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.

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. Debe completar el código en el notebook y responder a las preguntas de la Sección 1.1.

1.1 Simple CNN

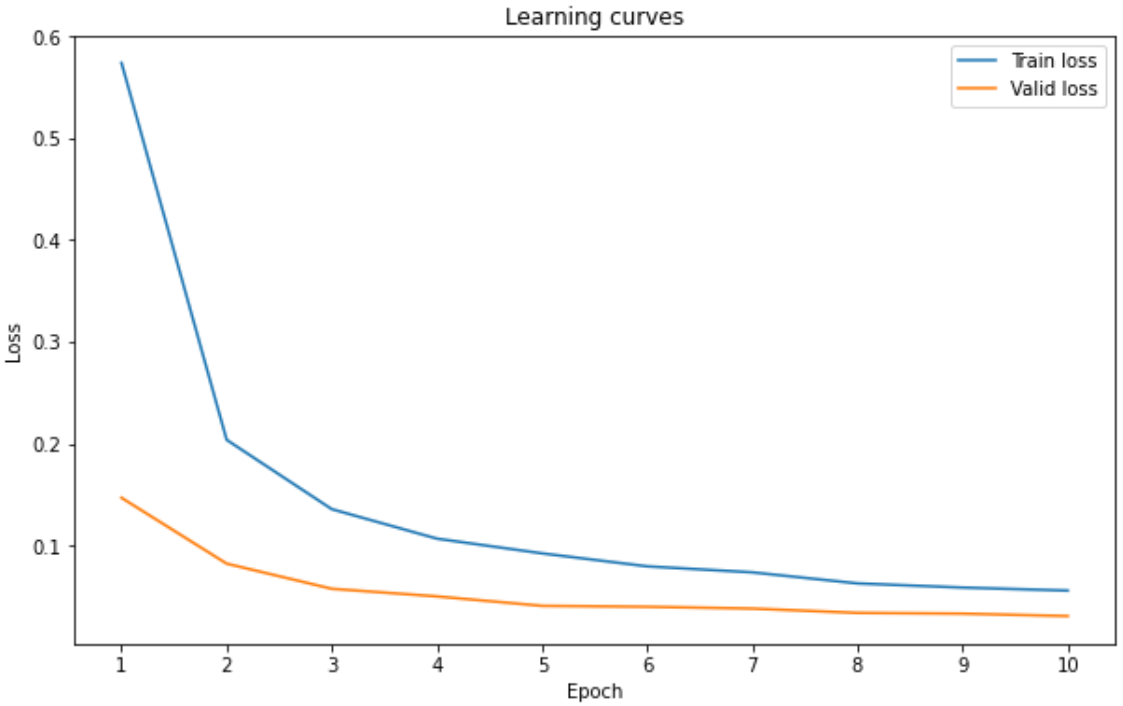
- Tamaños de los conjuntos de entrenamiento y validación descargados del *dataset* MNIST

	Alto de imagen	Ancho de imagen	N.º canales de imagen	N.º muestras
Entrenamiento	28	28	1	60000
Validación	28	28	1	10000

- Número de parámetros del modelo Simple CNN

	N.º parámetros entrenables
Simple CNN	813600 (sin bias)

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

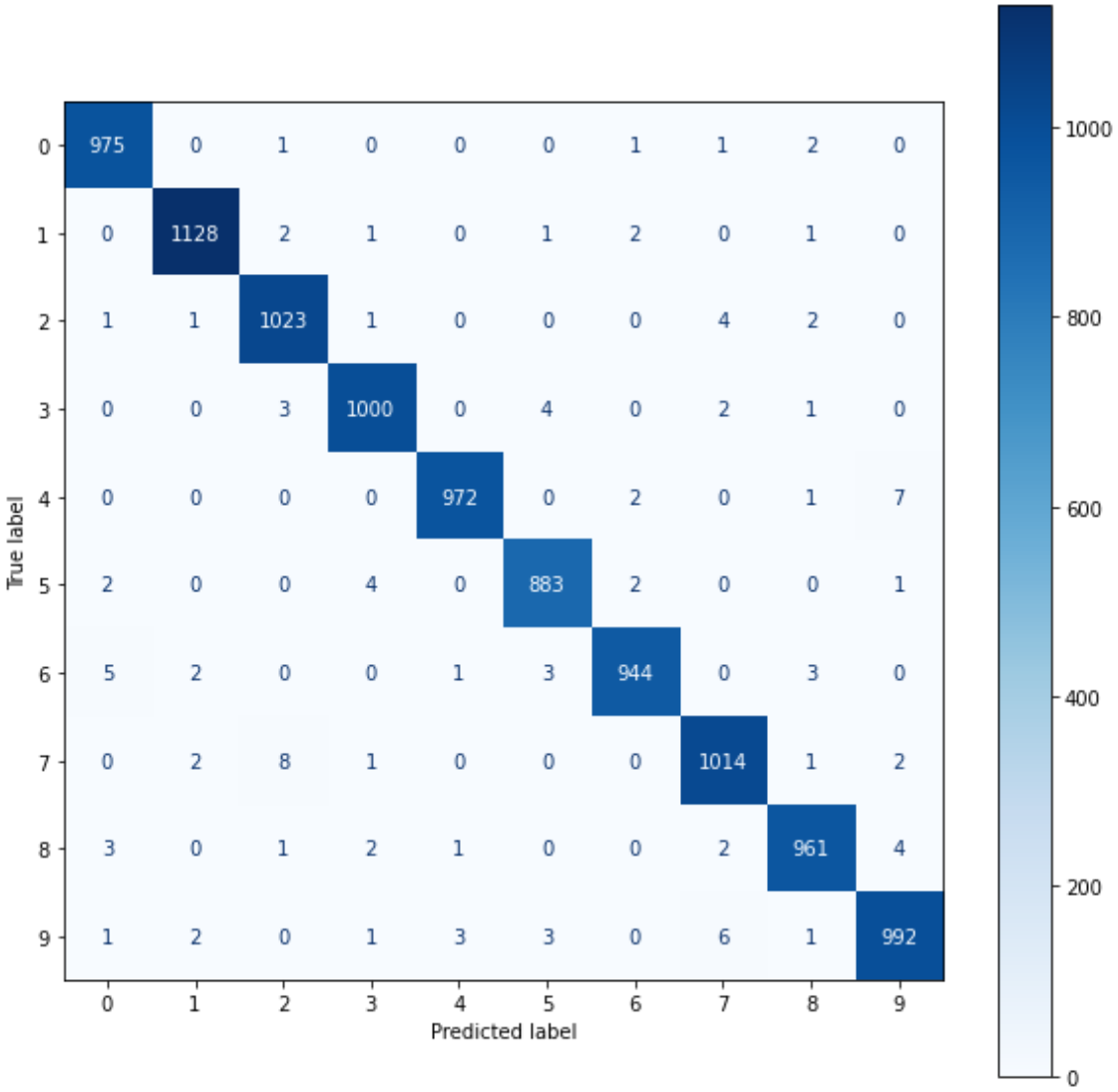


	Mejor precisión (validación)	Época con mejor precisión
Simple CNN	98.9200%	10

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

Tanto la *loss* en entrenamiento como en validación desciende durante todas nuestras épocas. Esto indica que no estamos sobreajustando, pues subiría el error en validación mientras sigue descendiendo en entrenamiento. Sin embargo, las pendientes para ambas curvas parecen estabilizarse. Un número mayor de épocas podría aumentar el rendimiento del modelo, pero la mejora sería menor en cada época adicional, siendo ya casi irrelevante.

- 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.

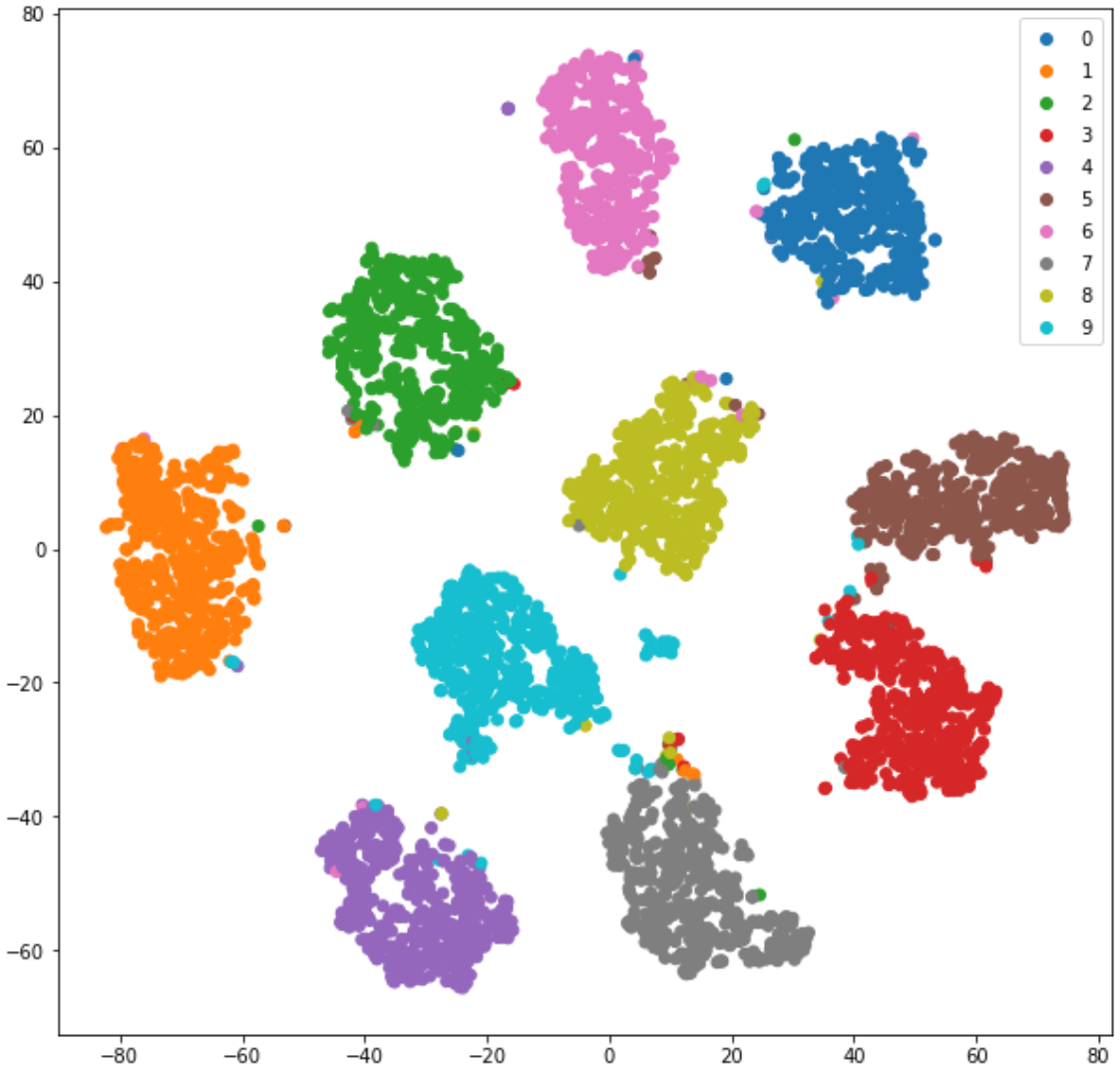


Los dos casos de confusión más frecuentes son:

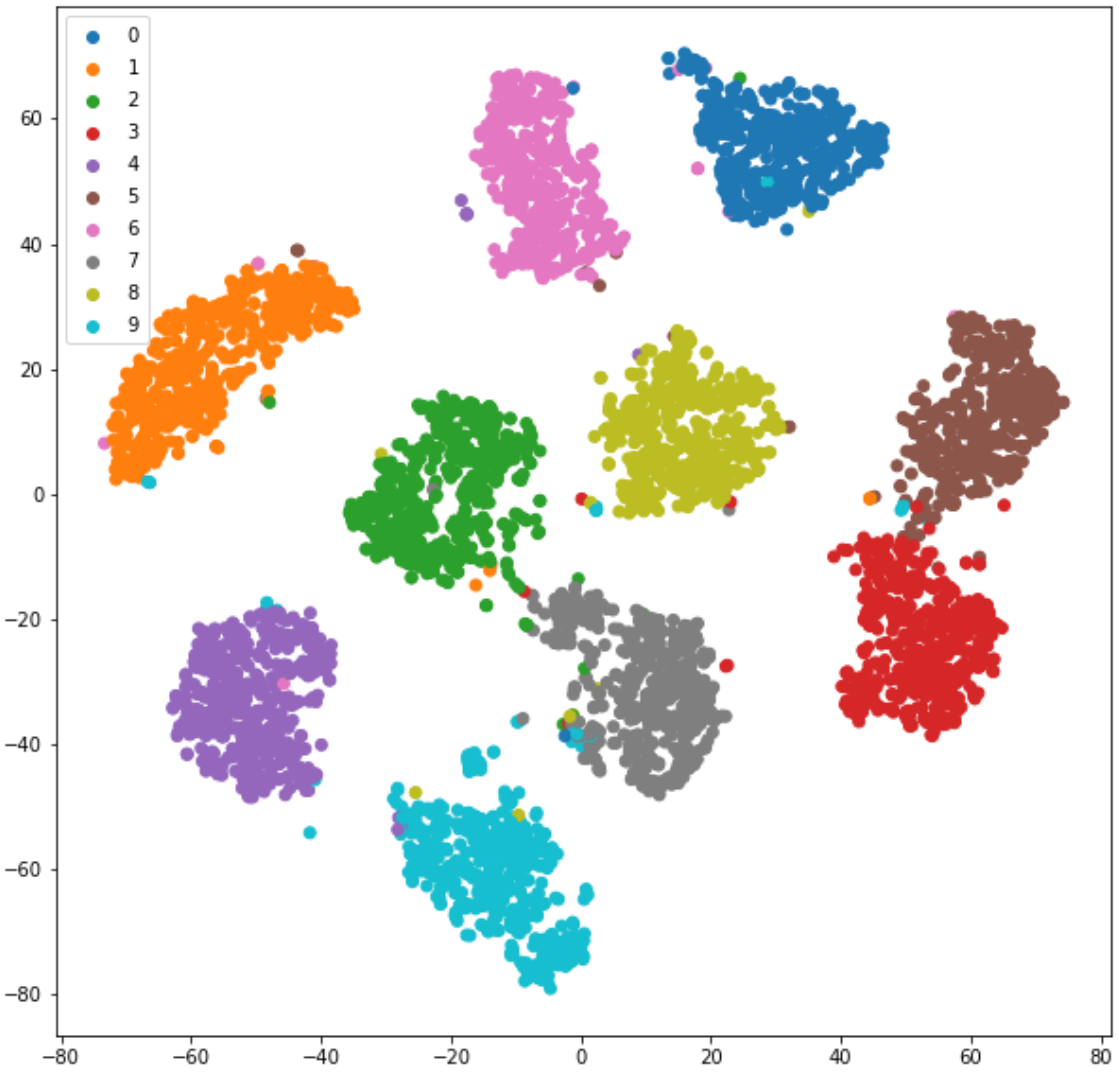
- Asociar a un 4 la etiqueta de un 9.
- Asociar a un 7 la etiqueta de un 2.

- 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.

En la última capa:



En la capa intermedia:



Vemos como los clusters están más separados. Esto es claro entre las clases 2, 7 y 8; así como entre las clases 7 y 90. También aumenta la distancia entre clusters.

Sin embargo, ambas representaciones son notablemente parecidas. Si nos quedamos con las características extraídas por las capas intermedias ya deberíamos ser capaces de separar las clases casi con la misma precisión y el número de parámetros de la red se reduciría considerablemente.

- 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 ya hemos comentado, la representación de las capas intermedias es más que suficiente para esta clasificación. Esto es, podemos quitar la última capa *fully connected* de nuestra red (con (128 + 1)*10 parámetros) y obtener casi la misma representación.

Puede encontrar aquí el segundo notebook. Debe completar el código en el notebook y responder a las preguntas de las Secciones 6.2 y 6.3.

1.2 AlexNet

- Incluya el código que ha utilizado para definir la `clase Alexnet`

Incluimos únicamente el código del método `__init__`, el que nosotros hemos implementado:

```
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

            # First (128) block
            nn.Conv2d(48, 128, 5, stride=1, padding=2),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True),

            # Second (192) block
            nn.Conv2d(128, 192, 3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 3, stride=1, padding=1),
            nn.ReLU(inplace=True),

            # Second (128) block
            nn.Conv2d(192, 128, 3, stride=1, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True)

        )

        self.classifier = nn.Sequential(
            # First linear layer
            nn.Dropout(p=0.5),
            nn.Linear(128*2*2, 2048), # final conv layer resolution 2x2
            nn.ReLU(inplace = True),

            # Second linear layer
            nn.Dropout(p=0.5),
            nn.Linear(2048, 2048),
            nn.ReLU(inplace=True),

            # Third Linear layer, without dropout nor ReLu
            nn.Linear(2048, output_dim),
        )
```

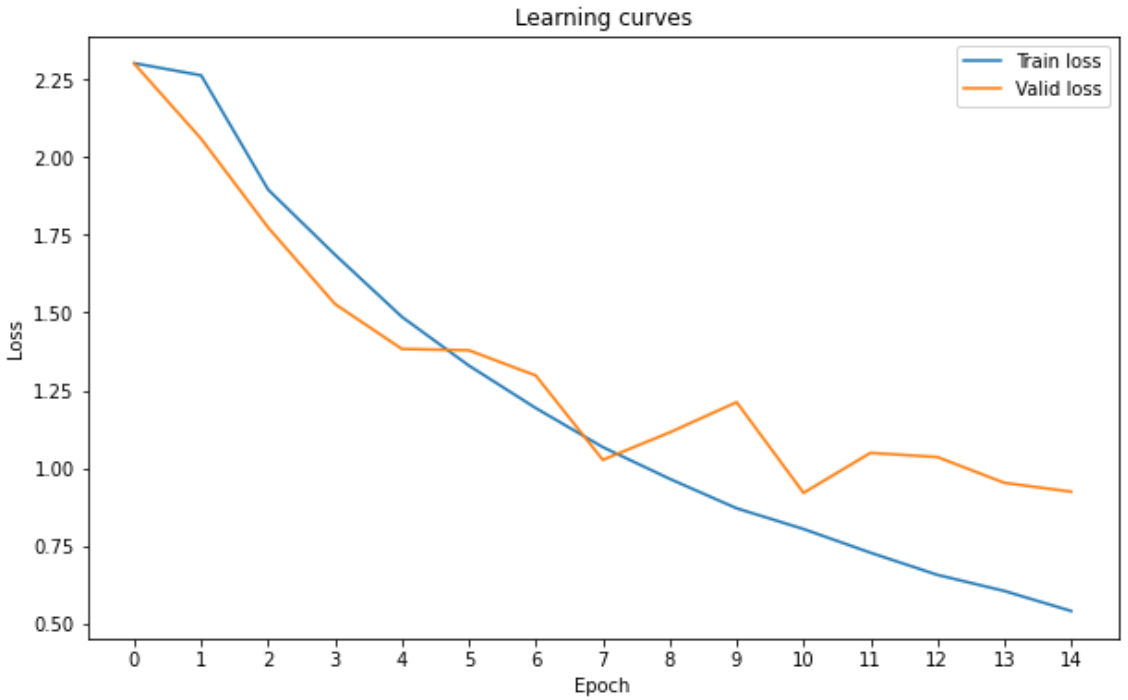
- Número de parámetros del modelo AlexNet

	Nº parámetros entrenables
AlexNet	6,199,498

- 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	70.1200%	15

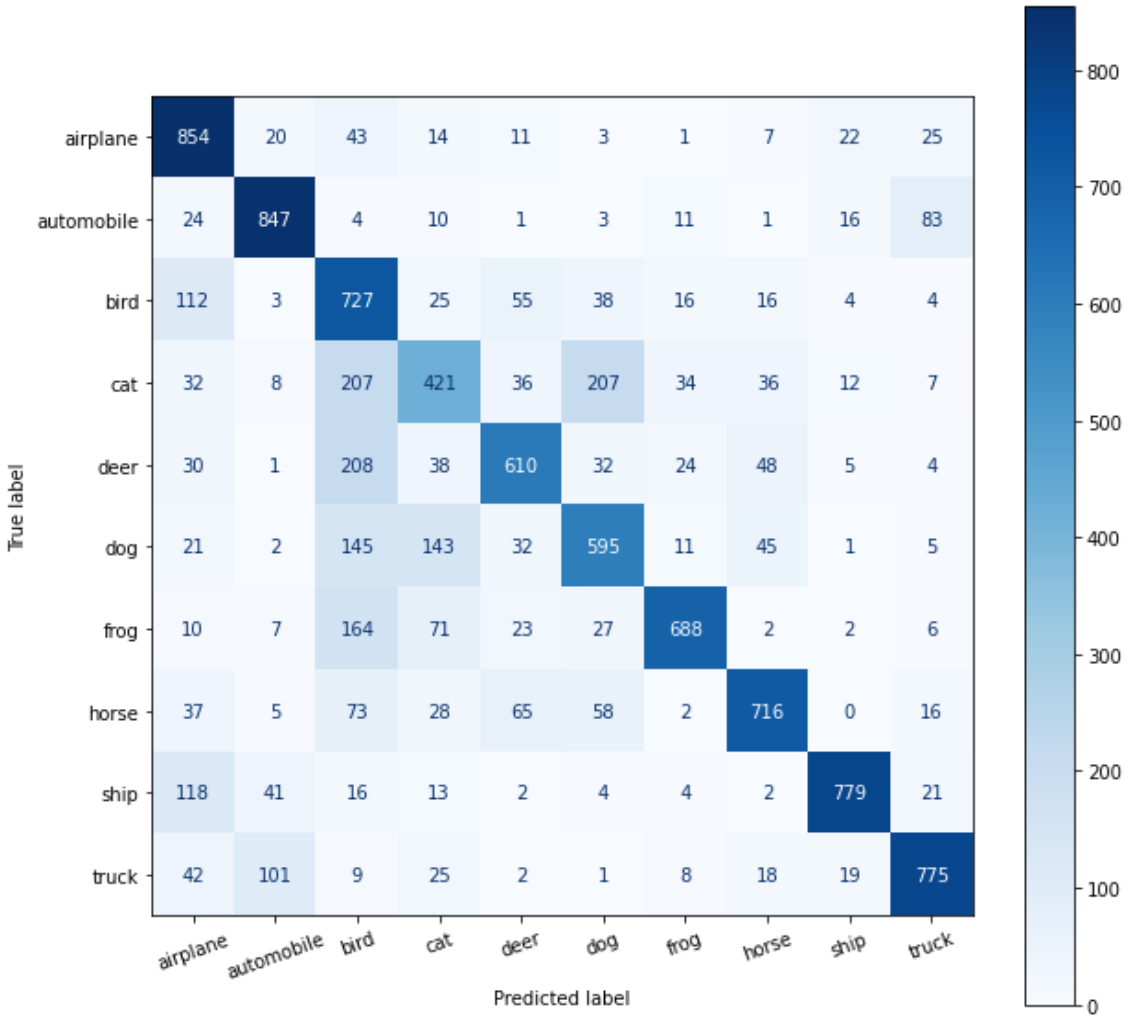
Comentar las conclusiones sobre la evolución de la *loss* de entrenamiento y validación, y comentar lo que posiblemente está sucediendo después de la época 10. Indique si considera que continuar con más épocas de entrenamiento mejoraría el rendimiento del modelo



A partir de la décima época observamos como la *loss* en validación no disminuye, sino que en algunos casos incluso aumenta. Mirando los valores de precisión obtenidos vemos que en alguna época a partir de la décima (en particular, en la décimo primera) la precisión se reduce. Esto significa que estamos realizando *overfitting* a partir de dicha época a los datos de entrenamiento (por ello la *loss* en entrenamiento si que se reduce).

Aumentar el número de épocas llegado este momento es contraproducente: sólamente realizaremos más *overfitting*, obteniendo un peor resultado en validación.

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



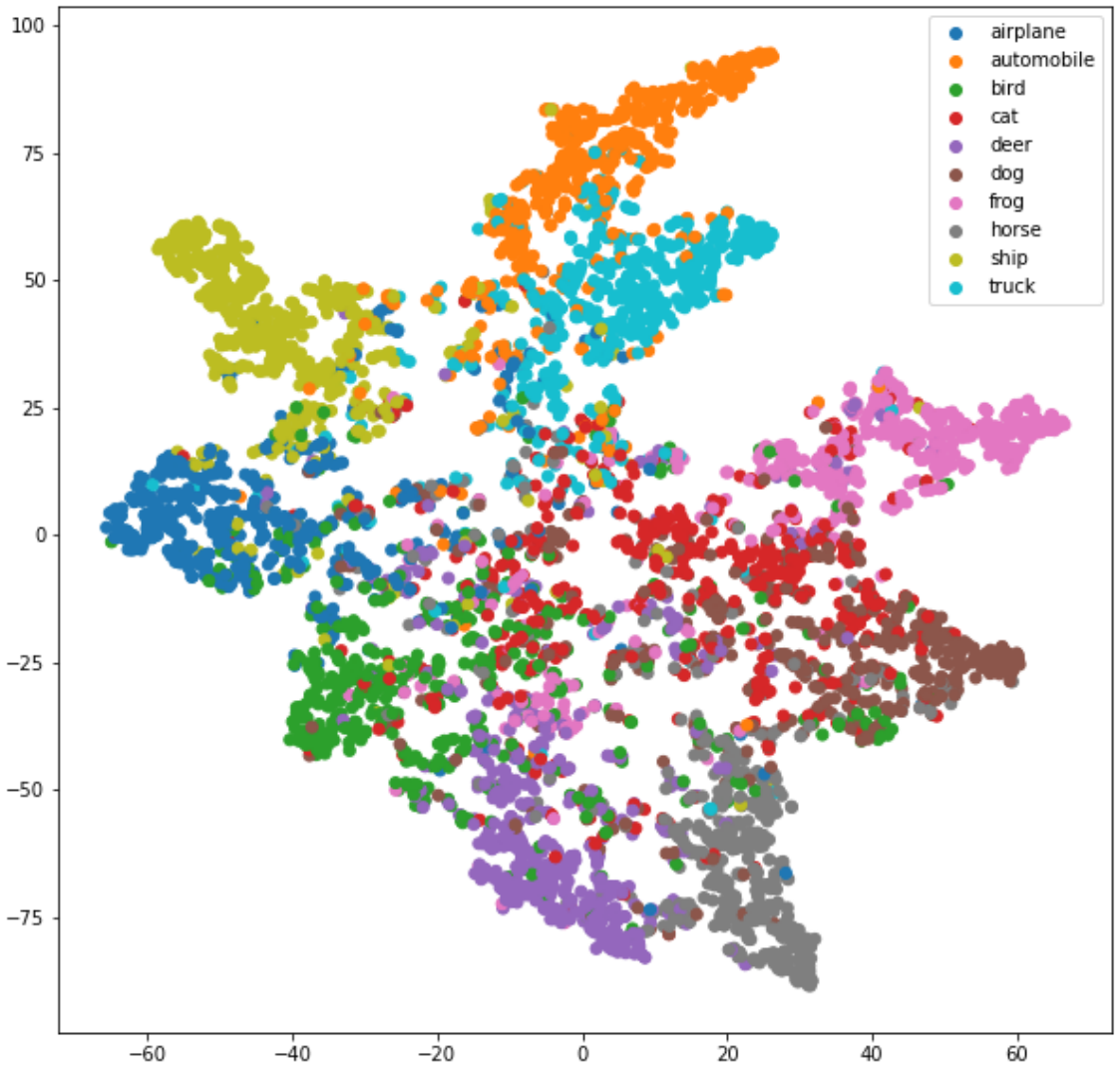
En primer lugar llama la atención como la clase **gato** es la menos acertada, confundiéndose a menudo con **pájaros** o **perros**. En el segundo caso es más comprensible pues las imágenes de gatos y perros tienen formas parecidas, pero es algo más raro confundirlos con pájaros. Adicionalmente, viendo la confusión obtenida entre gatos y perros sería razonable pensar que hay alta confusión entre gatos, perros y **caballos** en cualquier combinación, pero no es el caso. Ésta última clase suele distinguirse relativamente bien de las demás.

Vemos también que **pájaro** es la clase más marcada equivocadamente, lo que es realmente curioso. Además, habiendo imágenes de **aviones** en nuestro dataset, uno esperaría cierta confusión parecida con los aviones, pero no es el caso. Una posible aproximación a mejorar nuestro modelo podría comenzar por observar cómo son las imágenes de pájaros del conjunto de entrenamiento, y por qué nuestro modelo está sesgado a escoger esta clase.

Cabe destacar algunas otras parejas de elementos parecidas que se confunden a menudo: además de los mencionados (**gato, perro**) tenemos (**automóvil, camión**), (**barco, avión**) y (**pájaro, avión**).

Finalmente, hay un único 0 en nuestra matriz de confusión (aunque hay muchos otros valores por debajo de 5), lo que muestra que el modelo realmente distingue entre estas clases. En nuestro caso se trata de las clases **barco** y **caballo**.

- 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.



Como podemos apreciar, esta agrupación de clusters es radicalmente distinta a la obtenida en el dataset MNIST. La menor precisión obtenida en este dataset encaja claramente con esta agrupación, pues los clusters no están bien separados y cualquier intento de separarlos linealmente sería un absoluto desastre.

Cabe destacar que la clasificación no se realiza en el espacio de dos dimensiones que estamos viendo, sino en un espacio de dimensión muchísimo mayor. En dicho espacio quizás sí sea posible una mejor separación lineal de los datos, pero desde luego el algoritmo de t-SNE no infiere correctamente dicha información si éste es el caso.

Comencemos a observar los clusters. En la parte superior vemos los clusters de automóvil y avión bastante intersecados, lo que no coincide particularmente bien con nuestra matriz de confusión: no parece haber mucho problema en la distinción de estas dos clases entre sí. Sin embargo, los clusters de gato y perro (rojo y marrón respectivamente) si tienen mucho más solapamiento, volviéndose complicados de distinguir.

Estudiando el sesgo del pájaro comentado anteriormente, vemos como el cluster con dicha clase (verde) queda totalmente diseminado por la mitad inferior de la imagen, solapando los clusters de ciervo, caballo, gato, perro y, parcialmente, rana. Esto encaja a la perfección con los resultados obtenidos en la matriz de confusión: al obtener un cluster tan diseminado el modelo infiere que muchos otros elementos

del centro de la imagen pertenecen a la clase del pájaro.

Finalmente, podemos observar atentamente cualquier clúster y ver al menos algún elemento suelto perteneciente a cualquier otra clase. Esto vuelve a encajar con nuestra matriz de confusión, donde tenemos al menos 1 error para todas las parejas de clases (excepto para una, como ya comentamos).

Puedes encontrar aquí el tercer *notebook*. Debe completar el código en el *notebook* y responder a las preguntas de la Sección 2.1.1.

1.3 Transfer Learning

- Precisiones obtenidas para las diferentes alternativas analizadas:

Utilizamos los datos obtenidos por cada modelo en la última época, lo que no tiene por qué concordar con el valor obtenido en la mejor época.

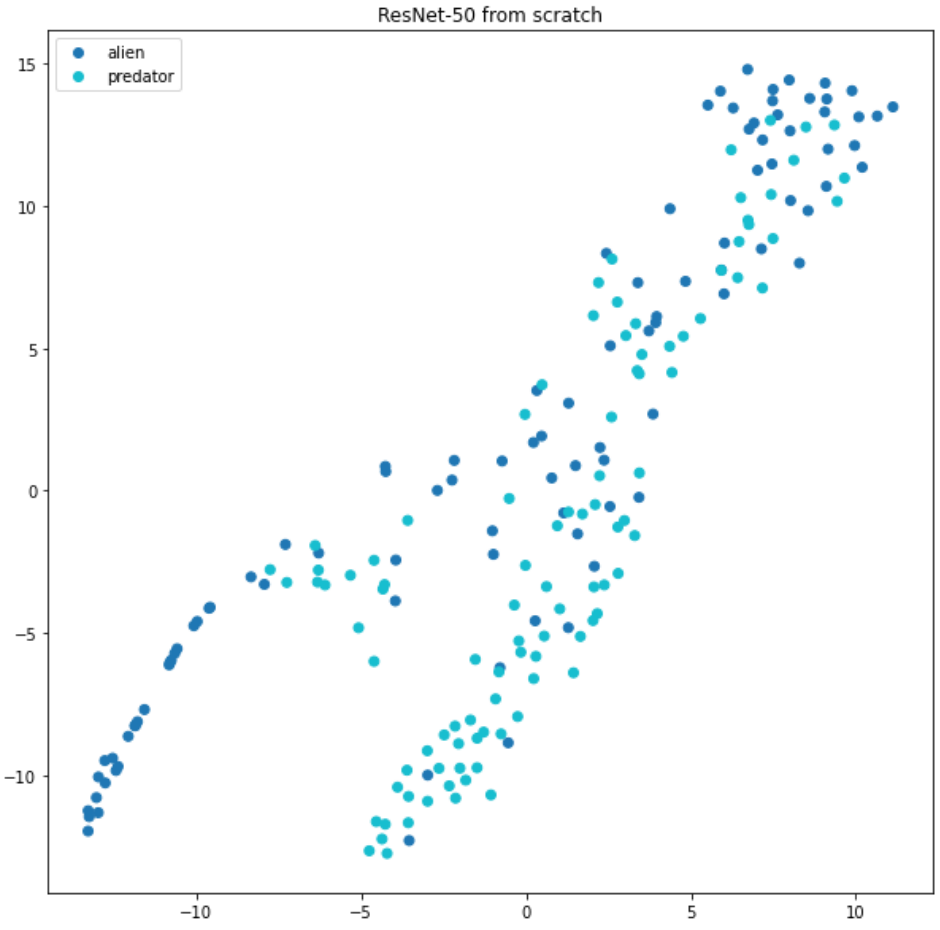
	Entrenado desde cero	Pre-entrenamiento + SVM	Ajuste fino (sin *data augmentation*)	Ajuste fino (con *data augmentation*)
Precisión	0.7200	0.905	0.9100	0.9150

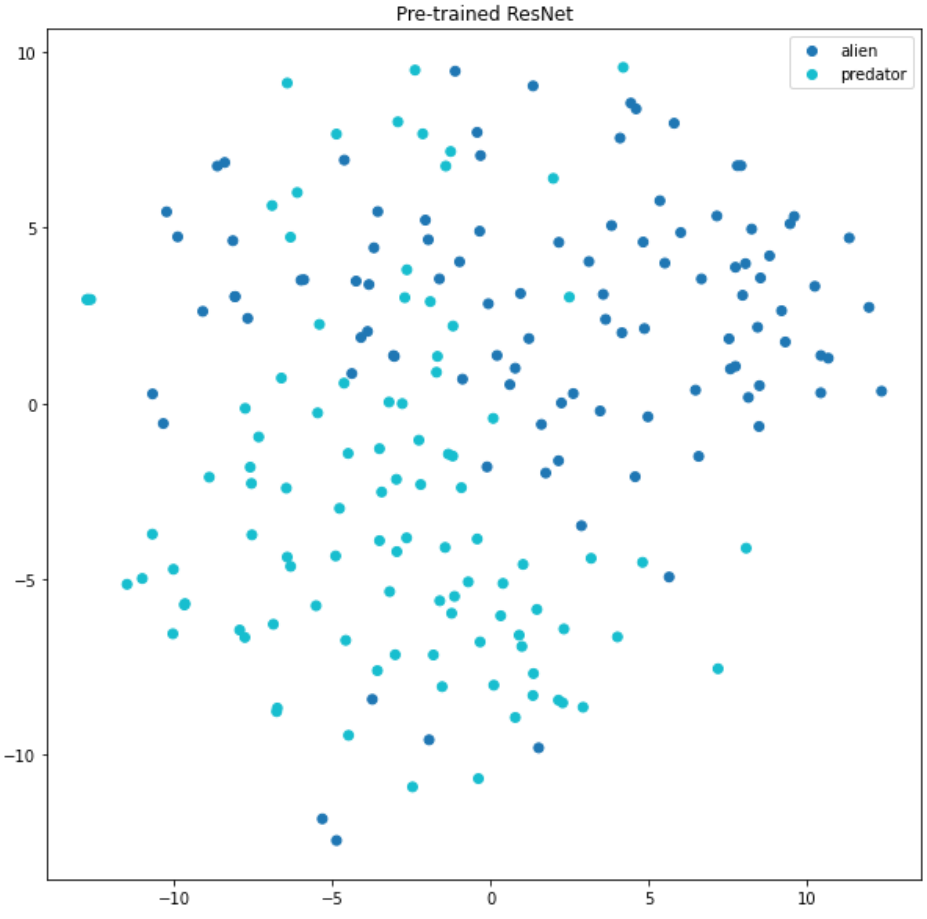
- Compare las representaciones t-SNE de las diferentes alternativas: entrenamiento desde cero, pre-entrenamiento + SVM, ajuste fino (sin *data augmentation*) y ajuste fino (con *data augmentation*) A partir de las diferentes representaciones obtenidas, en las cuatro alternativas analizadas., comente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

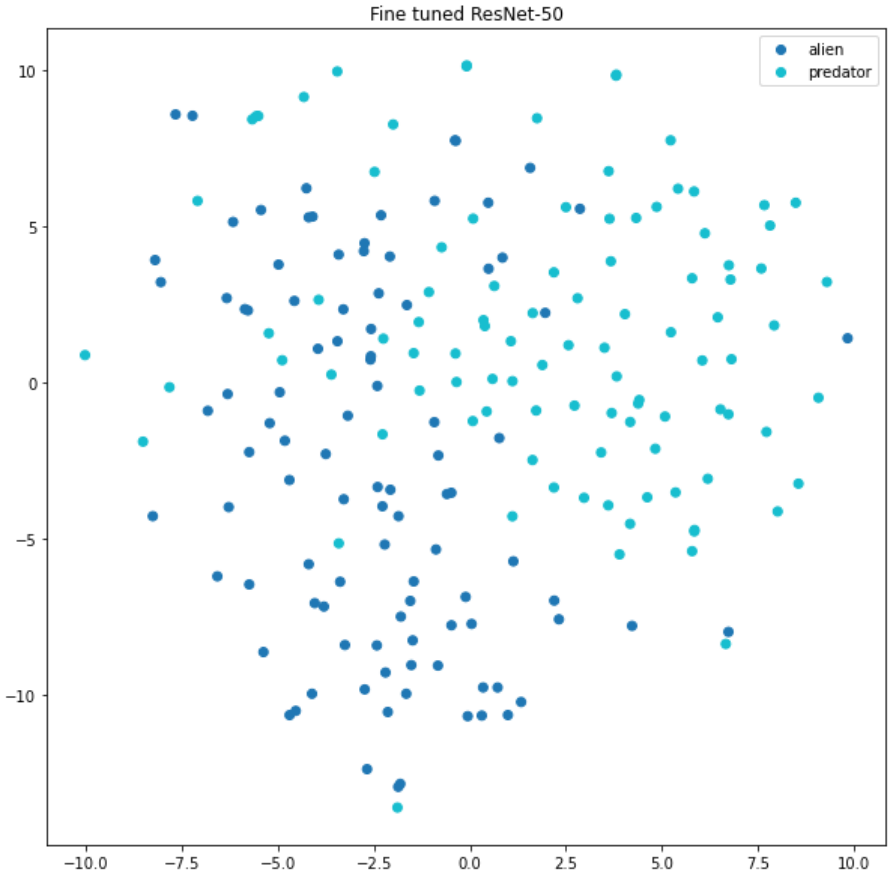
Las gráficas obtenidas se adjuntan más abajo.

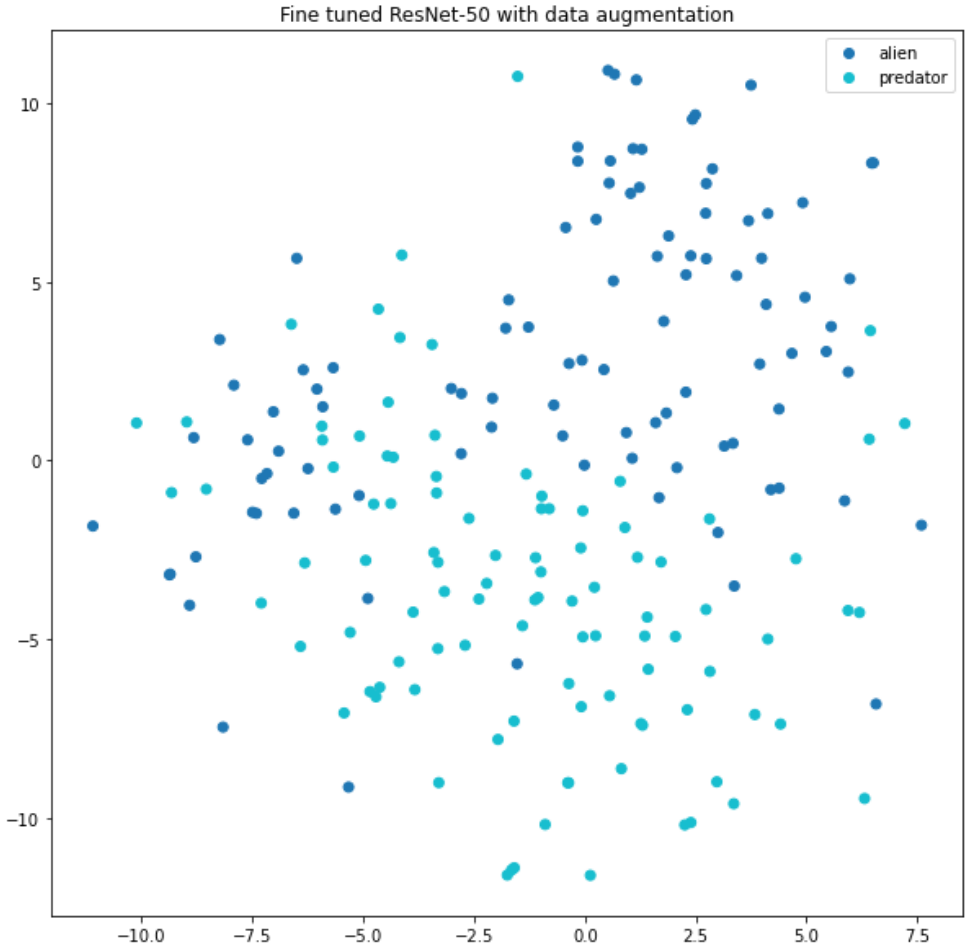
En primer lugar destacamos como las gráficas de *pre-trained*, *fine tune* y *data augmentation* son bastante similares, no pudiendo distinguir cuál es mejor visualmente. Esto encaja con los valores obtenidos para la precisión, pues son tan poco distintos entre sí que la diferencia geométrica se vuelve imperceptible.

En cuanto a la separación lineal de las clases, ResNet-50 entrenado por nosotros queda totalmente descartado. Los otros modelos se comportarían mejor, pero tampoco con resultados especialmente buenos. De nuevo hemos de recordar que la reducción de dimensionalidad puede ser engañosa: el clasificador final nunca se aplica sobre este espacio de dimensión reducida sino en el espacio de características obtenido por nuestra red. Es por ello que la precisión es notablemente distinta a lo que obtendríamos al aplicar un clasificador lineal a las representaciones obtenidas por t-SNE.


ResNet-50 from scratch
alien
predator
15
10
5
0
−5
−10
−10
−5
0
5
10


Pre-trained ResNet
alien
predator
10
5
0
−5
−10
−10
−5
0
5
10


Fine tuned ResNet-50
alien
predator
10
5
0
−5
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Fine tuned ResNet-50 with data augmentation
alien
predator
10
5
0
−5
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5