# ECI2019 - COMPETENCIA DESPEGAR
# CLASIFICADOR DE IMÁGENES DE HOTELES

**Leticia L. Rodríguez**

July 13, 2019

**RESUMEN**

En el presente informe se detalla el proceso llevado a cabo para a elaboración de un clasificador de imágenes de Hoteles en el contexto de las Competencias de Datos de la ECI 2019.

***K*eywords** Clasificación de Imágenes · Tranfer Learning · Redes Convolucionales · Optimización de Redes Neuronales

## 1 Sobre Pytorch

Pytorch sobre Tensorflow fue el framework elegido para la implementación. A diferencia de Keras, Pytorch provee mayor performance y mayor flexibilidad para implementar optimizaciones para el entramiento.

La elección de Pytorch también se vio influída por su simple sintaxis y la facilidad para cargar y pre-procesar las imágenes.

## 2 Preparación de los datos

El pre-procesamiento de datos se vió relacionado con el framework elegido, Pytorch.

### 2.1 Organización del filesystem

Pytorch provee una forma simple de leer las imágenes desde el disco con sus correspondientes categorías utilizando la clase ImageFolder.

Para que pueda asignar las categorias a cada imagen, precisa que los datos estén organizados de forma que cada categoria tenga una carpeta y dentro las imágenes correspondientes.

Así que la primer tarea fue descargar las imágenes de testeo y entrenamiento, y organizarlas de forma que puedan ser leidas fácilmente con por el framework con su correspondiente categoria. Dado que las categorias o labels se encontraban en un csv, se usaron scripts de python que organicen los archivos como se indica en la figura 1

### 2.2 Construcción de los conjuntos de entrenamiento, validación y testeo

En la competencia se proveen de sets de entrenamiento y testeo. Decidí reservar una porción de los datos de entrenamiento como validación de forma que sirvan para observar el progreso del entrenamiento y sobre todo condiciones como Overfitting (Sobreajuste). El 20% de los datos seleccionados de manera aletoria del conjunto de entrenamiento fue usado como validación.

### 2.3 Data Augmentation o Image Augmentation

Por último, Pytorch permite construir un DataLoader que va a contener las imágenes pre-procesadas con transformaciones que indiquemos. En el caso del dataset de entrenamiento, podemos especificar transformaciones que agreguen Image Augmentation, es decir, que apliquen transformaciones como rotaciones, flipeos, redimensionamientos, crops, etc para darle variabilidad a los datos.

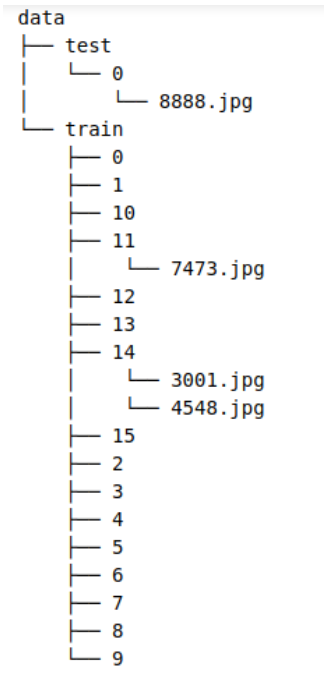


Figure 1: Organización de las imágenes en directorios para ser leidos por Pytorch

En Pytorch, estas tranformaciones se aplican en las épocas sin agrandar el dataset pero generando diferencias en una misma imagen entre las épocas lo que permite que se puedan reajustar los pesos acorde a las diferentes variaciones.

Hay varias tranformaciones disponibles provistas por Pytorch e incluso se pueden codear propias. En particular, trabajé con el siguiente pipeline de transformaciones:

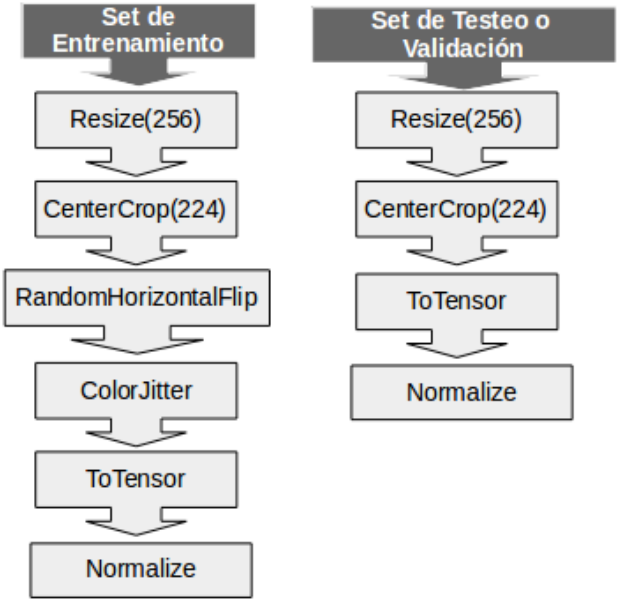


Figure 2: Pipeline de transformaciones aplicadas a los conjuntos de datos

Justificación:

- Resize(256) → CenterCrop(224): Dado que el modelo va a usar Transfer Learning sobre modelos entrenados con tamaños de entrada 224x224x3, las imagenes son redimensionadas a 256x256x3 para luego, mediante CenterCrop, ignorar los bordes quedándo con el centro, donde podría estar la información más relevante.
- ToTensor: Transforma los bits en un tensor
- Normalize: Normaliza los datos para mejorar convergencia

Para el dataset de entrenamiento, además se incluyeron las siguientes transformaciones:

- RandomHorizontalFlip: Debido al tipo de problema, una habitación vista de derecha a izquierda, o de izquierda a derecha pertenece a una misma categoría. Por ejemplo, una silla apuntando hacia la izquierda o la derecha no cambia la detección y poder invertir las imagenes permite darle mayor información al clasificador.
- ColorJitter: Aplica cambios aletorios al brillo, constranste y saturación de la imagen. La idea es despegar un poco de los colores que se ven en la habitación.

Otras transformaciones consideradas y descartadas:

- Grayscale: se probó convertir grayscale sin grandes resultados aparentemente porque los modelos pre-entrados usados trabajan con entradas a color.
- RandomRotation: Dado que las habitaciones parten de un piso, rotarlas carece de sentido. Se probó con una pequeña rotación de 10 grados sin grandes cambios.

### 2.4 Etiquetas y desbalanceo del conjunto de entrenamiento

El conjunto de entrenamiento suministrado está desbalanceado, es decir, algunas categorías tienen demasiadas imágenes en comparación a otras.

No modifiqué los datos ni excluí imágenes por esto. Simplemente, calculé los diferentes pesos de las categorías de las siguiente forma para ser usados por la función de pérdida (que va a ser un weighted categorical crossentropy, más información en las siguientes secciones).

## 3 Modelado

### 3.1 Transferencia del aprendizaje - Transfer Learning

En lugar de construir un modelo desde el llano, se optó por reentrenar un modelo existente y entrenado sobre el dataset Imagenet.

Se elige este enforque porque estos modelos se encuentran hiper optimizados y entrenados durante días sobre los datos. Lograr algo similar requeriría mucho tiempo y esfuerzo que carecería de sentido en este contexto.

Igualmente, estos modelos son siempre un buen benchmark para evaluar cualquier modelo que se quiera diseñar desde cero.

Para la competencia, se probaron modelos simples de pocas capas, intermedios y con muchas capas. Se probó con VGG19 con BatchNormalization, Resnet18, AlexNet, Resnet50 y Resnet101. Sin embargo, aquellos que tienen más capas son los que mejores resultados tuvieron: **Resnet152** y **Denset201**.

Adicionalmente, se probó el modelo inception que utiliza entradas de 299x299x3. Los resultados no fueron superiores que los dos detallados anteriormente.

La elección del modelo final tuvo 2 etapas. La primera relacionada con evaluar distintos modelos con diferentes learning rates, optimizadores y parámetors, y una segunda etapa, focalizada en la optimización de los modelos de interés.

De la primer etapa, se obtuvo el learning rate que hace que mejor decaiga la curva de funciones de pérdida (loss functions), optimizadores a utilizar, batch sizes a utilizar,

En un comienzo, no todos los entrenamientos fueron bajo las mismas condiciones, hasta que se encontraron parámetros y modelos interesantes que fueron expuestos a fine tuning que detallo en la próxima sección.

| Network | Top-1 error | Top-5 error |
|---|---|---|
| AlexNet | 43.45 | 20.91 |
| VGG-11 | 30.98 | 11.37 |
| VGG-13 | 30.07 | 10.75 |
| VGG-16 | 28.41 | 9.62 |
| VGG-19 | 27.62 | 9.12 |
| VGG-11 with batch normalization | 29.62 | 10.19 |
| VGG-13 with batch normalization | 28.45 | 9.63 |
| VGG-16 with batch normalization | 26.63 | 8.5 |
| VGG-19 with batch normalization | 25.76 | 8.15 |
| ResNet-18 | 30.24 | 10.92 |
| ResNet-34 | 26.7 | 8.58 |
| ResNet-50 | 23.85 | 7.13 |
| ResNet-101 | 22.63 | 6.44 |
| ResNet-152 | 21.69 | 5.94 |

| Network | Top-1 error | Top-5 error |
|---|---|---|
| SqueezeNet 1.0 | 41.9 | 19.58 |
| SqueezeNet 1.1 | 41.81 | 19.38 |
| Densenet-121 | 25.35 | 7.83 |
| Densenet-169 | 24 | 7 |
| Densenet-201 | 22.8 | 6.43 |
| Densenet-161 | 22.35 | 6.2 |
| Inception v3 | 22.55 | 6.44 |
| GoogleNet | 30.22 | 10.47 |
| ShuffleNet V2 | 30.64 | 11.68 |
| MobileNet V2 | 28.12 | 9.71 |
| ResNeXt-50-32x4d | 22.38 | 6.3 |
| ResNeXt-101-32x8d | 20.69 | 5.47 |

Figure 3: Modelos pre-entrenados - Benchmark - Fuente: Pytorch

Los resultados acompañan la lógica de que modelos más profundos captan mejor los detalles de los datos pero tienden a overfittear y ese fue el principal problema que apareció durante el entrenamiento. Además, la elección de Resnet152 y Densenet201 como candidatos está también justificada por los benchmarks presentados en la figura 4.

Por falta de tiempo, excluyo de la optimización a VGG19 con Batch Normalization e Inception, que me parecen bastante interesantes y que me hubiera gustado profundizar.

### 3.1.1 Bottleneck features vs. entrenamiento del modelo completo

También se consideró usar los modelos como features extractors, es decir, usar los pesos sobre los datos de entrenamiento y entrenar un modelo más pequeño que utilize esta información de entrada. Pero los resultados no fueron tan buenos como reentrenar el modelo completo.

La técnica de reentrenar el modelo consiste en usar los pesos pre-calculados como pesos iniciales y reentrenar el modelo completo, no únicamente las capas finales. Este segundo aproach tuvo resultados superiores.

| | |
|---|---|
| **Estrategias** | Transfer Learning – **Fine Tuning** y Feature Extrator |
| **Modelos** | **VGG19 BN** – Resnet18 – Resnet50 – **Resnet152** – Densenet161 – **Densenet201** – AlexNet – **Inception** |
| **Learning Rates** | 0.01, 0.001, 0.0001, 0.0001, 0.00001, 0.000001, 5e-7, 4e-7 |
| **Epochs** | 1-120 (**mejor rango: 10 a 30 con 1e-6**) |
| **Batch Sizes** | **16** – 32- 64 |
| **Optimizadores** | **Adam** – Adagrad – **SGD** – Adamax – RMSProp |
| **Loss Functions** | Categorical Cross Entropy – **Weighted Categorical Crossentroy** |
| **Otras** | **Weight Decay – Learning Rate Decay** |

Figure 4: Combinaciones y modelos testeados. En negrita, las combinaciones que tuvieron mejores resultados

## 3.2 El modelo

El modelo que mejor funcionó fue un Resnet152. Las redes neuronales residuales proponen una solución al desvanecimiento del gradiente durante el entrenamiento. Esta compuesta por bloques residuales que poseen ademas un enlace que lleva al final del bloque aplicando la función identidad sobre la entrada. Figura 6

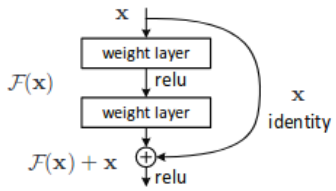

Figure 2. Residual learning: a building block.

Figure 5: Bloque Resnet

La última capa de esta red la he reemplazado por una capa completamente conectada que retorna 16 salidas, cada una correspondiente a la probabilidad de cada categoría.

| layer_name | output_size | 152-layer | |
|---|---|---|---|
| input | | 224x224x3 | |
| conv1 | 112x122 | 7x7, 64, stride 2 | |
| conv2_x | 56x56 | 3x3 max pool, stride 2 | |
| | | 1x1, 64<br>3x3, 64<br>1x1, 256 | X 3 |
| conv3_x | 28x28 | 1x1,128<br>3x3,128<br>1x1,512 | X 8 |
| conv4_x | 14x14 | 1x1, 256<br>3x3, 256<br>1x1, 1024 | X 36 |
| conv5_x | 7x7 | 1x1,512<br>3x3, 512<br>1x1, 2048 | X 3 |
| output | 1x1 | average pull, 16 fc, softmax | |

Figure 6: Resnet-152 modificada . Salida 16 nodos.

## 4 Evaluación de los Resultados