UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Inteligencia Artificial Avanzada

Grupo 10

Entrega N°04

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	13/11/2023

Muy bien!

El trabajo está bien realizado, y documentado.
Que bueno que lograron mejorar los resultados...

Reconocedor de Medios de Transporte

Resumen. Alumnos de último año de Ingeniería 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 cuarta parte, se encontrará el detalle de cómo fue llevada a cabo la implementación del sistema inteligente.

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 tres prototipos con diferentes tecnologías y se contrastaron los resultados, para luego implementar un sistema inteligente partiendo de la base de uno de ellos. Esto último fue volcado en esta entrega.

Secciones correspondientes a esta Entrega

Sobre la Tecnología y la Arquitectura:

Nuevamente realizamos el desarrollo con Python a través de Google Colab. Partimos de la base del primer prototipo, una RNA de tipo Convolucional, volviendo al framework Keras que nos resultó más cómodo para dicha tecnología y nos dio resultados levemente mejores.

Adicionalmente, se incorporó el callback “Early Stopping”, a los fines de prevenir un sobreentrenamiento. Esto consiste en dar por finalizado el entrenamiento en cuanto empieza a aumentar el error de validación en relación con los datos propios del mismo. En distintas corridas del algoritmo la cantidad de épocas alcanzadas fue variando entre 65 y 75, obteniendo resultados semejantes.

Sobre los datos:

Se repitió el dataset de las iteraciones anteriores, con las clases balanceadas y los mismos porcentajes destinados a entrenamiento, validación y testing.

Para normalizar las imágenes, al igual que en las entregas anteriores, se dividieron los valores entre 255, lo que restringe los valores a un rango entre 0 y 1. Este proceso garantiza que todas las imágenes tengan una escala similar y facilita el entrenamiento del modelo.

Además aplicaron Data-Augmentation

A su vez se volvió a utilizar el optimizador SGD (que con esta configuración nos dio mejores resultados que Adam) y la técnica de data augmentation, ambas incorporadas en la segunda versión pero esta vez con las herramientas provistas por el framework Keras:

```python
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator

# Definir las transformaciones de aumento de datos con una semilla
datagen = ImageDataGenerator(
    rotation_range=RANGO_ROTACION,  # Rango de rotación
    width_shift_range=CAMBIO_MAXIMO_ANCHO_IMAGEN,  # Cambio máximo en el ancho de la imagen (10%)
    height_shift_range=CAMBIO_MAXIMO_ALTO_IMAGEN,  # Cambio máximo en la altura de la imagen (10%)
    zoom_range=RANGO_ZOOM,  # Rango de zoom (10%)
    horizontal_flip=VOLTEAR_HORIZONTAL,  # Volteo horizontal
    fill_mode=MODO_RELLENO_TRANSFORMACIONES  # Modo de relleno para las transformaciones
)

# Preparar el generador de datos aumentados
datagen.fit(x_train_final, augment=True, seed=RANDOM_STATE)

# Definir la cantidad de veces que que se duplican los datos
num_augmented_samples = len(x_train_final)

# Generar datos aumentados
augmented_data = []
for x_batch, y_batch in datagen.flow(x_train_final, y_train_final, batch_size=num_augmented_samples, seed=RANDOM_STATE, shuffle=False):
    augmented_data.append((x_batch, y_batch))
    break

# Combinar los datos originales con los datos aumentados
augmented_x_train, augmented_y_train = augmented_data[0]

# Concatenar el dataset aumentado al dataset final de entrenamiento
x_train_final = np.concatenate([x_train_final, augmented_x_train])
y_train_final = np.concatenate([y_train_final, augmented_y_train])

# Verificar las dimensiones del conjunto de datos final después de aumentar
print(f"Número de datos de entrenamiento después de aumentar: {len(x_train_final)}")
```

```
Número de datos de entrenamiento después de aumentar: 22950
```

Algunas transformaciones en el Dataset Aumentado:

Topología

A los fines de mejorar la topología utilizada en prototipos anteriores, se evaluaron distintas combinaciones de capas, resultando la mejor la que se detalla a continuación:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_25 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_12 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 dropout_16 (Dropout)        (None, 16, 16, 32)        0

 conv2d_26 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_27 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_13 (MaxPooli  (None, 8, 8, 64)          0
 ng2D)

 dropout_17 (Dropout)        (None, 8, 8, 64)          0

 conv2d_28 (Conv2D)          (None, 8, 8, 128)         73856

 conv2d_29 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_14 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 dropout_18 (Dropout)        (None, 4, 4, 128)         0

 flatten_4 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 128)               262272

 dropout_19 (Dropout)        (None, 128)               0

 dense_9 (Dense)             (None, 3)                 387

=================================================================
Total params: 549667 (2.10 MB)
Trainable params: 549667 (2.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Ojo, en general no se recomienda usar Drop-Out con capas convolucionales (puede generar que el entrenamiento no sea muy estable)

Está sí se recomienda usar

- El modelo consta de:
  - 3 grupos en serie de capas:
    - Convolución (2): realizan operaciones matemáticas para extraer características relevantes, como bordes o texturas.
    - Max Pooling (1): Reduce dimensiones seleccionando valores máximos en regiones, útil para reconocer objetos trasladados/rotados.
    - Dropout (1): Durante entrenamiento, apaga aleatoriamente neuronas para evitar sobreajuste en la red, haciendo que se vuelva menos dependiente de ciertas conexiones.
  - Capa Flatten (1), que se ocupa de adaptar la matriz de entrada a un vector para la siguiente.

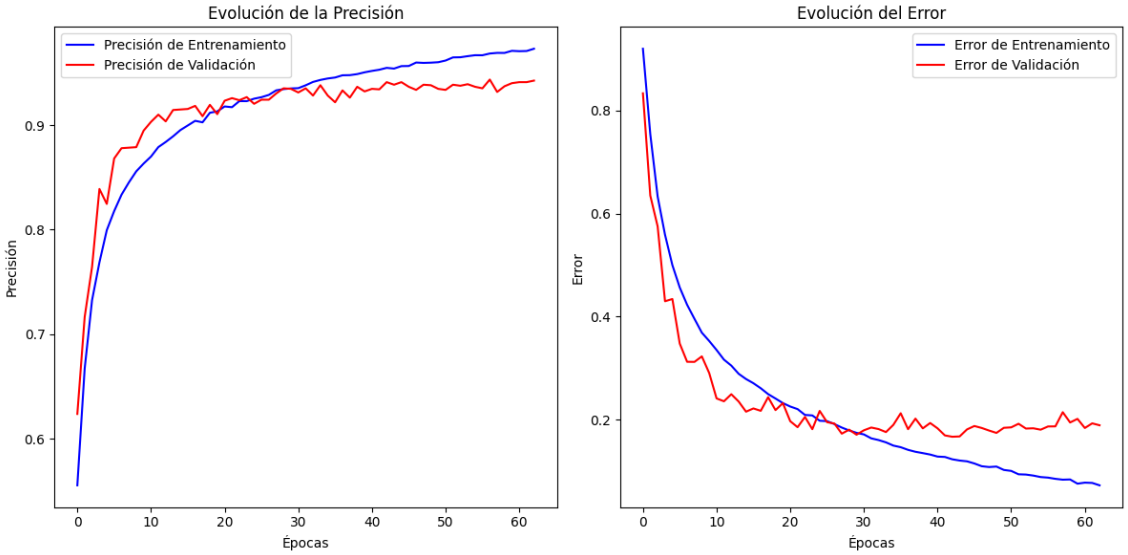
(ver comentario arriba, igual está bien)

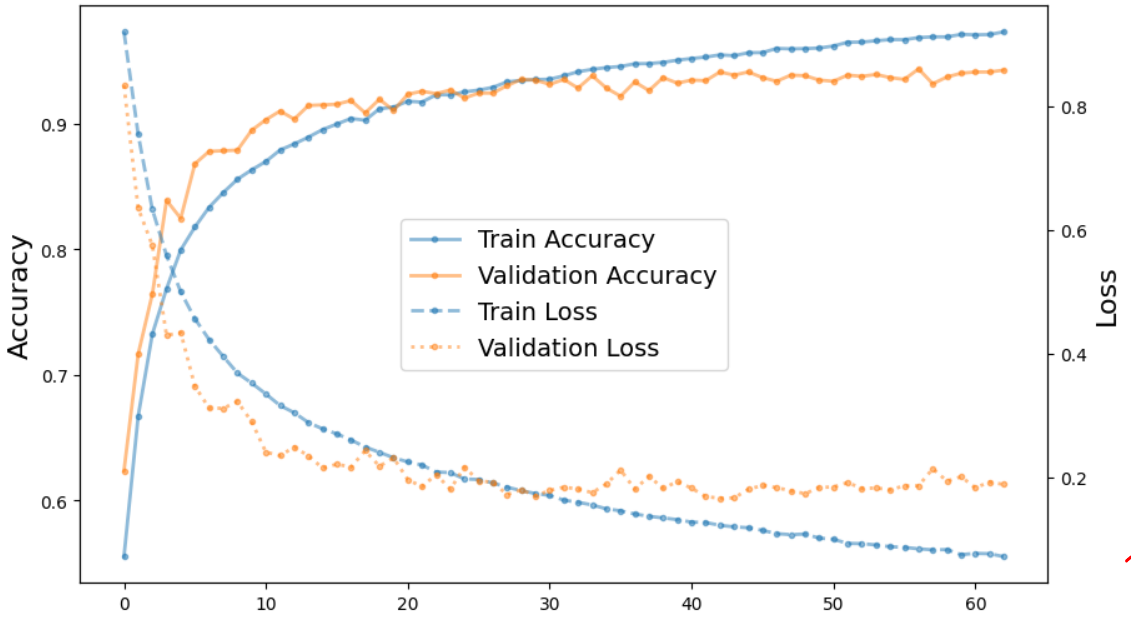
- Capa densa (1), similar a las ocultas de multi perceptrón
- Capa dropout
- Capa de salida (1), con 3 neuronas correspondientes a las 3 clases reconocidas por el modelo.

Resultados Obtenidos

De la Construcción del prototipo

Gráficos de precisión y error por cantidad de épocas:





- En la validación, ambos indicadores se ven algo inestables pero a medida que va ciclando, la oscilación es mucho más atenuada. (ver arriba sobre capas drop-out, igual bien que lo detectaron)
- En el entrenamiento, en cambio, se forman curvas mucho más suaves de principio a fin.
- Las dos precisiones continúan creciendo mucho después que los prototipos anteriores, a la inversa que la pérdida, por lo que esta vez hizo falta colocar un mayor número de épocas. De igual manera, con la cantidad que se había usado anteriormente, ya había alcanzado resultados relativamente buenos.

Informes de Métricas de esta entrega (Keras, CNN)

```
Informe de métricas para el conjunto de entrenamiento:
              precision    recall  f1-score   support

        Auto       0.99      0.98      0.98      7694
       Barco       0.99      0.99      0.99      7602
      Camión       0.97      0.98      0.98      7654

    accuracy                           0.98     22950
   macro avg       0.98      0.98      0.98     22950
weighted avg       0.98      0.98      0.98     22950

Accuracy Total (Entrenamiento): 0.9826

================================================

Informe de métricas para el conjunto de validación:
              precision    recall  f1-score   support

        Auto       0.93      0.93      0.93       662
       Barco       0.97      0.95      0.96       691
      Camión       0.92      0.95      0.93       672

    accuracy                           0.94      2025
   macro avg       0.94      0.94      0.94      2025
weighted avg       0.94      0.94      0.94      2025

Accuracy Total (Validación): 0.9412

================================================

Informe de métricas para el conjunto de prueba:
              precision    recall  f1-score   support

        Auto       0.94      0.94      0.94      1491
       Barco       0.97      0.95      0.96      1508
      Camión       0.92      0.94      0.93      1501

    accuracy                           0.94      4500
   macro avg       0.94      0.94      0.94      4500
weighted avg       0.94      0.94      0.94      4500

Accuracy Total (Prueba): 0.9429
```

Resumiendo:

	Entrenamiento			Validación			Testeo		
	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall
Autos	98%	99%	98%	93%	93%	93%	94%	94%	94%
Barcos	99%	99%	99%	96%	97%	95%	96%	97%	95%
Camiones	98%	97%	98%	93%	92%	95%	93%	92%	94%

Igual, aprendió bien, generaliza muy bien...

y se ve que mejora mucho con respecto a los anteriores.

Se pueden ver mucho mejores resultados que en la entrega 2 (también Keras, CNN):

```
Informe de métricas para el conjunto de entrenamiento:
              precision    recall  f1-score   support

        Auto       0.94      0.93      0.93      3847
       Barco       0.94      0.97      0.95      3801
      Camión       0.93      0.91      0.92      3827

    accuracy                           0.93     11475
   macro avg       0.93      0.93      0.93     11475
weighted avg       0.93      0.93      0.93     11475

Accuracy Total (Entrenamiento): 0.9344

================================================

Informe de métricas para el conjunto de validación:
              precision    recall  f1-score   support

        Auto       0.88      0.86      0.87       662
       Barco       0.91      0.93      0.92       691
      Camión       0.88      0.87      0.87       672

    accuracy                           0.89      2025
   macro avg       0.89      0.89      0.89      2025
weighted avg       0.89      0.89      0.89      2025

Accuracy Total (Validación): 0.8899

================================================

Informe de métricas para el conjunto de prueba:
              precision    recall  f1-score   support

        Auto       0.86      0.87      0.86      1491
       Barco       0.89      0.93      0.91      1508
      Camión       0.87      0.82      0.85      1501

    accuracy                           0.87      4500
   macro avg       0.87      0.87      0.87      4500
weighted avg       0.87      0.87      0.87      4500

Accuracy Total (Prueba): 0.8731
```

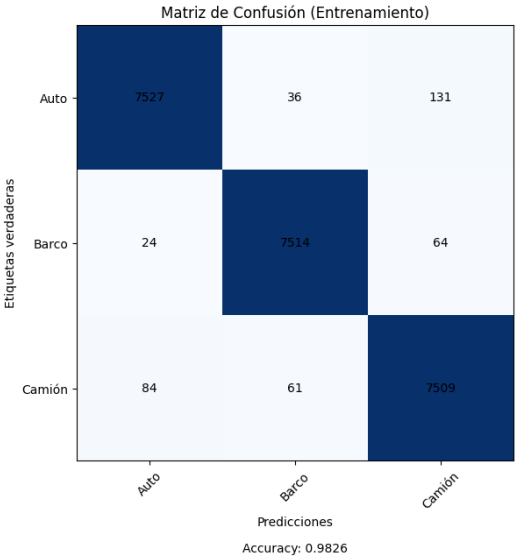
Y mejores resultados que en la entrega 3, la Resnet-18 (que calculamos distinto porque usamos PyTorch)

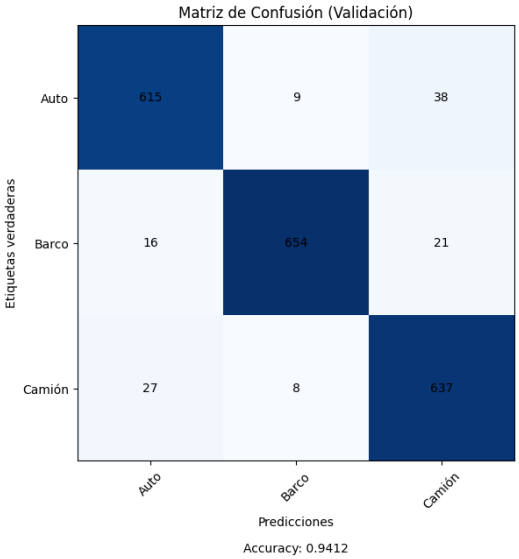
```
Finished Training:
Training Accuracy: 0.97, Training Loss: 0.11
Validation Accuracy: 0.93, Validation Loss: 0.22
Validation F1 Score: 0.93, Validation Recall: 0.93
```

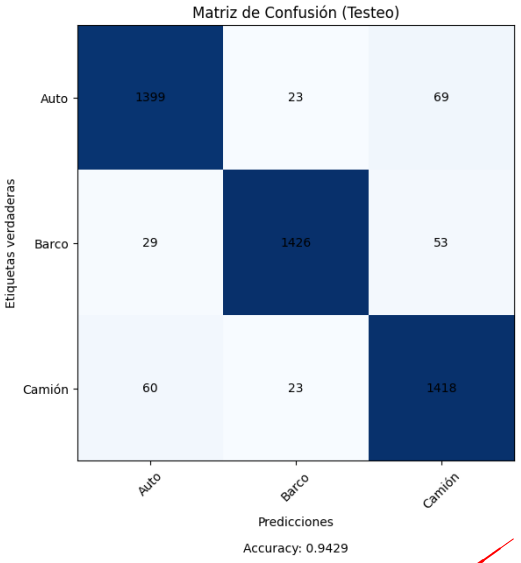
Se puede ver que en esta entrega, se mejoraron significativamente todos los valores.

Del testeo del prototipo

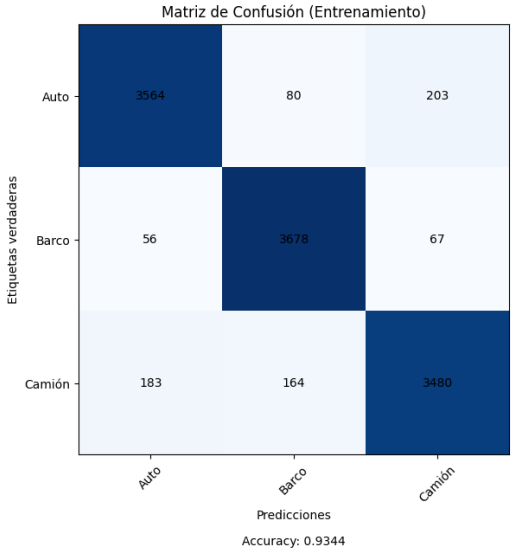
Matriz de confusión tanto para entrenamiento, validación y testeo. Keras, CNN

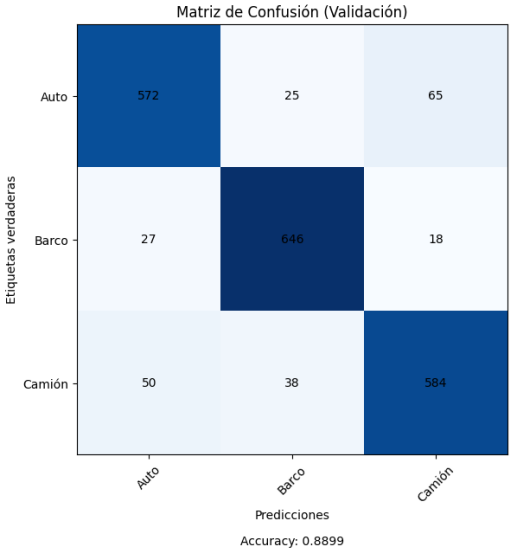


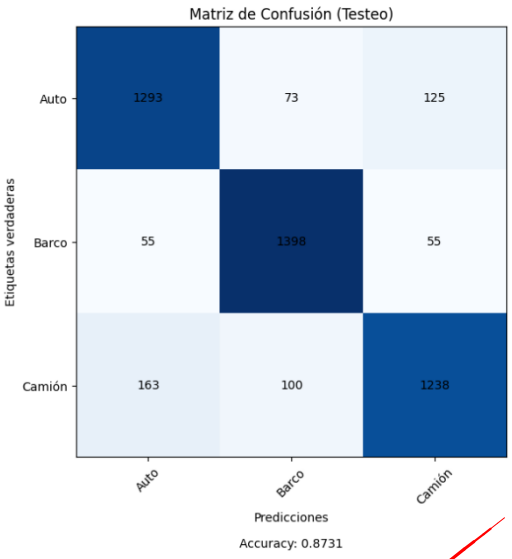




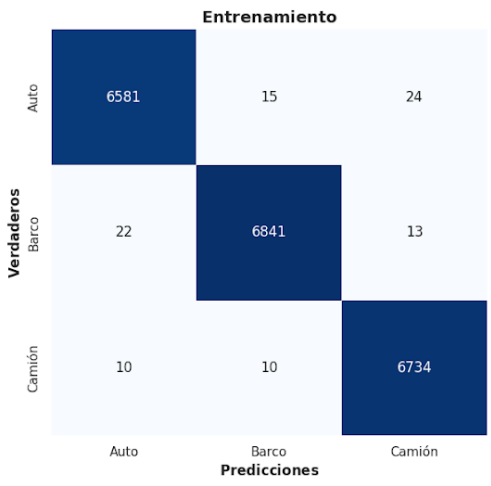
Entrega 2 (Keras, CNN)

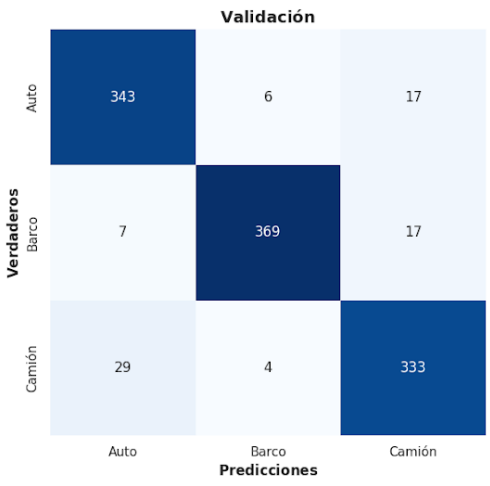


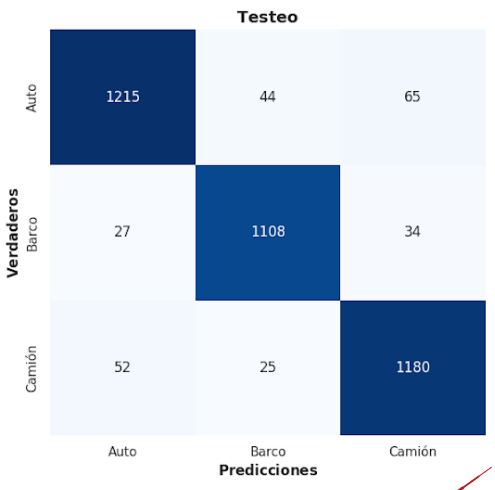




Entrega 3 (Pytorch Resnet-18)







Tablas de predicciones por clase (entendemos que no tiene mucho sentido poner las de todas las entregas):

Info del dataset:

	Cantidad en Entrenamiento	Cantidad en Validación	Cantidad en Testeo	Total
Auto	7694	662	1491	[9847]
Barco	7602	691	1508	[9801]
Camión	7654	672	1501	[9827]
Total	22950	2025	4500	29475

Tabla de Predicciones (Entrenamiento):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	7527	167	97.83%
Barco	7514	88	98.84%
Camión	7509	145	98.11%
Total	22550	400	98.26%

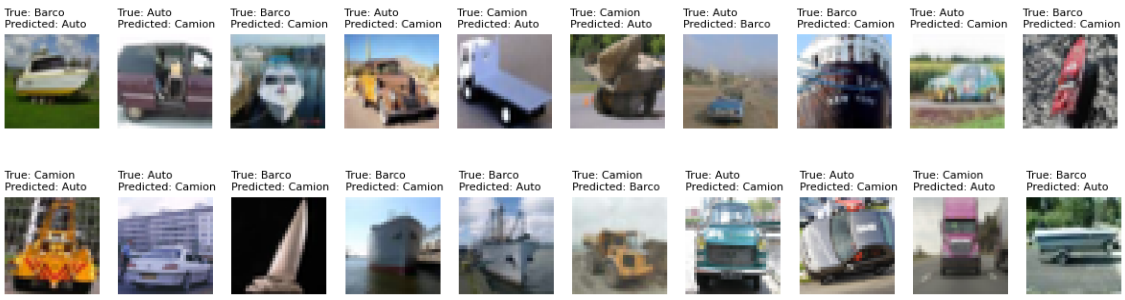
Tabla de Predicciones (Validación):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	615	47	92.90%
Barco	654	37	94.65%
Camión	637	35	94.79%
Total	1906	119	94.12%

Tabla de Predicciones (Testeo):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	1399	92	93.83%
Barco	1426	82	94.56%
Camión	1418	83	94.47%
Total	4243	257	94.29%

Algunas de las imágenes clasificadas incorrectamente por el modelo:



Análisis de Resultados y Comparación de resultados entre prototipos

Las 3 clases performan de manera similar en entrenamiento, aunque la de autos obtuvo un desempeño ligeramente más bajo. En validación y testing, la de todas es algo menor al entrenamiento, lo que es esperable. La de los autos vuelve a ser ligeramente menor a las otras.

¿tal vez sea porque los autos tienen formas y colores más diversos?

En relación con los modelos anteriores, se observan incrementos considerables en las métricas, y en la clasificación de las imágenes de las 3 clases.

Problemas e Inconvenientes encontrados

Sí, ¿se basaron en algún paper / artículo / tutorial, o fue inspiración de ustedes? Cuentenlo en la presentación por favor.

Fue un desafío encontrar una topología de CNN que pudiera mejorar a la entrega anterior, la Resnet-18. Teniendo de referencia ese modelo, armamos algo más complejo que lo que teníamos en la primera entrega. Esto fue lo principal para obtener mejores resultados, así como la optimización de los distintos parámetros. Volvimos a aplicar data augmentation como en la entrega anterior.

Conclusiones

Con cerca de 95 de cada 100 datos clasificados correctamente por el modelo en cada clase de testeo, podemos decir que el mismo satisface los objetivos propuestos, superando ampliamente los demás prototipos. Dicho esto, concluimos que se logró a través del mismo resolver el problema planteado inicialmente, pero debemos tener en cuenta que el dataset es de 2009, por lo que no podemos garantizar que se adapte a vehículos actuales. Sin embargo, no se observan cambios demasiado pronunciados en la forma de estos medios de transporte que dieran lugar a confusión por lo que suponemos que obtendría un buen desempeño.

(en general se puede asumir que sí)

En futuras investigaciones para mejorar el modelo, sugerimos considerar como posibles cursos de acción, incrementar el dataset, incluir datos más actualizados y evaluar otros optimizadores modificando los parámetros.

En cuanto a las lecciones aprendidas, podemos dar cuenta de la importancia del trabajo previo con los datos, por ejemplo, viendo la mejora de performance con data augmentation. Lo mismo se puede decir de la normalización, y del balanceo. Por otro lado, destacar la necesidad de adecuar la topología al problema, logrando encontrar un término medio entre una demasiado simple, que fuera incapaz de aprender bien los datos lo cual se manifiesta en una mala performance, y una demasiado compleja, que tarde mucho en entrenar y sufra sobreajuste.

Finalmente, podemos comentar que las CNN son una herramienta muy poderosa a la hora de pensar este tipo de problemas, complejos de abordar de forma eficiente con tecnologías tradicionales.

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