
- **Mejora:** +1.71 puntos porcentuales

Conclusiones y Próximos Pasos:

El ensemble logró una mejora como esperaba, confirmando que las diferentes arquitecturas capturan información complementaria. Sin embargo, el ensemble es computacionalmente muy costoso (entrenar y ejecutar 3 modelos) y no fué tan significativa como se esperaba la mejora por lo que vistos los resultados por separado de cada modelo lo mas sensato es tirar por la familia EfficientNetB3 pero sin ensemble.

Referencias:

- [Machine Learning Mastery: Ensemble Methods for Deep Learning](#)
 - [Kaggle: Ensembles for Deep Learning](#)
 - [YouTube: Ensembles in Deep Learning](#)
 - [42 AI School - Ensembles](#)
-

Iteración 7: EfficientNetB7 (Primera Implementación)

Fecha: 09/11/2025

Descripción del Cambio:

Migración a EfficientNetB7, el modelo más grande de la familia EfficientNet. Configuré resolución alta (384×384), batch_size=16 (limitado por VRAM), Transfer Learning con 10 épocas + Fine-Tuning de las últimas 30 capas con 10 épocas adicionales. Implementé data augmentation agresivo (flip, rotation 15%, zoom 15%, translation 10%, contrast 15%, brillo de 10%), dropout=0.6, y un cabezal denso más profundo: 512→256→128→1 neuronas con BatchNormalization entre capas.

Hipótesis/Justificación:

EfficientNetB7 es considerado uno de los mejores modelos para imágenes su diseño está optimizado mediante NAS y compound scaling. Mi hipótesis era que B7, con 66M parámetros y optimizado para resoluciones altas, superaría significativamente el ensemble anterior con un solo modelo. La alta resolución (384×384) debería permitir capturar detalles finos que distinguen perros de gatos. Sin embargo está claro que es un modelo sobredimensionado para el dataset y seguramente no valga la pena el tiempo invertido en entrenamiento.

Resultado Obtenido:

- **Validation Accuracy:** 0.92
- **Supplementary Accuracy:** 0.89
- **Kaggle Public Score:** 0.90230 → 0.94962
- **Mejora:** +4.73 puntos porcentuales
- **Tiempo de entrenamiento:** ~5 horas

Conclusiones y Próximos Pasos:

EfficientNetB7 superó ampliamente el ensemble con un solo modelo, confirmando su superioridad arquitectónica. Sin embargo, el tiempo de entrenamiento de 5 horas es prohibitivo para experimentación rápida, y B7 puede ser excesivo para este problema relativamente simple. El modelo es innecesariamente grande (66M parámetros). El siguiente paso es intentar optimizar B7 y sino bajar a B3.

Referencias:

- [Paper: EfficientNet: Rethinking Model Scaling for CNNs](#)

- [Keras: EfficientNet Application](#)
 - [YouTube: EfficientNet Explained](#)
 - [42 AI School - EfficientNet](#)
-

Iteración 8: EfficientNetB7 Optimizado (Sin TTA)

Fecha: 10/11/2025

Descripción del Cambio:

Optimización de la iteración 7 reduciendo épocas de entrenamiento: Transfer Learning de 10→8 épocas y Fine-Tuning de 10→6 épocas. Eliminé Test-Time Augmentation (TTA) que aplicaba 10 transformaciones por imagen en predicción. Mantuve todas las demás configuraciones: resolución 384×384, dropout=0.6, data augmentation agresivo, arquitectura del cabezal denso 512→256→128→1.

Hipótesis/Justificación:

Mi hipótesis era que podía reducir el tiempo de entrenamiento aproximadamente a la mitad sin afectar significativamente el rendimiento, ya que observé que el modelo convergía antes de completar todas las épocas (early stopping frecuente). TTA, aunque mejora robustez, añade overhead computacional 10x en inferencia y puede no ser necesario si el modelo es suficientemente robusto por el data augmentation en training.

Resultado Obtenido:

- **Validation Accuracy:** 0.92 (sin cambios)
- **Kaggle Public Score:** 0.94962 (idéntico)
- **Tiempo de entrenamiento:** ~5 horas → ~2.5 horas (reducción 50%)

Conclusiones y Próximos Pasos:

Resultado interesante: la optimización redujo el tiempo de entrenamiento un 50% sin pérdida alguna de accuracy, confirmando que había épocas redundantes. El modelo converge más rápido de lo que pensaba. Sin embargo, B7 sigue siendo demasiado grande y lento para este problema. La conclusión definitiva es migrar a EfficientNetB3, que con 12M parámetros (5.5x más pequeño que B7) debería ser óptimo para el balance accuracy/velocidad en este problema, permitiendo iterar más rápidamente en optimizaciones de hiperparámetros.

Referencias:

- [Keras: EarlyStopping Callback](#)
 - [Machine Learning Mastery: Test-Time Augmentation](#)
 - [YouTube: Test-Time Augmentation](#)
 - [42 AI School - Augmentation](#)
-

Iteración 9: Migración a EfficientNetB3 V1

Fecha: 10/11/2025

Descripción del Cambio:

Migración completa a EfficientNetB3 manteniendo la arquitectura exitosa de B7. Configuración: resolución 300×300, batch_size=32 (el doble que B7), Transfer Learning 15 épocas + Fine-Tuning 10 épocas con descongelado de las últimas 20 capas. Implementé learning rate schedule: Warmup (2 épocas lineales) + Cosine Decay con AdamW optimizer (weight_decay=1e-4). Mantuve arquitectura densa 512→256→128→1, dropout=0.25 (reducido de 0.6), label_smoothing=0.05 (reducido de 0.1).

Hipótesis/Justificación:

Mi hipótesis era que B3 (12M parámetros) es el punto óptimo para este problema: suficientemente expresivo pero sin la redundancia de B7. El learning rate schedule con warmup debería estabilizar el entrenamiento inicial, mientras que cosine decay permite convergencia suave. Dropout más bajo (0.25 vs 0.6) porque B3 tiene menos tendencia a overfitting que B7. La reducción de label_smoothing (0.05 vs 0.1) permite al modelo ser más confiado en sus predicciones correctas.

Resultado Obtenido:

- **Validation Accuracy:** 0.93
- **Supplementary Accuracy:** 0.9167
- **Kaggle Public Score:** 0.94962 → 0.97030 (Top #1)
- **Mejora:** +2.07 puntos porcentuales
- **Tiempo de entrenamiento:** ~1 hora (5x más rápido que B7)

Conclusiones y Próximos Pasos:

EfficientNetB3 alcanzó el primer puesto de la competición con 5 veces menos tiempo de entrenamiento. Esto confirma que B7 era excesivo para este problema y B3 es la arquitectura óptima. La configuración de hiperparámetros (learning rate schedule, dropout reducido, label smoothing moderado) está bien balanceada. El único experimento pendiente es Test-Time Augmentation (TTA), que fue removido en iteración 8. TTA podría mejorar el score privado si hay overfitting al test set público.

Referencias:

- [Paper: AdamW Optimizer](#)
- [Paper: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour \(Warmup\)](#)
- [Paper: Rethinking the Inception Architecture for Computer Vision \(Label Smoothing\)](#)
- [YouTube: EfficientNetB3 in Keras](#)
- [42 AI School - EfficientNetB3](#)

Iteración 10: EfficientNetB3 V2 (Con TTA - Modelo Final)

Fecha: 10/11/2025

Descripción del Cambio:

Versión final añadiendo Test-Time Augmentation (TTA) con 10 transformaciones aleatorias por imagen en inferencia: flip horizontal, rotation 5%, zoom 5%, translation 3%. Las predicciones incluyen una baseline sin augmentation + 10 versiones aumentadas, promediando todas para la predicción final. Mantuve toda la configuración exitosa de la iteración 9: B3, resolución 300×300, arquitectura densa 512→256→128→1, learning rate schedule, dropout=0.25, label_smoothing=0.05.

Hipótesis/Justificación:

Mi hipótesis era que TTA mejoraría la robustez del modelo promediando predicciones sobre versiones ligeramente perturbadas de cada imagen de test. Aunque la iteración 8 mostró que TTA no mejoró el score público con B7, el contexto es diferente: B3 es menos propenso a overfitting y TTA podría ayudar específicamente en el score privado (que mide generalización real), incluso si no mejora el público. TTA con augmentations suaves (5% rotación vs 15% en training) evita distorsionar demasiado.

Resultado Obtenido:

- **Validation Accuracy:** 0.93 (sin cambios)
- **Supplementary Accuracy:** 0.9167 (sin cambios)
- **Kaggle Public Score:** 0.97030 → 0.97014 (ligera caída de -0.016)

- **Kaggle Private Score:** 0.98305 (Score final revelado)
- **Diferencia público/privado:** +1.29 puntos porcentuales a favor del privado

Conclusiones y Próximos Pasos:

TTA no mejoró el score público, pero el score privado final de 0.98305 sugiere excelente generalización. La diferencia de 1.29 puntos entre público y privado indica que el modelo es robusto y no está overfitteado al test set público. TTA cumplió su objetivo de mejorar generalización. Este es el modelo final para la entrega: EfficientNetB3 con TTA logra un balance óptimo entre accuracy (98.3% privado), eficiencia computacional (1h y media de entrenamiento) y robustez.

Referencias:

- [Kaggle: TTA Discussion](#)
 - [TensorFlow TPU: EfficientNet Best Practices](#)
 - [YouTube: Test-Time Augmentation for Image Classification](#)
 - [42 AI School - EfficientNetB3](#)
-

Conclusión y modelo final

El modelo final, EfficientNetB3 V2 (con TTA) en la decima iteración, ha sido la solución definitiva dado que es la mas equilibrada en precisión y control sobre el overfitting, el modelo es relativamente ligero, por lo que los testeos no han llevado tanto tiempo como en otras iteraciones y los resultados han sido excepcionales.

El camino hacia este resultado no fue lineal ya que las pruebas iniciales demostraron que el Transfer Learning era obligatorio tras el fracaso por overfitting de las arquitecturas CNN básicas en las dos primeras iteraciones y la clave (y mi gran reto) fue encontrar la arquitectura perfecta, descartando modelos como VGG16 (I5) y B7 (I7) por ser demasiado lentos o sobredimensionados así como metodologías innecesarias como esemble learning.

EfficientNetB3 como modelo ha acabado siendo con diferencia el mas optimo, ofreciendo una precisión alta sin ganar demasiado overfitting el cual ha sido relativamente facil de controlar con Learning Rate Schedule, el Label Smoothing, o finalmente con el TTA en la Iteración 10. TTA en concreto garantizó que el modelo no solo fuera preciso en la validación, sino que también generalizara a la perfección, lo que se reflejó en el excelente score privado final de 0.98305, consolidando al B3 con TTA como mi modelo definitivo de la actividad.

- Ismael Hernández Clemente 21/11/2025 - 21:08