

## Reporte 4: Convolutional Neural Networks

### Introducción

El siguiente reporte contiene el análisis realizado para un modelo de clasificación de imágenes para la detección de especies de Monos. Para ello se tiene un dataset extraído desde Kaggle con cerca de 1400 imágenes correspondientes a 10 especies de Monos. El objetivo del reporte es utilizar distintas arquitecturas de Redes Convolucionales para poder clasificar cada imagen con la especie correcta. El modelo fue entrenado en **Pytorch**[1] y **Pytorch Lightning**[2] utilizando arquitecturas pre-entrenadas extraídas desde la librería **timm**[3]. El modelo se validará utilizando un esquema de *Validación Holdout*, predefinido de antemano por los mismos organizadores de Kaggle que cuenta con 1097 imágenes de entrenamiento y 272 de validación. Se reportarán distintas métricas de Clasificación tales como *Accuracy* y *Macro F1-Score*.



### 1. Convolutional Neural Networks

Las Redes Convolucionales (CNN) son un tipo de red neuronal artificial que se volvió particularmente popular para aplicaciones de visión computacional. Dentro de sus propiedades destaca su capacidad para ser invariantes al desplazamiento y al espacio y están basadas en el concepto de compartir pesos cuando se crean feature maps luego de aplicar convoluciones en distintos filtros o kernels.

Las redes convolucionales están inspiradas en procesos biológicos similares a la corteza visual de los animales, en la que sólo ciertas neuronas responden a los estímulos visuales de una imagen (campo receptivo). La imagen es entendida porque el campo receptivo de distintas partes del ojo se superponen, algo que las Redes Convolucionales tratan de emular. Una de las primeras arquitecturas de Redes Convolucionales fue Propuesta por Lecun[4] para el reconocimiento de códigos postales (ZIP Codes) en el año 1998, el cual dio inicio a una búsqueda por Arquitecturas más poderosas que pudieran identificar elementos cada vez más complejos en imágenes.

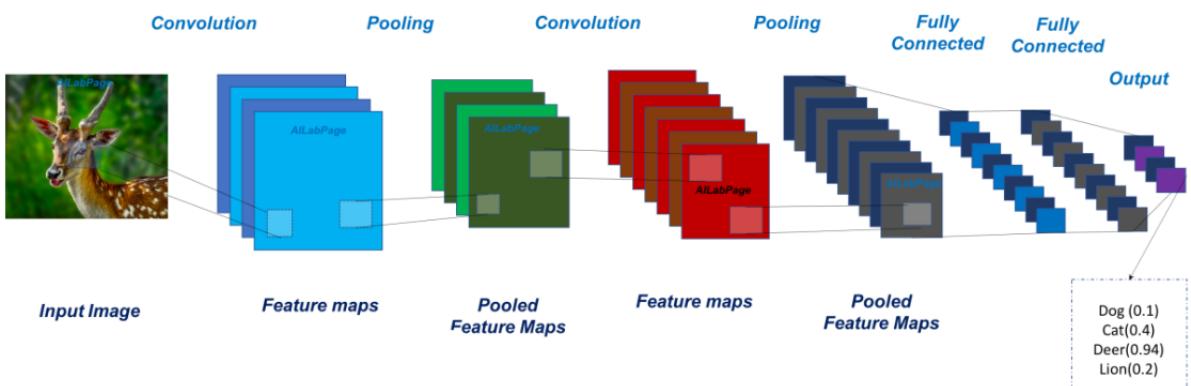


Figura 1: Representación de una Red Convolucional.

En la Figura 1 se puede observar las principales componentes de una Red Convolucional, las cuales se detallan a continuación:

- **Redes Convolucionales:** Son aquellas que permiten detectar las features importantes de la imagen generando feature maps.
- **Capas de Pooling:** Son las encargadas de ir disminuyendo el espacio de búsqueda de features en la imagen. Esto permitirá que capas posteriores puedan encontrar features cada vez más finas.
- **Capa Flatten:** Permitirá que los feature maps puedan tomar una forma vectorial (plana) para alimentar las capas Fully Connected que siguen a continuación.
- **Capas Fully Connected:** Son las encargadas de llevar a cabo la clasificación final de la imagen.

A lo largo de este trabajo se explorarán 3 arquitecturas famosas las cuales se detallan a continuación:

### 1.1. AlexNet

*AlexNet*[5] nace como una necesidad de mejorar el *ImageNet Challenge* (un concurso de visión computacional en el que se probaba el rendimiento de las arquitecturas de visión). *AlexNet* fue la primera arquitectura de aprendizaje profundo en lograr una diferencia considerable en Accuracy **ImageNet LSVRC-2012** logrando un 84.7 % (el segundo lugar obtuvo un 73.8 %). La idea principal de esta arquitectura (ver Figura 2) es aprovechar las capas convolucionales y los campos receptivos. Principalmente la red está compuesta de 5 capas convolucionales y 3 capas Fully Connected (MLP). Cada capa está activada utilizando ReLU.

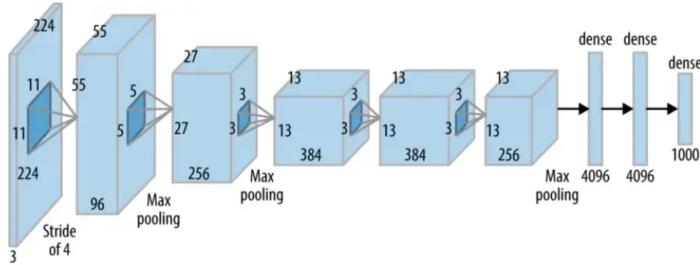


Figura 2: Representación de una Arquitectura AlexNet.

### 1.2. ResNet

*ResNet*[6] fue introducida en el **ImageNet LSVRC-2012** para abordar principalmente el problema de *vanishing gradient*. Esto debido a que, en la búsqueda de mejores rendimientos, se comenzaron a agregar más y más capas. *ResNet* introduce las conexiones residuales las cuales permiten evitar dicho problema y transmitir gradientes incluso en redes muy profundas. Existen varias versiones de esta red caracterizada por su número de capas, partiendo en la *ResNet-18* hasta la *ResNet-200* (Ver Figura 3).

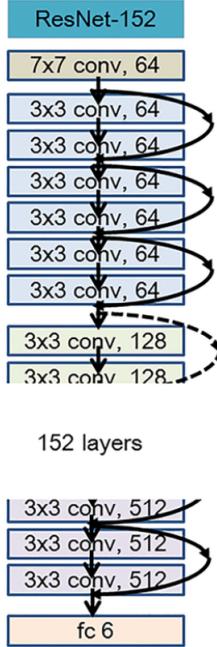


Figura 3: Representación de una ResNet-152.

### 1.3. EfficientNet

Una de las arquitecturas modernas más famosas es la EfficientNet[7]. Esta arquitectura utiliza un método para escalar uniformemente todas las dimensiones (profundidad, ancho y resolución) utilizando un cierto coeficiente. Este coeficiente se justifica por la intuición de que imágenes más grandes requerirán más capas para incrementar el campo receptivo y más canales para capturar patrones más finos. La EfficientNet está creada en 8 tamaños desde B0 (Ver Figura 4) hasta B7.

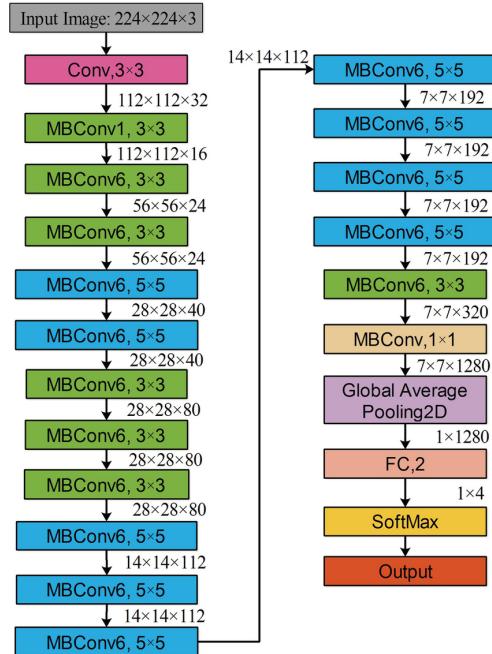


Figura 4: Representación de una EfficientNetB0.

## Transfer Learning

Transfer Learning se refiere al proceso en el que el aprendizaje de una red neuronal aprendida para una tarea en específica, puede ser utilizada para realizar una tarea distinta. En este caso, la mayoría de los modelos de visión han sido pre-entrenados en ImageNet [8], un dataset estándar que contiene 14.197.122 imágenes anotadas en 1000 categorías.

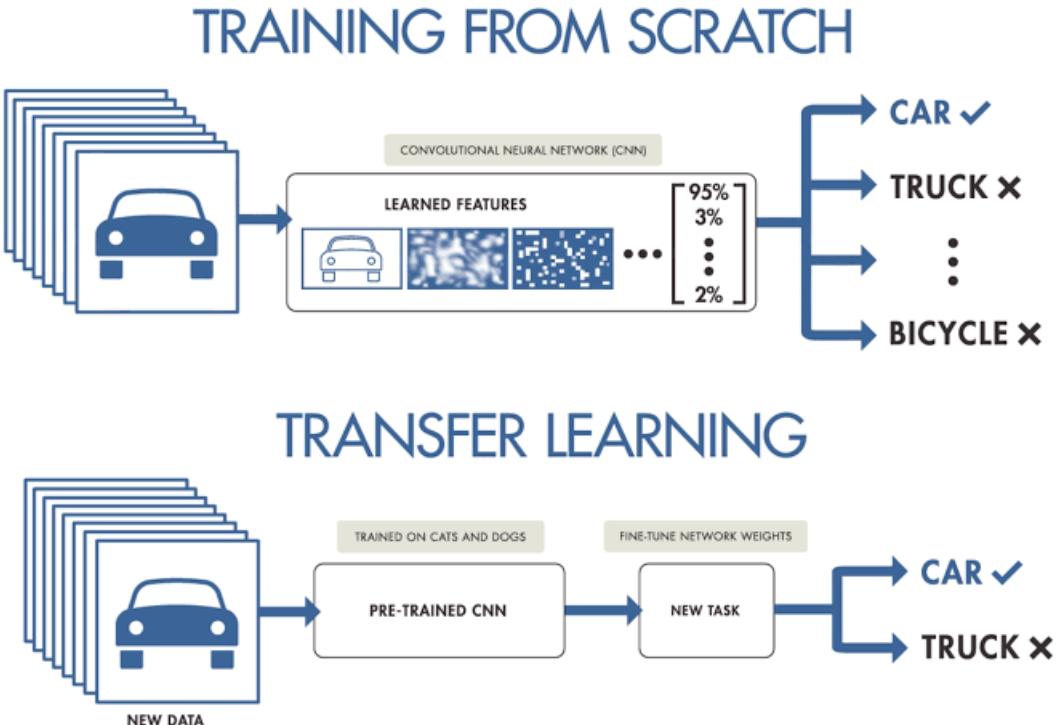


Figura 5: Esquema de funcionamiento de Transfer Learning.

## 2. Los Datos

Los datos a utilizar corresponden a un dataset extraído desde Kaggle que contiene cerca de 1400 imágenes asociadas a 10 especies de monos. Cada imagen corresponde a una fotografía en formato **.jpeg**. La imagen comúnmente será de tamaño 400x300 píxeles o superior. Las imágenes corresponden a tomas de una especie de mono que puede estar en cualquier posición (puede ser foto de cuerpo entero o sólo la parte superior). Las clases correspondientes a las distintas especies se detallan a continuación:

- **n0: alouattapalliata:** El mono aullador negro.
- **n1: erythrocebuspatas:** El mono rojo.
- **n2: cacajaocalvus:** El uacarí calvo.
- **n3: macacafuscata:** El macaco japonés.
- **n4: cebuellapygmaea:** El tití pigmeo.
- **n5: cebuscapucinus:** El mono carablanca.
- **n6: micoargentatus:** El tití plateado.
- **n7: saimirisciureus:** El mono ardilla común.

- **n8:** *aotusnigriceps*: El mico nocturno de cabeza negra.
- **n9:** *trachypithecusjohnii*: El Langur de Nilgiri.

### 3. Análisis Exploratorio

Como dijimos anteriormente el dataset posee cerca de 1400 imágenes. En la Figura 6 se puede observar la distribución de imágenes por clase para entrenamiento y validación del dataset. En el set de entrenamiento cada clase posee entre 100 y 120 imágenes, mientras que el Test Set tiene entre 25 y 30 imágenes por clase.

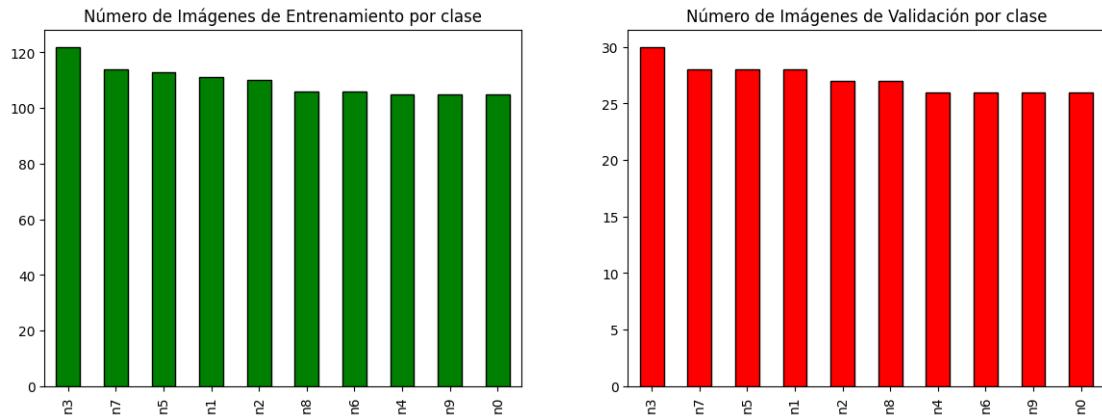


Figura 6: Distribución de Imágenes por Clase de Entrenamiento (Izquierda) y Test (Derecha)

En la Figura 8 se puede ver una muestra del tipo de imágenes que incluye el dataset:

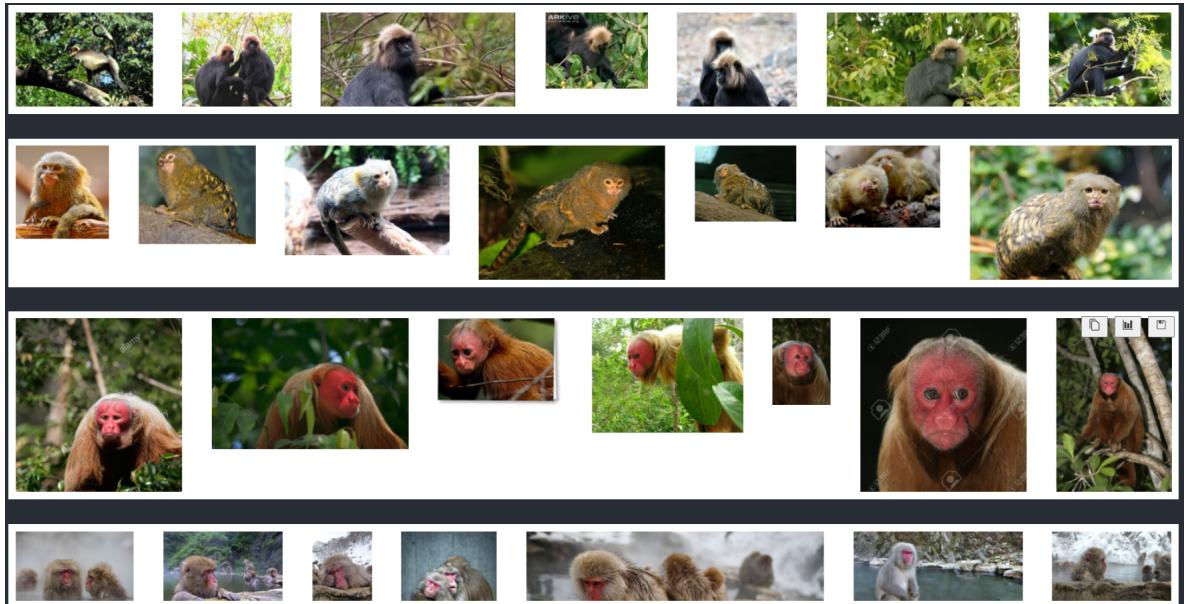


Figura 7: Muestra de Imágenes

Se puede apreciar claramente que las imágenes son bastante heterogéneas y que no existe un patrón estable en la pose que puede tener un mono en la imagen. Además, pueden existir imágenes que tengan

algún tipo de marca de agua, o el mono puede estar cubierto por hojas o partes de algún árbol. Debido a que cada imagen posee distintos tamaños es que será necesario estandarizar esto para permitir que ingresen a la red neuronal.

## 4. Modelamiento

El proceso de Modelamiento contemplará la comparación de 3 Arquitecturas las cuales se compararán utilizando Transfer Learning. Para ello se modificará el proceso de entrenamiento, congelando (freezing) el *backbone* o permitiendo un *fine-tuning*.

El *freezing* corresponde a utilizar el *backbone* sin optimizar sus parámetros. Es decir sólo se optimiza la cabeza de predicción, mientras que en el *fine-tuning* se pueden también actualizar los pesos del *backbone*.

### 4.1. Preprocesamiento

Es importante entender que las Arquitecturas utilizadas fueron entrenadas con un cierto preprocesamiento. Para obtener los mejores resultados es importante intentar replicar estos preprocesamientos:

- **Resize** de la imagen a 256x256.
- **CenterCrop** de la imagen a 224x224.
- **Normalización** por canal:
  - **Media:** [0.485, 0.456, 0.406]
  - **Desviación Estándar:** [0.229, 0.224, 0.225]

El proceso de entrenamiento está configurado con los siguientes Hiperparámetros:

- **Batch Size:** 32
- **Optimizador:** Adam
- **learning\_rate** = 0.001

### 4.2. Detalles de Arquitecturas

#### Vanilla CNN

- **Backbone:** 3 Bloques Convolucionales compuestos de una Red Convolucional, ReLU y Max Pooling.
- **Número de Parámetros:** 1.481.48
- **Head:**
  - Capa Salida de 256 a 10 Clases de Salida.

#### AlexNet

- **Backbone:** AlexNet
- **Número de Parámetros:** 11.918.154
- **Head:**
  - Capa Oculta 1024 con Relu y 40 % de Dropout.
  - Capa Salida de 1024 a 10 Clases de Salida.

## ResNet

- **Backbone:** ResNet34
- **Número de Parámetros:** 21.287.242
- **Head:**
  - Capa Oculta 256 a 10 Clases de Salida.

## EfficientNet

- **Backbone:** EfficientNetB0.
- **Número de Parámetros:** 4.012.678
- **Head:**
  - Capa Oculta 512 a 10 Clases de Salida.

### 4.3. Evaluación

Debido a que se trata de un modelo multiclase se medirá la performance del modelo mediante Accuracy, Macro F1-Score. Además se mostrará la Matriz de Confusión.

### 4.4. Resultados del Modelo

Arquitectura	Frozen	Accuracy	Macro F1
VanillaCNN	-	0.4632	0.4611
AlexNet	Yes	0.9412	0.9418
AlexNet	No	0.1103	0.0199
Resnet34	Yes	0.9669	0.9664
Resnet34	No	0.8971	0.8981
<i>EfficientNetB0</i>	Yes	0.9485	0.9482
<b>EfficientNetB0</b>	<b>No</b>	<b>0.9816</b>	<b>0.9812</b>

Tabla 1: Resultados Entrenamiento

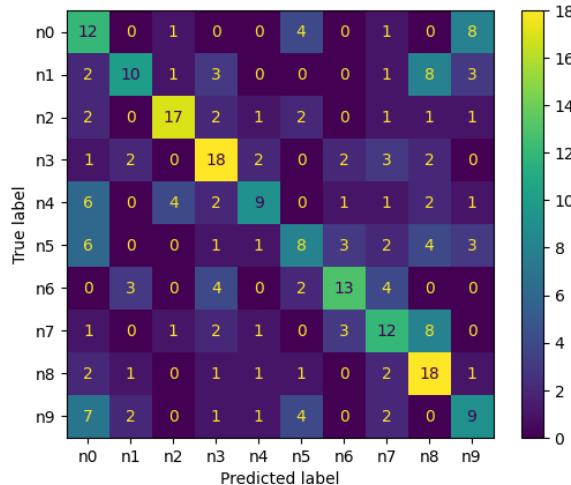


Figura 8: Matriz de Confusión para una CNN Convencional.

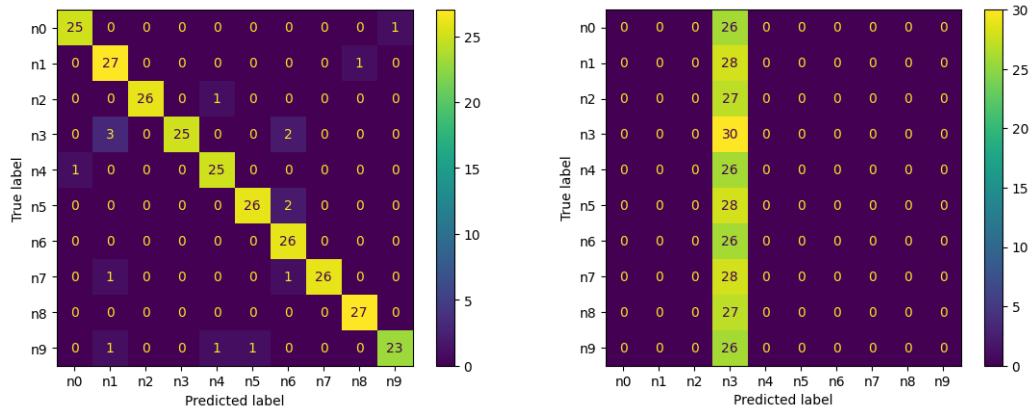


Figura 9: Matriz de Confusión para Modelo AlexNet para parámetros Frozen (izquierda) y not Frozen (derecha).

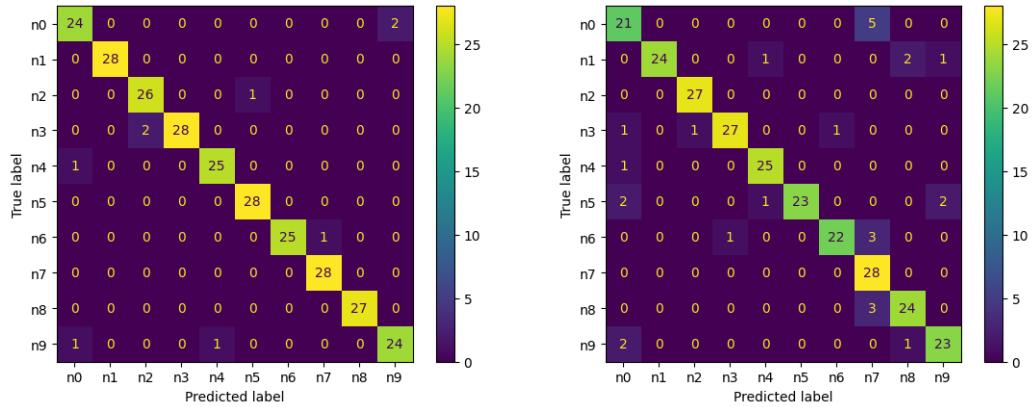


Figura 10: Matriz de Confusión para Modelo ResNet para parámetros Frozen (izquierda) y not Frozen (derecha).

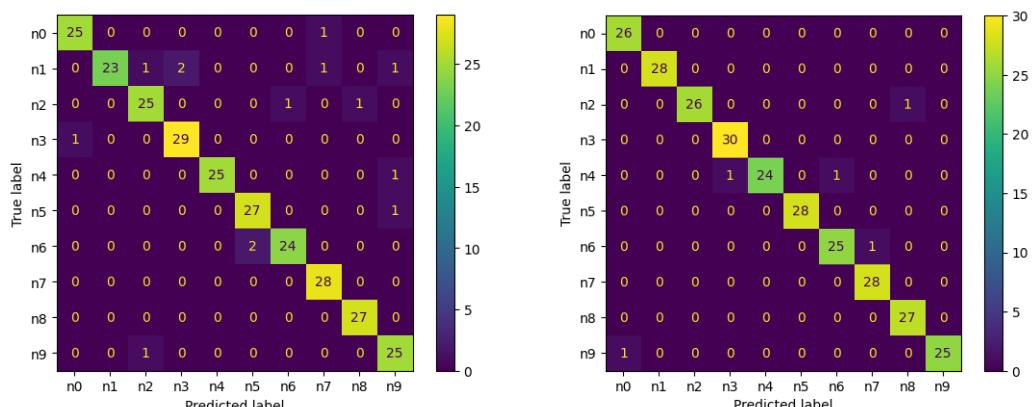


Figura 11: Matriz de Confusión para Modelo EfficientNet para parámetros Frozen (izquierda) y not Frozen (derecha).

## 5. Conclusiones

- Luego de realizar varios experimentos se puede concluir que efectivamente las Redes Neuronales Convolucionales son una excelente opción para la clasificación de Imágenes. Los puntajes obtenidos son bastante buenos a pesar de la dificultad de la tarea solicitada.
- Se puede ver que resolver un problema de clasificación de imágenes no es una tarea sencilla, y utilizar una Red Convencional no es tan efectiva como aplicar una Arquitectura diseñada específicamente para tareas de clasificación.
- En este proceso comparamos 4 arquitecturas, todas con un número sumamente significativo de parámetros. En el caso de la CNN convencional se cuenta con cerca de 1.5 Millones de parámetros. Lamentablemente sólo utilizar CNNs no entregó buenos resultados.
- Se pudo comparar que el número de parámetros no es el aspecto más importante de una Red. **AlexNet** cuenta con cerca de 12M de Parámetros, **ResNet-34** con 21M y sorprendentemente **EfficientNetB0** con tan sólo 5M fue la que entregó los mejores resultados de la experimentación.
- Se puede concluir que efectivamente agrupar capas de manera inteligente (como en EfficientNet) es más efectivo que sólo generar redes extremadamente profundas (como en ResNet, aunque también entregó excelentes resultados).
- El proceso de *Transfer Learning* funciona perfecto. Todas las redes utilizadas (a excepción de la **Vanilla CNN**, usada sólo como referencia) fueron pre-entrenadas en ImageNet y obtuvieron una mejor performance por mucho, en número muy pequeño de Epochs.
- En general congelar el **backbone** terminó dando mejores resultados, con excepción de la **EfficientNetB0 la cual funcionó mejor sin congelar**. Un aspecto que me llamó bastante la atención fue que la performance obtenida por **AlexNet sin congelar fue extremadamente pobre**, mostrando que parece ser que los pesos obtenidos con Imagenet funcionan bastante bien.

## Referencias

- [1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.
- [2] W. Falcon and The PyTorch Lightning team, “PyTorch Lightning,” 2019.
- [3] R. Wightman, “Pytorch image models.” <https://github.com/rwightman/pytorch-image-models>, 2019.
- [4] Y. Lecun, L. E. Bottou, Y. Bengio, and P. H. Abstract—, “Gradient-based learning applied to document recognition,”
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,”
- [6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,”
- [7] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,”
- [8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.