| **Práctica 1: Clasificación de imágenes. Arquitecturas CNN. *Transfer Learning*** |
| --- |

Esta práctica se implementa en Google Colab, un entorno que permite desarrollar código Python en Python Notebooks y utilizar aceleración GPU para entrenar modelos de Aprendizaje Profundo. Para usarlo, tendrá que crear una cuenta personal de Google. Para trabajar en Google Colab, siga estas instrucciones:

* Cree una carpeta para las prácticas del curso en su Google Drive personal.
* Abra los enlaces de Colab incluidos en este guion
* Guarde una copia de los notebooks en su carpeta de Google Drive
* Cada notebook de Colab (es decir, los archivos que terminan en .ipynb) corresponde a una parte de la práctica. En Google Drive, haga doble clic en el notebook y seleccione Abrir con Colab.
* Una vez que haya completado la práctica (es decir, ha llegado al final del notebook), puede guardar el archivo editado y pasar al siguiente bloc de notas.
* Asegúrese de guardar periódicamente el notebook (Guardar archivo), para no perder el progreso si la máquina virtual de Colab se desconecta.🡪

El objetivo de esta práctica es presentar al estudiante el problema de clasificación de imágenes, los conceptos básicos de varias arquitecturas CNN, el manejo de datasets de imágenes con PyTorch, y la utilización de estrategias de *Transfer Learning*. Para completar esta práctica, deberá leer la documentación de Pytorch. Puede encontrarla [aquí.](https://pytorch.org/docs/stable/index.html)

Debe completar el código de los notebooks de Colab, y completar un informe de práctica con las respuestas a las preguntas que se incluyen en las secciones siguientes. Una vez haya terminado, debe subir a Moodle una copia de los notebooks completados (archivos .ipynb) y el informe de práctica, combinados en un solo archivo zip.  **IMPORTANTE: Tanto el archivo del informe como el archivo zip deben tener el siguiente nombre: 'APELLIDO(s)\_NOMBRE'**

Puedes encontrar [aquí el primer notebook.](https://drive.google.com/file/d/1k_2-KUN3rR93jq4aKeDNJ96yvaLsMA3I/view?usp=sharing) Debe completar el código en el notebook y responder a las preguntas de la Sección 1.1.

## Simple CNN

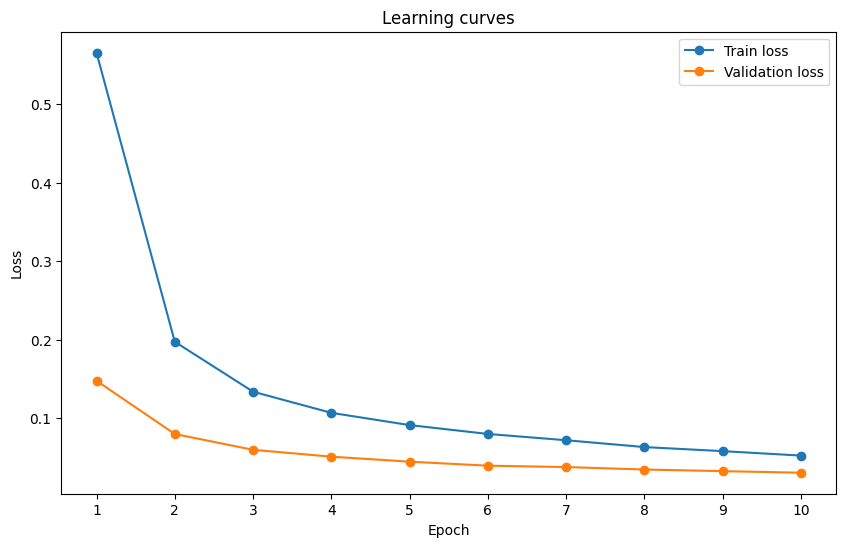
* Tamaños de los conjuntos de entrenamiento y validación descargados del *dataset* MNIST

|  | Alto de imagen | Ancho de imagen | N.º canales de imagen | N.º muestras |
| --- | --- | --- | --- | --- |
| Entrenamiento | 28 | 28 | 1 | 60.000 |
| Validación | 28 | 28 | 1 | 10.000 |

* Número de parámetros del modelo Simple CNN

|  | N.º parámetros entrenables |
| --- | --- |
| Simple CNN | 813.802 |

* Incluya las curvas de entrenamiento y validación para 10 épocas. Indique también la mejor precisión obtenida, y en qué época se logra este resultado

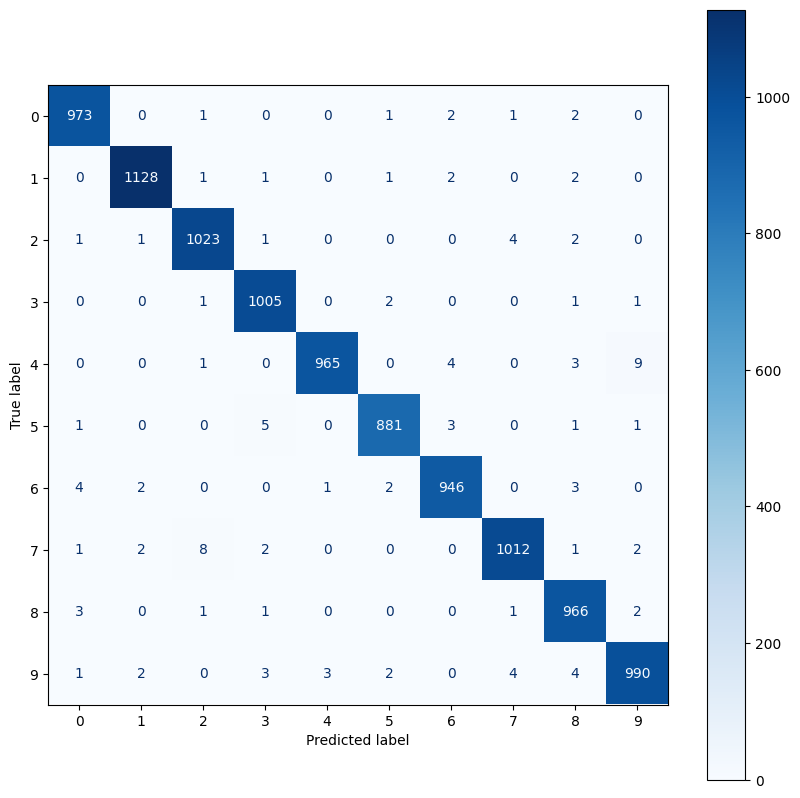


|  | Mejor precisión (validación) | Época con mejor precisión |
| --- | --- | --- |
| Simple CNN | 98.89% | 10 |

Comentar las conclusiones sobre la evolución de la *loss* de entrenamiento y validación, con respecto a posibles problemas de sesgo (*high-bias*) o sobreajuste (*overfitting*). Indique si considera que continuar con más épocas de entrenamiento mejoraría el rendimiento del modelo

A partir de esta gráfica no se observan ningún problema de sesgo ni de overfitting. Esto es debido a que el valor de loss se va reduciendo constantemente tanto en validación como en entrenamiento. Esto puede ser consecuente al uso de métodos de regularización como el uso de capas Dropout y a la casi perfecta adaptación del modelo (parametros) al problema a resolver. Al no observar un punto crítico donde realizar EarlyStopping, no se puede concluir que no vaya a mejorar más, pero en validación tiene un valor casi perfecto igual a 98,89%. Por lo que podría seguir mejorando, pero no sería una mejora tan significativa.

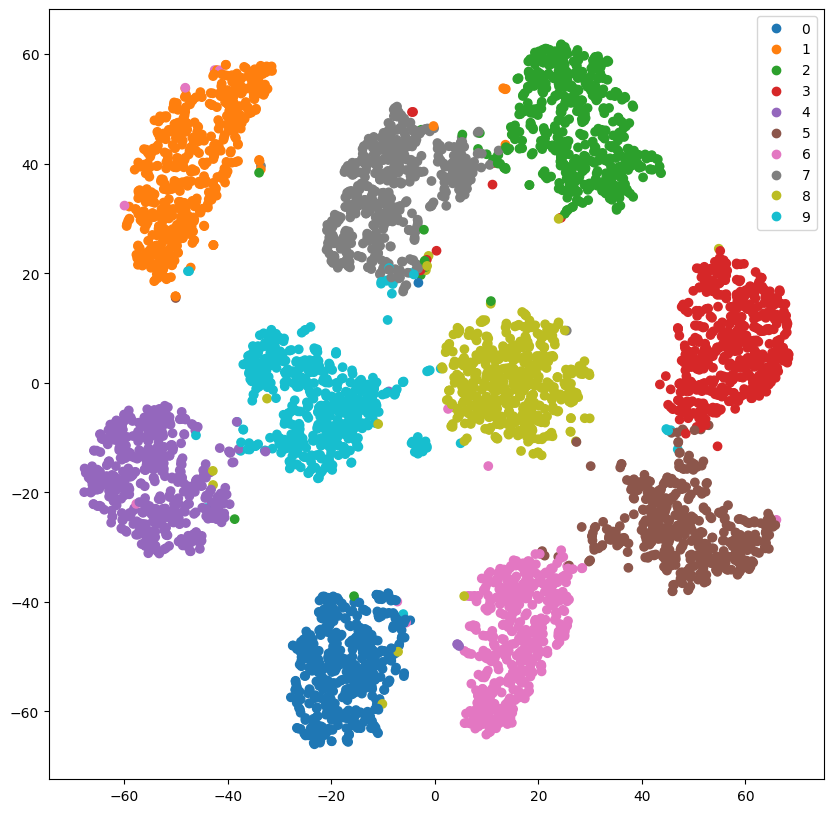
* Incluir la matriz de confusión obtenida. Dada esta matriz de confusión, informe de los 2 casos de confusión entre clases que ocurren con más frecuencia.



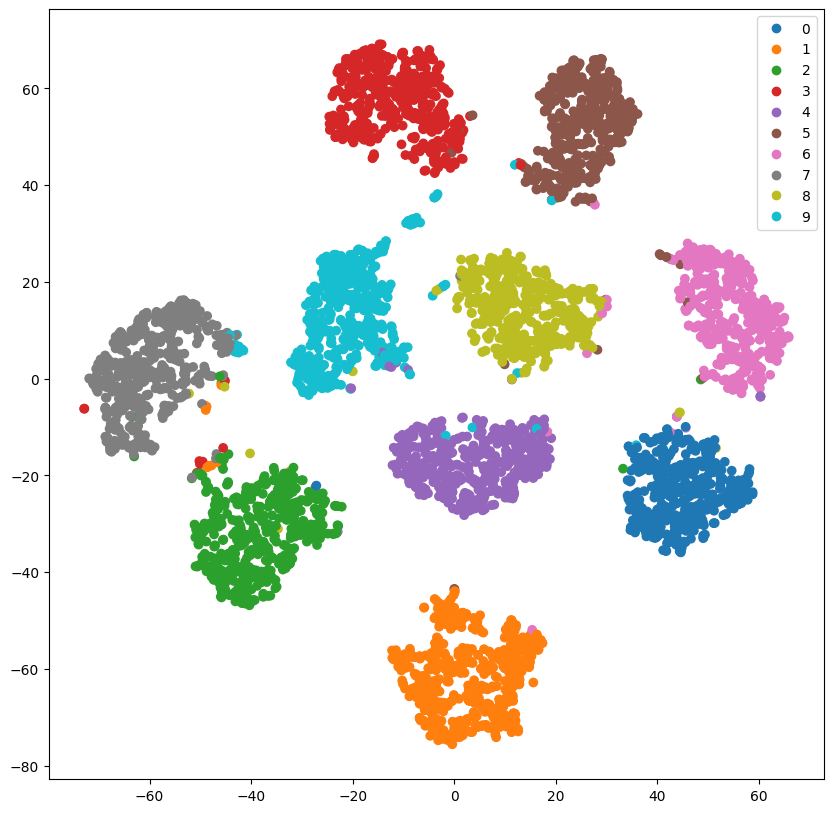
Se observa que se comete un total de 111 errores de un total de 10000 predicciones. Los dos casos con más errores son el “9” con el “4” y el “2” con el “7”. Estos errores significan que en alguna de las imágenes comparten las mismas o parecidas características, lo que causará estas confusiones, pero son casos muy minoritarios que se pueden considerar outliers.

* Comente las diferencias entre el gráfico t-SNE de la representación de las capas final e intermedia de la CNN, aplicado a las imágenes del conjunto de validación. Para ello, considere la proximidad y la dispersión entre los clústeres en ambas representaciones, y su relación con la capacidad de realizar una correcta clasificación de las muestras.

Representación intermedia:



Representación final:



En cuanto a la representación intermedia se ve una clara separación de clases, pero se observan ciertos solapamientos como los casos erróneos comentados en el apartado de la matriz de confusión. Pero de manera general, simplemente con las capas intermedias realiza una clasificación bastante precisa.

Ya al observar las capas finales, se observa que las clases están mucho más separadas, compactas y se producen mucho menos solapamientos, por lo que la precisión es mucho mayor con el uso de las características de las capas finales.

* Dadas las diferencias entre la representación t-SNE de ambas capas, y dada la arquitectura de la red implementada, identifique en qué capa de la red se extraen las características, y proponga una forma de reducir la complejidad de la red, con una penalización baja en la precisión de la clasificación.

Como se ha observado en el apartado anterior, la diferencia entre las capas intermedias a la capa final no es tan significativa, debido a que los conjuntos de datos ya son clasificados en sus respectivas clases salvo por cierto solapamiento de más. Por lo que al observar este aspecto, se podría eliminar la última capa y reducir la complejidad de la red con una pequeña reducción en la precisión. Y observando el modelo creado, las características aprendidas por las capas convolucionales son extraídas a partir de la primera capa lineal.

Puede encontrar [aquí el segundo notebook.](https://drive.google.com/file/d/1xNxvWbpejyvBSqQHoH32W5AKhjpY4i4d/view?usp=sharing) Debe completar el código en el notebook y responder a las preguntas de las Sección 1.2.

## AlexNet

* Incluya el código que ha utilizado para definir la clase Alexnet

class AlexNet(nn.Module):

def \_\_init\_\_(self, output\_dim):

super().\_\_init\_\_()

self.features = nn.Sequential(

# First convolutional layer. Use 5x5 kernel instead of 11x11

nn.Conv2d(3, 48, 5, 2, 2), #in\_channels, out\_channels, kernel\_size, stride, padding

nn.MaxPool2d(2), #kernel\_size

nn.ReLU(inplace = True),

# Complete the following four conv layers of the AlexNet model.

# Subsampling is only performed by 2x2 max pooling layers (not with stride in the

# convolutional layers)

# Pay special attention to the number of input and output channels of each layer

nn.Conv2d(48, 128, 5, 1, 2),

nn.MaxPool2d(2),

nn.ReLU(inplace=True),

nn.Conv2d(128, 192, 3, 1, 1),

nn.ReLU(inplace=True),

nn.Conv2d(192, 192, 3, 1, 1),

nn.ReLU(inplace=True),

nn.Conv2d(192, 128, 3, 1, 1),

nn.MaxPool2d(2),

nn.ReLU(inplace=True)

)

self.classifier = nn.Sequential(

# First linear layer

nn.Dropout(0.5),

nn.Linear(128 \* 2 \* 2, 2048), # final conv layer resolution 2x2

nn.ReLU(inplace = True),

# second linear layer

nn.Dropout(0.5),

nn.Linear(2048, 2048),

nn.ReLU(inplace=True),

# Last Linear layer. No ReLU

nn.Linear(2048, output\_dim),

)

def forward(self, x):

x = self.features(x)

interm\_features = x.view(x.shape[0], -1)

x = self.classifier(interm\_features)

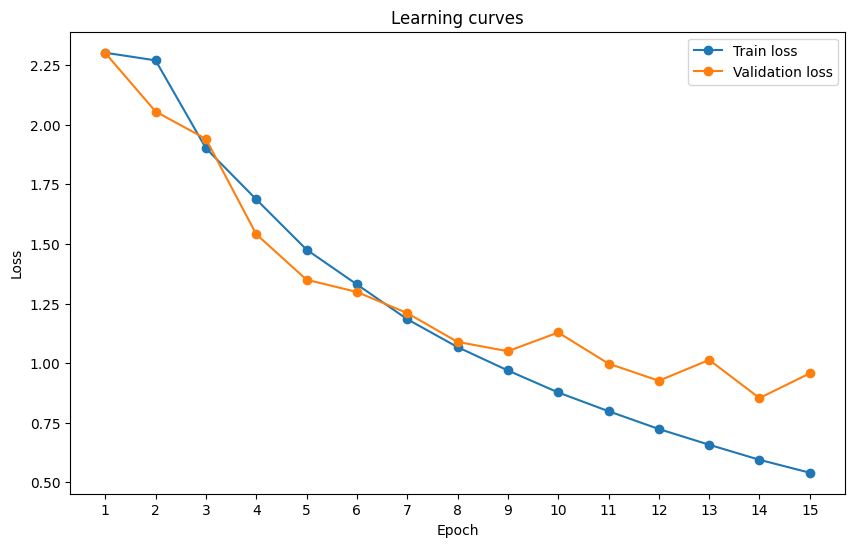
return x, interm\_features

* Número de parámetros del modelo AlexNet

|  | Nº parámetros entrenables |
| --- | --- |
| AlexNet | 6199498 |

* Incluya las curvas de entrenamiento y validación para 15 épocas. Indique también la mejor precisión obtenida, y en qué época se logra este resultado

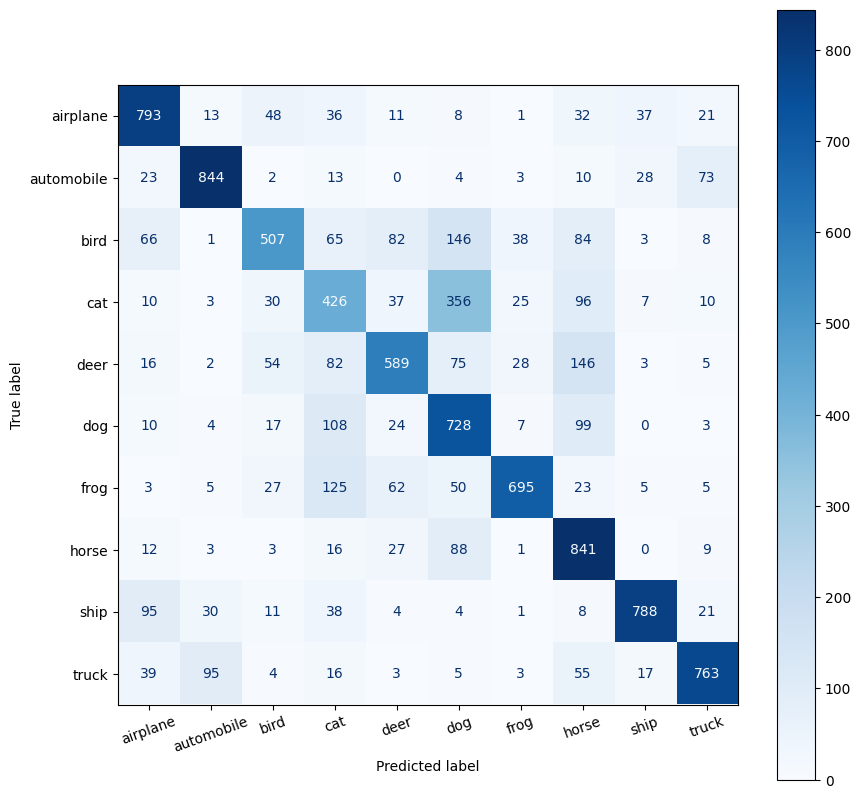
|  | Mejor precisión (validación) | Época con mejor precisión |
| --- | --- | --- |
| AlexNet | 72.01% | 14 |



Comentar las conclusiones sobre la evolución de la *loss* de entrenamiento y validación, y comentar lo que posiblemente está sucediendo después de la época 10. Indique si considera que continuar con más épocas de entrenamiento mejoraría el rendimiento del modelo

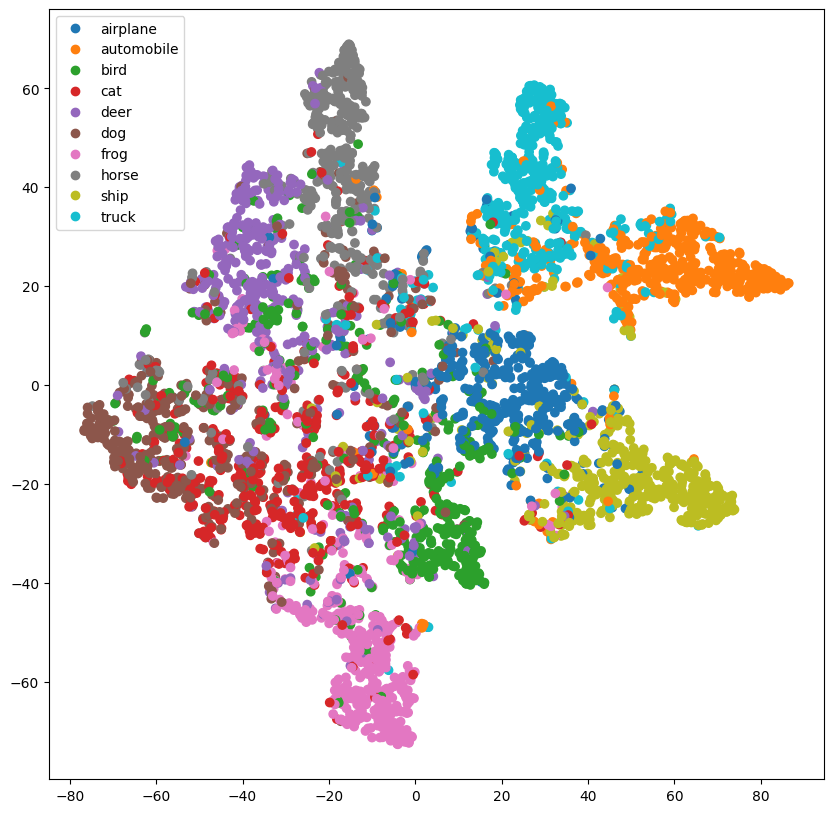
Se observa que el modelo va aprendiendo correctamente por el descenso continuo del valor de loss durante las iteraciones tanto en validación como en entrenamiento, pero a partir de la iteración 10 (realmente diría que a partir de la 9) el valor de loss en validación empieza a hacer cosas extrañas. Se produce un estancamiento y comienzan a ocurrir pequeñas subidas mientras que en training sigue decreciendo continuamente. Esto es debido que a partir de la iteración comentada anteriormente es el punto óptimo, donde si añadimos una capa de EarlyStopping para la regularización se detendría, y a partir de este punto se produce el efecto de sobreajuste.

* Incluir la matriz de confusión. Comentar los resultados obtenidos atendiendo a las características de las imágenes de cada clase



Se observa que las clases donde se producen más errores son la clase “dog”con “cat”, “dog” con “birds”, “horse” con “deer” y “cat” con “frog”. En el caso de “dog” con “cat” y “horse” and “deer” pueden ser entendibles de una manera más lógica y simples, ya que son casos de animales con un tamaño, color y características morfológicamente parecidas que dependiendo de otros aspectos como la pose, la distancia o el contexto de la imagen. En cuanto a los otros casos de forma lógica son animales muy diferentes, pero al tratar ciertas imágenes se obtendrán ciertas características parecidas y por eso se producirán estos errores. O por las técnicas utilizadas, ya que con otros modelos se podrían obtener resultados más exactos.

* Incluya los resultados t-SNE para la capa última capa de la red: analice estos resultados (proximidad, dispersión, agrupación de clústeres) teniendo en cuenta la apariencia de las imágenes de las diferentes clases, sus características típicas y compare los resultados con los resultados t-SNE en el *dataset* MNIST.



Hay ciertas clases que tienen cierta separación, pero se observa mucho solapamiento entre ellas. Esto se ha comentado anteriormente con lo que ocurre con las clases “dog”, “cat”, “bird”, “horse” y “deer”. Por lo que a la conclusión que se ha llegado es que la clasificación de MNIST con t-SNE no es correcta, pues el modelo no tiene la capacidad de diferenciar correctamente las clases.

Puedes encontrar [aquí el tercer](https://drive.google.com/file/d/1Yug_QXpw1yrmWvKHrN83BfOPsjOUK9ma/view?usp=sharing) [*notebook*.](https://drive.google.com/file/d/1Yug_QXpw1yrmWvKHrN83BfOPsjOUK9ma/view?usp=sharing) Debe completar el código en el *notebook* y responder a las preguntas de la Sección 1.3.

## Transfer Learning

* Precisiones obtenidas para las diferentes alternativas analizadas:

|  | Entrenado desde cero | Pre-entrenamiento + SVM | Ajuste fino (sin *data augmentation*) | Ajuste fino (con *data augmentation*) |
| --- | --- | --- | --- | --- |
| Precisión | 71.5% | 88% | 91.5% | 95% |

* Compare las representaciones t-SNE de las diferentes alternativas: entrenamiento desde cero, pre-entrenamiento + SVM, ajuste fino (sin *data augmentation*) y ajuste fino (con *data augmentation*) A partir de las diferentes representaciones obtenidas, en las cuatro alternativas analizadas., comente sus diferencias en cuanto a la capacidad de separar linealmente ambas clases, y el nivel de muestras clasificadas erróneamente dada esta separación lineal.

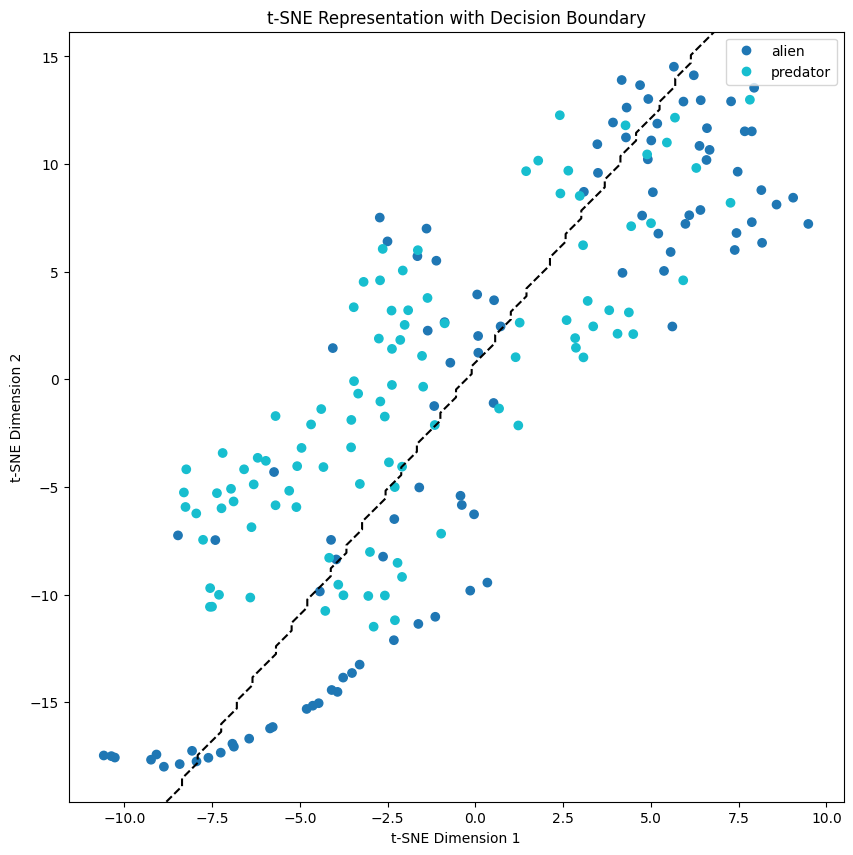
Se observa una mejora con el añadido de las distintas técnicas, donde el mayor salto en cuanto a la precisión se obtiene entre el entrenamiento de scratch desde 0 a utilizar pre-entrenamiento más SVM.

Esto es debido claramente a la transferencia de información utilizando un modelo pre entrenado como extractor de características y además añadir un clasificador adicional en este caso SVM. Esto resulta muy beneficioso, ya que se puede aprovechar las características ya aprendidas de un conjunto de datos mucho mayor, los cuales son utilizados para entrenar el modelo SVM. También los modelos pre entrenados suelen estar regularizados durante el entrenamiento, por lo que también influirá en el rendimiento y en evitar el sobreajuste.

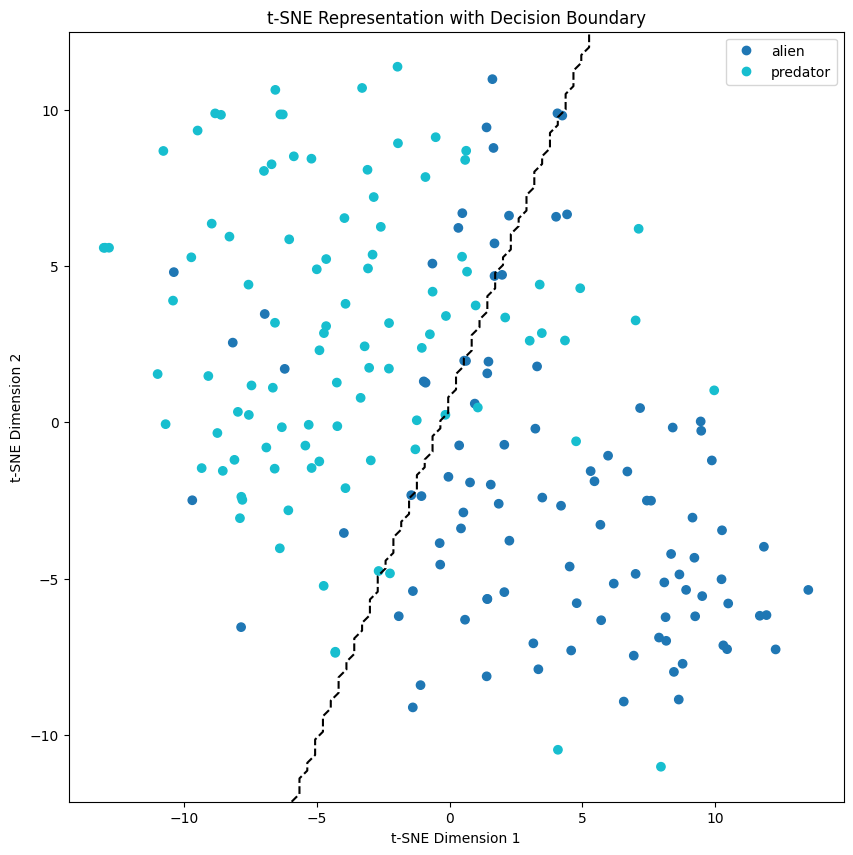
Comparando pre entrenamiento más SVM, ajuste fino (sin data augmentation) y ajuste fino (con data augmentation) no se produce una mejora tan alta (7% como mucho entre pre entrenamiento + SVM y data augmentation). Aunque la mejora no haya sido grande, puede ser simplemente por el conjunto de datos utilizados y el problema a resolver. En casos más complejos el adaptar los pesos pre entrenados de un modelo pre entrenado a una tarea objetivo puede ser muy beneficioso, al poder capturar características relevantes específicas al problema.a resolver. Y el uso de data augmentation también resulta muy beneficioso debido al aumentar la variabilidad en el conjunto de datos a través de transformaciones sobre el conjunto de datos, el cual puede ser muy pequeño, como pueden ser rotaciones, recortes o cambios de brillo (en nuestro caso un cambio de tamaño en la imagen, un recorte más pequeño que el tamaño de la imagen y una probabilidad de 0.5 de rotar horizontalmente).

Visualmente se observa la mejora en la clasificación. En el caso desde cero la clasificación no se realiza ni de cerca para considerarse correcta, en cuanto al pre entrenado + SVM se produce una gran mejora pero sigue habiendo una pequeña cantidad de datos mal clasificados (lo considero un resultado bueno, mayoria de datos bien clasificados), ajuste fino sin data augmentation se observa una pequeña mejora en la clasificación pero sigue habiendo un pequeña cantidad mal clasificada y ajuste fino con data augmentation clasifica casi perfectamente bien, pudiendo considerar los valores mal clasificados como outliers.

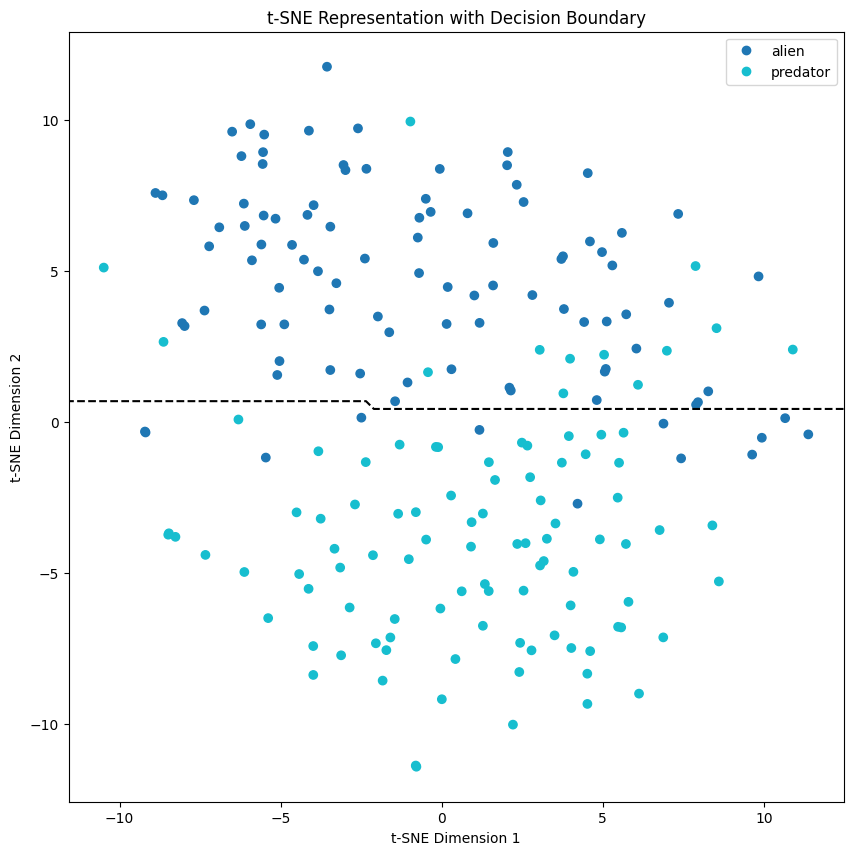
* Desde cero:



* Pre entrenado + SVM:



* Ajuste fino (sin data augmentation):



* Ajuste fino (con data augmentation):

