WorkShop 1

Image Classification con Redes Neuronales Convolucionales (CNN)

1. Use el cuaderno de Keras sobre TensorFlow: **Keras_Clasificacion_de_Digitos_Usando_Redes_Convolucionales.ipynb** (para el conjunto de digitos hechos a mano Mnist). para clasificar los números a mano.

   - Una vez tenga el modelo anterior descarge de internet un par de imágenes de números modifíquelas al tamaño del dataset Mnist y haga una predicción usando el modelo anterior.

   - Use modelos pre-entrenados (Transfer Learning), compare y saque conclusiones comparando diferentes arquitecturas MobileNet, VGG16, EfficientNet, y ResNet. Cuál funcionó mejor?

```
================================================================
=========
COMPARISON RESULTS
================================================================
=========
LeNet-5 (Original):     0.9637
----------------------------------------
VGG16                : 0.9570 (Time: 3149.0s)
ResNet50             : 0.9262 (Time: 1583.5s)
MobileNet            : 0.4964 (Time: 427.3s)

================================================================
=========
CONCLUSIONS
================================================================
=========
1. Best performing model: VGG16 with 0.9570 accuracy
2. LeNet-5 outperformed transfer learning by 0.0067
3. Transfer learning observations:
   - VGG16: Simple architecture, good baseline
   - ResNet: Deep architecture with skip connections
   - MobileNet: Lightweight, good for mobile deployment
4. For MNIST digit classification:
   - Simple architectures like LeNet-5 are often sufficient
   - Transfer learning may be overkill for this simple task
   - Pre-trained models excel more on complex, natural image
tasks
```

2. Corra el cuaderno **Clasificacion_de_perros_y_gatos_KERAS.ipynb**, hágalo en Colab o en las máquinas de Kaggle, use solo 2000 imágenes para entrenamiento y 200 para validación.

- Compare el modelo anterior con diferentes modelos pre-entrenandos (EfficientNetB0, ResNet50 y VGG16).

- Ahora descargue de internet un par de imágenes de perros y gatos y páselas por el mejor modelo obtenido anteriormente y escriba sus conclusiones.

Key Observations and Conclusions

Performance Comparison:

EfficientNetB0: 49.97% test accuracy

ResNet50: 62.78% test accuracy

VGG16: 50.18% test accuracy

Best Performing Model:

ResNet50 achieved the highest accuracy of 62.78% on the test set.

Training Observations:

Stability and convergence: With frozen backbones and a small classification head, training is generally stable. Given the relatively low test accuracies (especially for EfficientNetB0 and VGG16 near chance), the models likely plateaued quickly, suggesting limited feature transfer or a domain/setup mismatch.

Over/underfitting: The gap between training and validation accuracy (not shown here) should be checked, but the low test results suggest underfitting or insufficient

adaptation of the pre-trained features to the dataset rather than classic overfitting. Freezing all backbone layers may have constrained representational adaptation.

Relative compute/time: From the test-time traces, evaluation latency indicates ResNet50 ran slower than EfficientNetB0 and faster than VGG16:

EfficientNetB0 evaluate: ~43s

ResNet50 evaluate: ~127s

VGG16 evaluate: ~379s This aligns with expectations: VGG16 is parameter-heavy and slow; EfficientNetB0 is compute-efficient; ResNet50 sits in between. Training times typically follow a similar ordering per epoch.

Recommendations:

Fine-tune the backbone:

Unfreeze the top 10–30% of layers of the best model (ResNet50) and fine-tune with a small learning rate (1e-5 to 3e-5). Consider a two-stage schedule: warm up the head, then fine-tune the top blocks.

Use ReduceLROnPlateau or cosine decay, and keep EarlyStopping(monitor='val_accuracy', patience=3–5).

Improve regularization and augmentation:

Add label smoothing (e.g., 0.05), L2 weight decay on dense layers, and richer augmentations (random brightness/contrast, color jitter, random crops).

Data and input pipeline:

Verify the dataset balance and labeling. Ensure train/ and test/ are properly split and representative.

Increase image size modestly (e.g., 256–288) and/or try EfficientNetB1/B2 or ResNet50V2 for potential gains if GPU memory allows.

Training diagnostics:

Plot training vs. validation curves (already set up) and confusion matrices to identify systematic errors.

Evaluate with ROC-AUC and per-class precision/recall to understand failure modes.

Practical deployment:

After improvements, consider mixed precision for speed and TFLite export if targeting edge devices.

Final Thoughts:

ResNet50 currently provides the best performance, but the overall accuracies indicate the frozen-feature approach wasn’t sufficient for this dataset. Fine-tuning selective backbone layers, stronger augmentation, and careful LR scheduling typically unlock substantial gains for Cats vs Dogs. With these adjustments, it’s reasonable to expect test accuracy to move well beyond the current 62.78% ceiling.