



Inteligencia Artificial Avanzada

Grupo 10

Entrega N°03

INTEGRANTES DEL GRUPO

Apellido y Nombres	E-Mail
Iakantas, Gabriel Maximiliano	giakantas@frba.utn.edu.ar
Anzorandía, Matías Leandro	manzoranda@frba.utn.edu.ar
[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]
Maksymon, Lucas	lmaksymon@frba.utn.edu.ar

Fecha de Presentación	20/10/2023
-----------------------	------------

Muy bien!

El trabajo está bien realizado y documentado.
Pueden continuar con la siguiente entrega.

Reconocedor de Medios de Transporte

Resumen. Alumnos de último año de Ing. en Sistemas de información nos reunimos con el propósito de implementar un modelo de Machine Learning capaz de categorizar distintos medios de transporte a partir de imágenes. En este documento, correspondiente a la tercera parte, se encontrará el detalle de cómo fue llevado a cabo el segundo prototipo.

Introducción

El presente informe, realizado en el marco de la cátedra de Inteligencia Artificial Avanzada, materia de quinto nivel de Ing. en Sistemas de Información de la UTN FRBA, refleja nuestro progreso en la implementación de un modelo de Machine Learning capaz de categorizar medios de transporte a partir de imágenes tomadas desde distintos ángulos.

A tal fin, partiendo de un dataset de acceso público, se confeccionaron dos prototipos con diferentes tecnologías y se contrastaron los resultados, de los cuales uno será seleccionado para seguir avanzando. Como ya comentamos, en esta entrega mostraremos el segundo prototipo.

Secciones correspondientes a esta Entrega

Sobre la Tecnología y la Arquitectura:

Al igual que en la iteración previa realizamos el desarrollo con Python a través de Google Colab. Esta vez reemplazamos el framework Keras por la biblioteca PyTorch y consideramos dos opciones: repetir el modelo CNN o implementar una red ResNet-18 ya pre entrenada. Contrastando los resultados de ambas, nos inclinamos por el segundo. Más adelante en este informe se exponen dichos resultados.

Sobre los datos:

Se utilizó el mismo dataset, con las clases balanceadas y con los mismos porcentajes destinados a entrenamiento, validación y testing. Esta vez aplicamos data augmentation:

```
# Definir transformaciones de aumento de datos para el conjunto de entrenamiento
transformations = transforms.Compose([
    transforms.RandomAffine(degrees=10, translate=(0.05, 0.05), scale=(0.9, 1.1)),
    transforms.RandomHorizontalFlip(p=1)
])
```

Transformaciones en el Dataset Aumentado



Topología

Con respecto a la topología utilizada, repetimos la del modelo anterior para poder comparar, ya que nos pareció interesante poder usar la misma tecnología. Buscamos así eliminar posibles variaciones de performance que se deban al uso de otra tecnología. No obstante, obtuvimos resultados similares para las CNN tanto en PyTorch como en Keras. Como se puede observar, si bien esta información es demasiado técnica para un usuario final, la topología de ResNet-18 es mucho más compleja que la CNN implementada.

- CNN

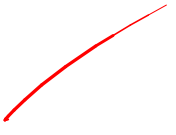
```
CNN(  
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
  (fc1): Linear(in_features=4096, out_features=512, bias=True)  
  (fc2): Linear(in_features=512, out_features=3, bias=True)  
)
```

- ResNet-18


```
ResNetModel(  
  (resnet): 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)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (downsample): Sequential(  
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)  
          (1): 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)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
  )
```

```
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```



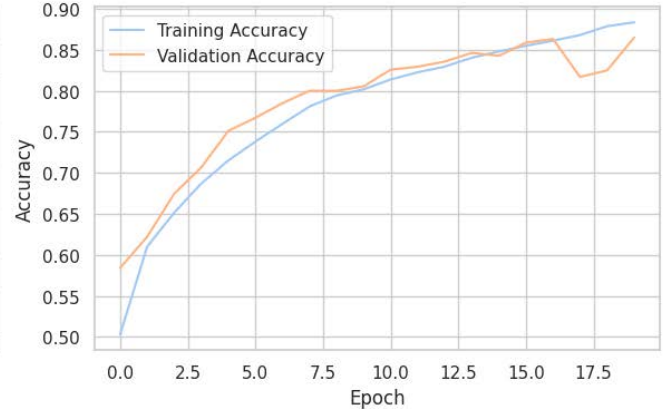
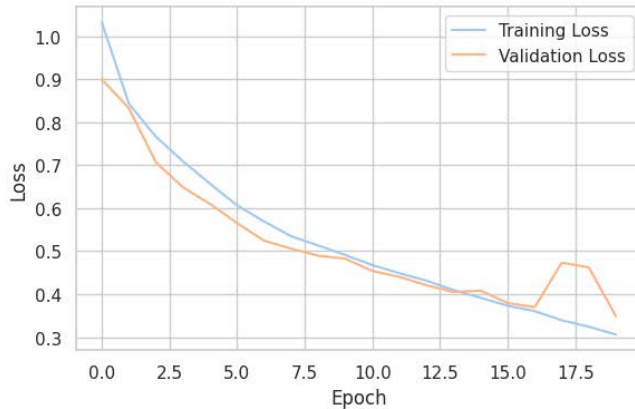
```
(layer4): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, 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=3, bias=True)
)
```



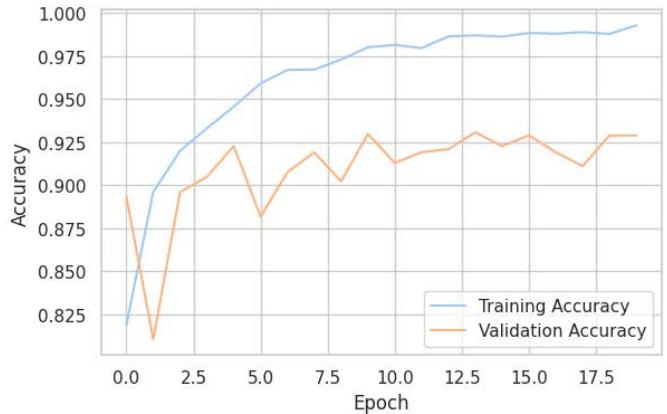
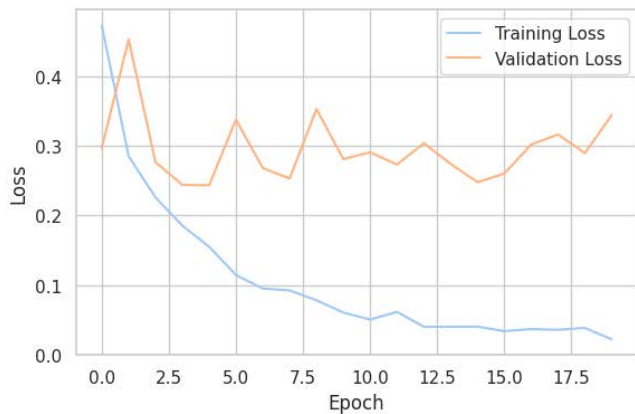
Resultados Obtenidos De la Construcción del prototipo

Gráficos de precisión y error por cantidad de épocas:

- CNN



- Resnet-18



Se puede ver que Resnet obtiene mejores resultados que la CNN, con una loss más estable. En entrenamiento los resultados son mucho mejores, y en validación son mejores.

Sí, es más estable para los datos de entrenamiento, pero para validación no tanto (hay leves variaciones)

Informes de Métricas:

- CNN

```
Finished Training:
Training Accuracy: 0.89, Training Loss: 0.28
Validation Accuracy: 0.87, Validation Loss: 0.35
Validation F1 Score: 0.87, Validation Recall: 0.87
```

- ResNet18:

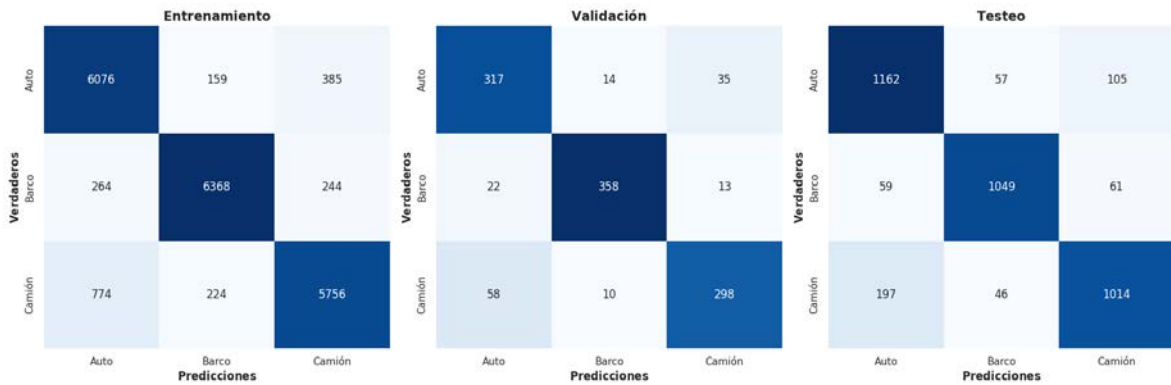
```
Finished Training:
Training Accuracy: 0.97, Training Loss: 0.11
Validation Accuracy: 0.93, Validation Loss: 0.22
Validation F1 Score: 0.93, Validation Recall: 0.93
```

Una cosita, para que tengan en cuenta: las métricas calculadas durante el entrenamiento suelen ser "mentirosas" (el modelo no está del todo estable por estas "abierto")... sirven sólo de referencia. Por eso siempre se recomienda mirar y calcular las métricas luego usando los métodos de ejecución con el modelo ya "cerrado".

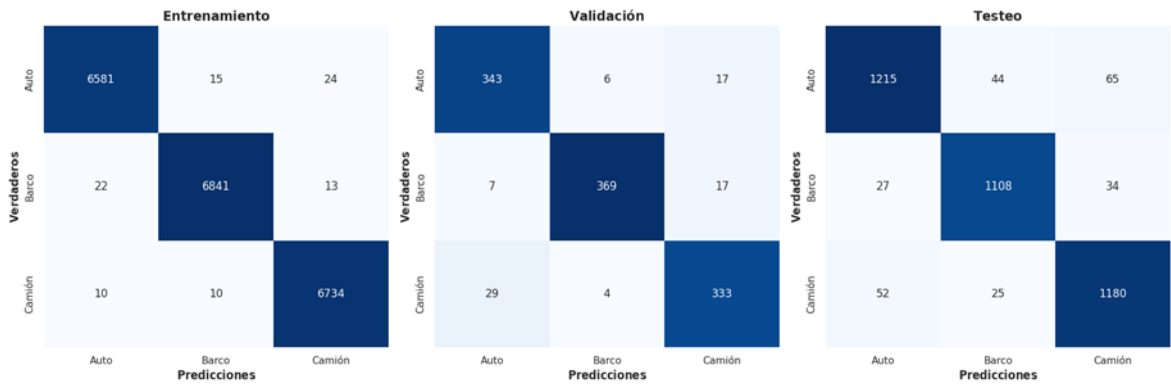
Del testeo del prototipo

Matriz de confusión tanto para entrenamiento, validación y testeo.

- CNN



- ResNet-18 → Se nota que esta red ha aprendido mejor, y que también "generaliza" mejor.



En testeo performa mejor la Res-Net pero quizás no tan bien como cabría esperar. ✓

Tablas de predicciones:

- CNN

Tabla de Predicciones (Entrenamiento):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	6180	440	93.35%
barco	6242	634	90.78%
camion	5798	956	85.85%
Total	18220	2030	89.98%

Tabla de Predicciones (Validación):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	328	38	89.62%
barco	352	41	89.57%
camion	294	72	80.33%
Total	974	151	86.58%

Tabla de Predicciones (Testeo):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	1191	133	89.95%
barco	1045	124	89.39%
camion	1025	232	81.54%
Total	3261	489	86.96%

ResNet:

Tabla de Predicciones (Entrenamiento):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	6575	45	99.32%
barco	6875	1	99.99%
camion	6671	83	98.77%
Total	20121	129	99.36%
Tabla de Predicciones (Validación):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	337	29	92.08%
barco	387	6	98.47%
camion	327	39	89.34%
Total	1051	74	93.42%
Tabla de Predicciones (Testeo):			
Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
auto	1176	148	88.82%
barco	1131	38	96.75%
camion	1143	114	90.93%
Total	3450	300	92.00%

Información quizás bastante técnica pero que sirve para ir a detalle del accuracy. Resnet-18 es bastante mejor que la CNN.

En base a las correcciones de la entrega anterior, mostramos el porcentaje de aciertos en porcentaje para mejorar la legibilidad. **Ok, pero falta el informe de clasificación por clase para mayor detalle (traten de incluirlo en las próximas entregas)**
Algunas de las imágenes clasificadas incorrectamente por ambos modelos:

- CNN



- ResNet



Análisis de Resultados y Comparación de resultados entre prototipos

Con ambos modelos actuales se destacó nuevamente la categoría de Barcos, si bien esta vez fue mucho más parejo.

Para la CNN, tanto en entrenamiento como en validación se obtuvo una mayor pérdida que en el primer prototipo (entrega 2), si bien continúa sin ser una cantidad significativa. En cambio, con la ResNet se logró reducir casi a la mitad, obteniendo mejores resultados en todas las épocas. De forma similar, con la precisión se obtuvo una mejora en el caso de la ResNet mientras que en la CNN decayó levemente.

Dos métricas importantes que miramos al entrenar los modelos son recall y f1-score. El recall mide la capacidad del modelo para encontrar todos los casos positivos. Un alto valor de recall indica que el modelo está identificando la mayoría de los casos relevantes.

El F1-score es una medida que combina la precisión y el recall en una sola puntuación. Representa el equilibrio entre la capacidad de un modelo para encontrar casos positivos (recall) y su capacidad para ser preciso al identificarlos. Es útil cuando queremos una métrica que considere tanto los falsos positivos como los falsos negativos.

Problemas e Inconvenientes encontrados

A modo de comentario, PyTorch resultó ser muy distinto a Keras por lo que fue necesario reescribir el código, aún para generar la misma topología. Fuera de eso, no se encontraron problemas significativos durante la realización.

Sí, es otra serie de librerías que usan una forma de trabajo diferente.

Posibles cursos de acción para mejorar los resultados obtenidos

Lo que podríamos hacer para la entrega final es ver de tomar el modelo de resnet-18 y poder hacerle un mayor ajuste a los hiper parámetros y a la arquitectura para ver si obtenemos mejores resultados.

Ok, me parece bien.

Conclusiones

El modelo pre-entrenado funciona mejor que la CNN, lo que es esperable, dado que tiene una arquitectura más compleja y fue entrenado con muchas más imágenes, gracias a la técnica de data augmentation que permitió mejorar su capacidad de generalización a partir de variaciones de las mismas imágenes.

Finalmente, concluimos que los resultados de esta iteración fueron apropiados a los fines propuestos y que de los modelos obtenidos, uno resultó considerablemente mejor que el anterior, si bien cabe considerar que se aplicaron técnicas que no fueron evaluadas en el primero.

Referencias

Código Fuente. <https://github.com/643riel/IAA>

Dataset. Alex Krizhevsky, Vinod Nair, Geoffrey Hinton (2009). *Cifar-10*
<https://www.cs.toronto.edu/~kriz/cifar.html>