UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Inteligencia Artificial Avanzada

Grupo 10

Entrega N°02

INTEGRANTES DEL GRUPO

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

Fecha de Presentación	23/10/2023

Muy Bien!

El trabajo está bien realizado y documentado.
Los resultados son aceptables para el primer prototipo pero igual se podrían mejorar...

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, ud. encontrará el detalle de cómo fue llevado a cabo el primer prototipo.

1. 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 confeccionarán dos prototipos con diferentes tecnologías y se contrastarán los resultados, para luego seleccionar una u otra para seguir avanzando. Como ya comentamos, en esta entrega mostraremos el primer prototipo.

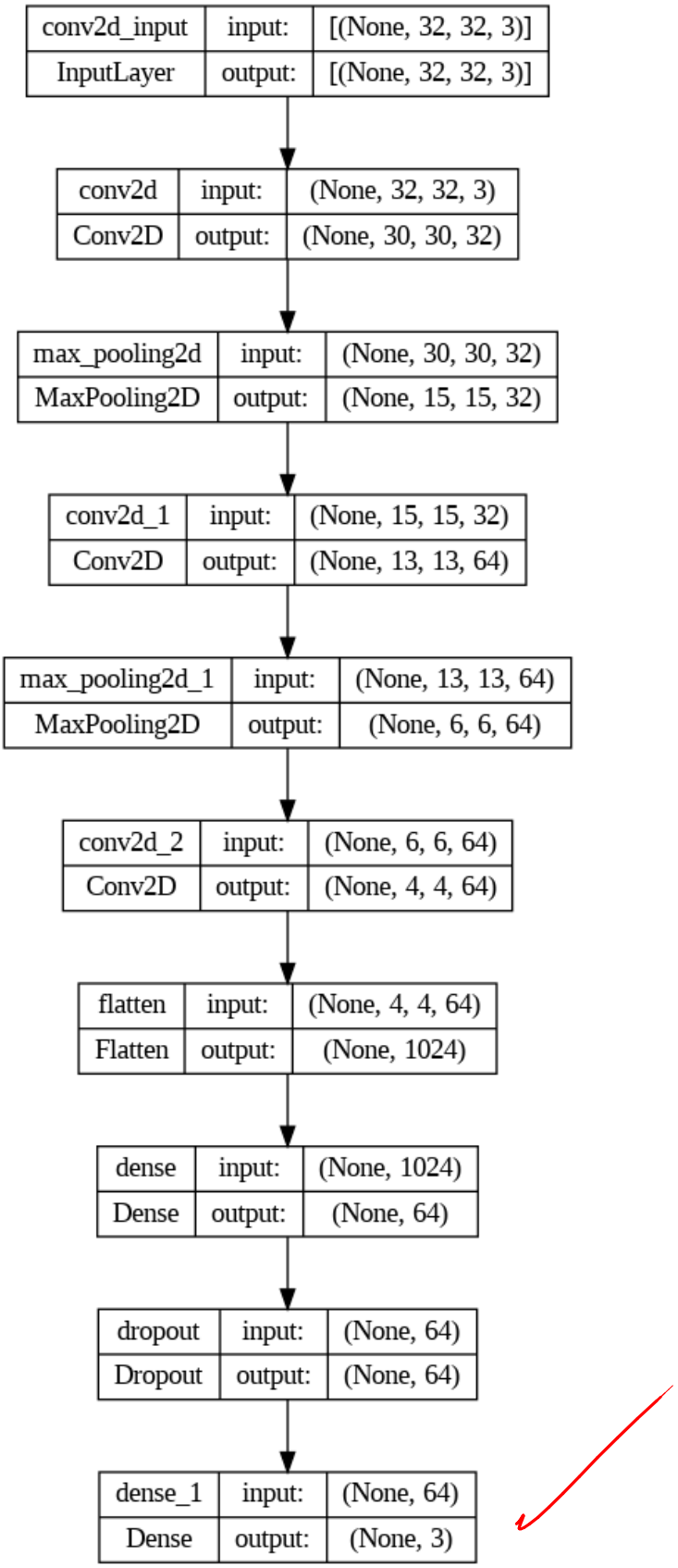
2. Secciones correspondientes a esta Entrega

1. Sobre la Tecnología, Arquitectura y Topología:

En cuanto a tecnología, se decidió ir por una Red Neuronal Artificial, con una arquitectura Convolucional (CNN), modelo que se destaca por ser especialmente efectivo para problemas de clasificación de imágenes.

La topología seleccionada fue la siguiente:

```
model.summary()

Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_102 (Conv2D)         (None, 30, 30, 32)        896

 max_pooling2d_68 (MaxPooli  (None, 15, 15, 32)        0
 ng2D)

 conv2d_103 (Conv2D)         (None, 13, 13, 64)        18496

 max_pooling2d_69 (MaxPooli  (None, 6, 6, 64)          0
 ng2D)

 conv2d_104 (Conv2D)         (None, 4, 4, 64)          36928

 flatten_34 (Flatten)        (None, 1024)              0

 dense_68 (Dense)            (None, 64)                65600

 dropout_34 (Dropout)        (None, 64)                0

 dense_69 (Dense)            (None, 3)                 195

=================================================================
Total params: 122115 (477.01 KB)
Trainable params: 122115 (477.01 KB)
Non-trainable params: 0 (0.00 Byte)
```


conv2d_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 30, 30, 32)
max_pooling2d | input: | (None, 30, 30, 32)
MaxPooling2D | output: | (None, 15, 15, 32)
conv2d_1 | input: | (None, 15, 15, 32)
Conv2D | output: | (None, 13, 13, 64)
max_pooling2d_1 | input: | (None, 13, 13, 64)
MaxPooling2D | output: | (None, 6, 6, 64)
conv2d_2 | input: | (None, 6, 6, 64)
Conv2D | output: | (None, 4, 4, 64)
flatten | input: | (None, 4, 4, 64)
Flatten | output: | (None, 1024)
dense | input: | (None, 1024)
Dense | output: | (None, 64)
dropout | input: | (None, 64)
Dropout | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 3)

```python
# Construir el CNN. Se agrega capa de dropout
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))  # dropout rate de 0.5 en caso de que haya mucho overfitting
model.add(layers.Dense(3, activation='softmax'))  # 3 classes: Auto, Camión, Barco
```

A modo de comentario de cada una de las capas:

- Capa Conv2D (Convolucional): Esta capa realiza operaciones de convolución en la imagen de entrada para detectar patrones y características.
- Capa MaxPooling2D: La operación de agrupación máxima reduce el tamaño de la representación y reduce la cantidad de parámetros de la red.
- Capa Flatten: Esta capa transforma la salida de las capas convolucionales en un vector unidimensional que puede ser alimentado a las capas densas.
- Capa Dense: Estas capas completamente conectadas realizan cálculos basados en la salida de las capas anteriores.
- Capa Dropout: Esta capa ayuda a reducir el overfitting al apagar aleatoriamente una fracción de las unidades durante el entrenamiento.

2. Sobre las Herramientas, Frameworks y Bibliotecas

El modelo fue implementado en Python utilizando Google colab. Se hizo uso de las bibliotecas *TensorFlow* y *Keras*.

El modelo se compila utilizando el optimizador 'adam' y la función de pérdida *'sparse_categorical_crossentropy'*, y se entrena durante un número específico de épocas.

```python
# Compilar el modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

3. Sobre los Datos

En la implementación del prototipo se utilizó un dataset de CIFAR 10, un repositorio público, del cual se tomaron imágenes de autos, barcos y camiones:

```
Info del dataset:
+--------+---------------------------+-----------------------+--------------------+--------+
|        | Cantidad en Entrenamiento | Cantidad en Validación | Cantidad en Testeo | Total  |
+--------+---------------------------+-----------------------+--------------------+--------+
|  Auto  |           3847            |          662          |        1491        | [6000] |
| Barco  |           3801            |          691          |        1508        | [6000] |
| Camión |           3827            |          672          |        1501        | [6000] |
| Total  |           11475           |          2025         |        4500        | 18000  |
+--------+---------------------------+-----------------------+--------------------+--------+
```

OK. Las clases están balanceadas tanto para Entrenar, Validar y Probar.

Los mismos se leen del sitio y se seleccionan los que correspondan a las clases mencionadas. Se tomaron 60% de datos para entrenamiento, 25% para test y el 15% restante para la validación.

```
# 25% para test
x_train_final, x_test_final, y_train_final, y_test_final = train_test_split(
    x_combined, y_combined, test_size=0.25, random_state=42)

# Variables de validación vacías al principio
x_validation_final, y_validation_final = [], []

if include_validation:
    # 15% para validación
    x_train_final, x_validation_final, y_train_final, y_validation_final = train_test_split(
        x_train_final, y_train_final, test_size=0.15, random_state=42)
```

```
# Compilar el modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

early = keras.callbacks.EarlyStopping(patience=8, monitor='val_loss',
                                      restore_best_weights=True)

# Entrenamiento del modelo. Guardo en fiteo porque después voy a usar eso para graficar
if include_validation:
    fiteo = model.fit(x_train_final, y_train_final, epochs=epocas,
                      validation_data=(x_validation_final, y_validation_final), callbacks=[early,])
else:
    fiteo = model.fit(x_train_final, y_train_final, epochs=epocas, callbacks=[early,])
```

Las imágenes se procesaron en formato rgb y como método de normalización, se dividieron los componentes de cada píxel por 256, de modo que el dato que llegue a la red sea el porcentaje de intensidad de cada color.

Se usó cross validation para determinar el número óptimo de épocas, esto para no sobreajustar y que el modelo pueda responder bien ante datos que nunca vio.

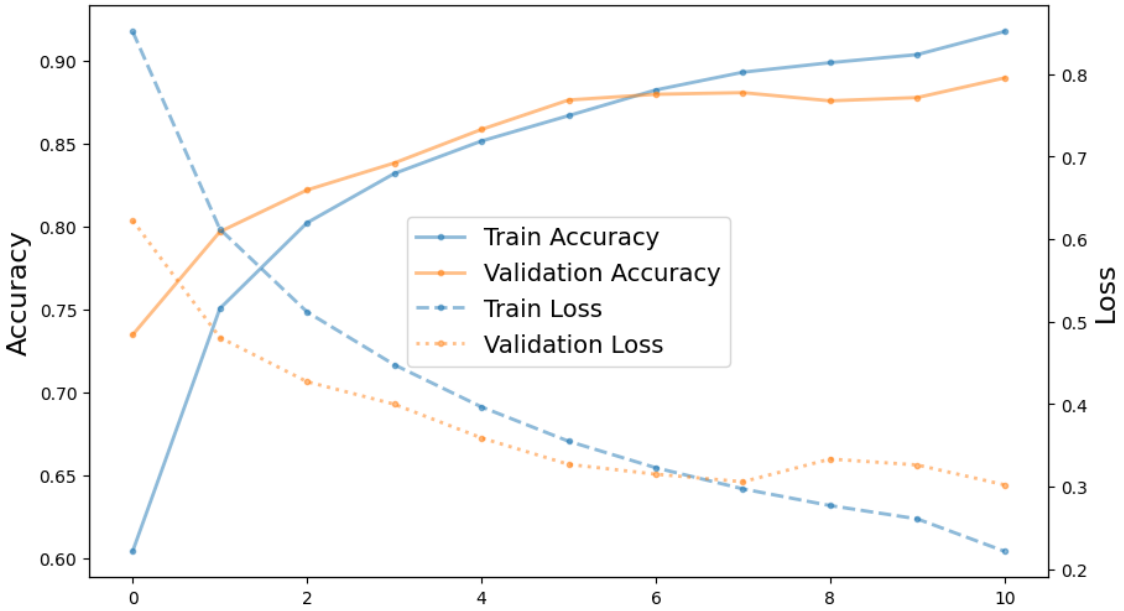
```
epochs_to_test = [8, 9, 10, 11, 12, 13]
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
Epochs: 8, Accuracy: 0.8672000050544739, Loss: 0.3551080048084259
Epochs: 9, Accuracy: 0.8619555592536926, Loss: 0.3696424841880798
Epochs: 10, Accuracy: 0.8669333338737488, Loss: 0.36349326372146606
Epochs: 11, Accuracy: 0.8740444540977478, Loss: 0.35961806774139404
Epochs: 12, Accuracy: 0.869777774810791, Loss: 0.3754478931427002
Epochs: 13, Accuracy: 0.8784000158309937, Loss: 0.3615347683429718
```

4. Resultados Obtenidos

i. De la Construcción del prototipo

Gráficos de precisión y error por cantidad de épocas:



ii. Del testeo del prototipo

Algunas de las imágenes clasificadas incorrectamente por el modelo:

Nota:
Tenga en cuenta que si esto se debería presentar a una gerencia, sería necesario formatear la tabla (por ejemplo usando porcentajes en lugar de valores decimales)

```
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
```

Métricas más que aceptables

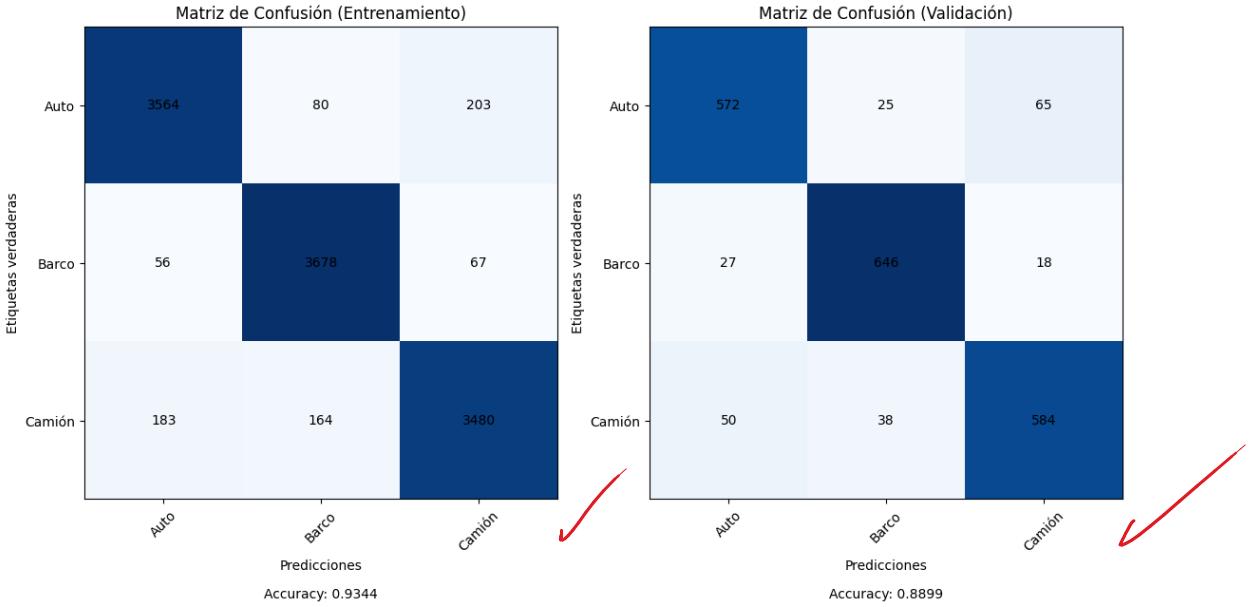
Métricas aceptables pero se podrían mejorar

Métricas aceptables pero se podrían mejorar

La precisión mide la proporción de predicciones positivas que son verdaderamente positivas en un problema de clasificación, y se calcula como Verdaderos Positivos dividido por la suma de Verdaderos Positivos y Falsos Positivos.

Recall (Recuperación o Sensibilidad): El recall mide la proporción de ejemplos positivos que fueron correctamente identificados por el modelo en un problema de clasificación, y se calcula como Verdaderos Positivos dividido por la suma de Verdaderos Positivos y Falsos Negativos.

Matriz de confusión tanto para entrenamiento, validación y testeo. Acá se nota que las precisiones correctas quedan en la diagonal.



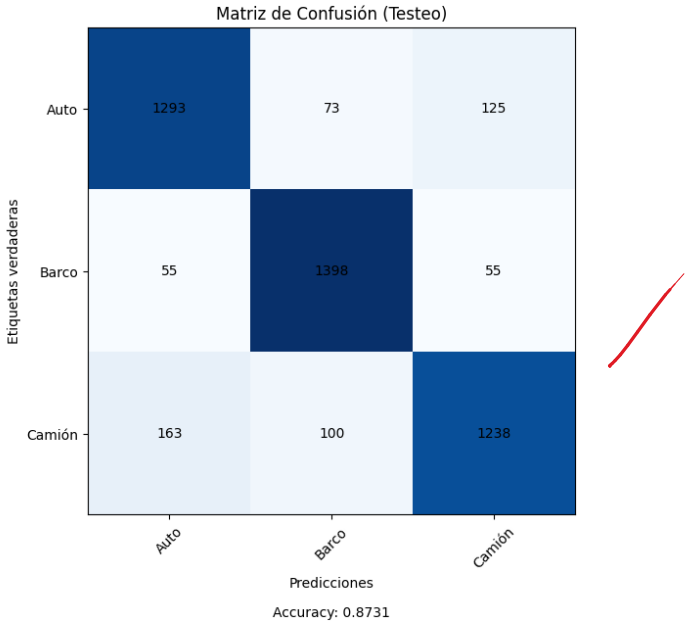


Tabla con información interesante sobre la cantidad de imágenes del dataset y el accuracy por clase:

Info del dataset:

	Cantidad en Entrenamiento	Cantidad en Validación	Cantidad en Testeo	Total
Auto	3847	662	1491	[6000]
Barco	3801	691	1508	[6000]
Camión	3827	672	1501	[6000]
Total	11475	2025	4500	18000

Tabla de Predicciones (Entrenamiento):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	3564	283	92.64%
Barco	3678	123	96.76%
Camión	3480	347	90.93%
Total	10722	753	93.44%

Tabla de Predicciones (Validación):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	572	90	86.40%
Barco	646	45	93.49%
Camión	584	88	86.90%
Total	1802	223	88.99%

Tabla de Predicciones (Testeo):

Clase	Predicciones Correctas	Predicciones Incorrectas	Porcentaje de Aciertos
Auto	1293	198	86.72%
Barco	1398	110	92.71%
Camión	1238	263	82.48%
Total	3929	571	87.31%

5. Análisis de Métricas

La categoría con mejores resultados fueron los Barcos. Esperábamos que esto sucediera debido al fondo, siendo que estos se encontrarán mayormente en el agua. Sin embargo, la diferencia con las otras no es demasiado significativa. En general podemos decir que se obtuvieron resultados satisfactorios.

En muchos casos donde el modelo confunde vehículos terrestres con barcos, notamos que suele darse que el suelo quede fuera de la imagen o bien aparece un fondo color azul, lo cual lo hace confundirse con agua.

(sí, puede ser ¿son muchos casos?)

El modelo tiene buenos resultados tanto en accuracy como en loss:

La precisión es una métrica que se utiliza para evaluar el rendimiento de un modelo de clasificación. Se calcula como la proporción de predicciones correctas (verdaderos positivos y verdaderos negativos) en relación con el número total de predicciones realizadas.

La pérdida, también conocida como función de pérdida o función de costo, es una medida que evalúa cuán erróneas son las predicciones de un modelo en comparación con las etiquetas reales en un problema de aprendizaje supervisado. En este caso, usamos "Sparse Categorical Cross-Entropy" (entropía cruzada categórica dispersa) que es una función de pérdida comúnmente utilizada en problemas de clasificación multiclase cuando las etiquetas de clase se representan de forma dispersa o en formato entero (en lugar de codificación one-hot).

6. Problemas e Inconvenientes encontrados

i. En un principio por error utilizamos los mismos datos para prueba que para validación, lo cual se reflejaba en muy buenos resultados pero no concluyentes. Esto se corrigió y se logró obtener buenos resultados.

7. Posibles cursos de acción para mejorar los resultados obtenidos

Para la siguiente entrega deberían cambiar de tecnología, entonces les recomendaría alguna de estas alternativas:
- usar AutoKeras para imágenes
- usar algún modelo pre-Entrenado con Transfer Learning
- usar algún miniTransformer

i. Aumentar la complejidad del modelo mediante la adición de capas ocultas y/o aumentando el número de neuronas en cada capa. Esto permite que el modelo capture características más complejas en los datos.
ii. Usar arquitecturas más avanzadas, como redes neuronales recurrentes (RNN) o modelos preentrenados, para encontrar un mejor modelo. Ojo que las RNN no son adecuadas para procesar imágenes!
iii. Utilizar técnicas de regularización, como L1 o L2, que pueden ayudar a evitar el sobreajuste al penalizar los pesos de las capas de la red.

3. Conclusiones

Destacamos la importancia de haber trabajado con un set con un gran número de datos (6000 imágenes para cada clase) que además ya esté bien armado, bastando con normalizar las imágenes para poder trabajar con él.

De lo expuesto anteriormente, concluimos que obtuvimos resultados satisfactorios a los fines de este estudio, que el modelo es un buen candidato para ser escogido para el producto final y consideramos viable a futuro seguir iterando sobre esta base, siguiendo los cursos de acción propuestos.

4. Referencias

1. Código Fuente. https://github.com/643riel/IAA
2. Alex Krizhevsky, Vinod Nair, Geoffrey Hinton (2009). *Cifar-10* https://www.cs.toronto.edu/~kriz/cifar.html