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

ADRIÁN RUBIO PINTADO 09/04/2022

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	3	60000
Validación	28	28	3	10000

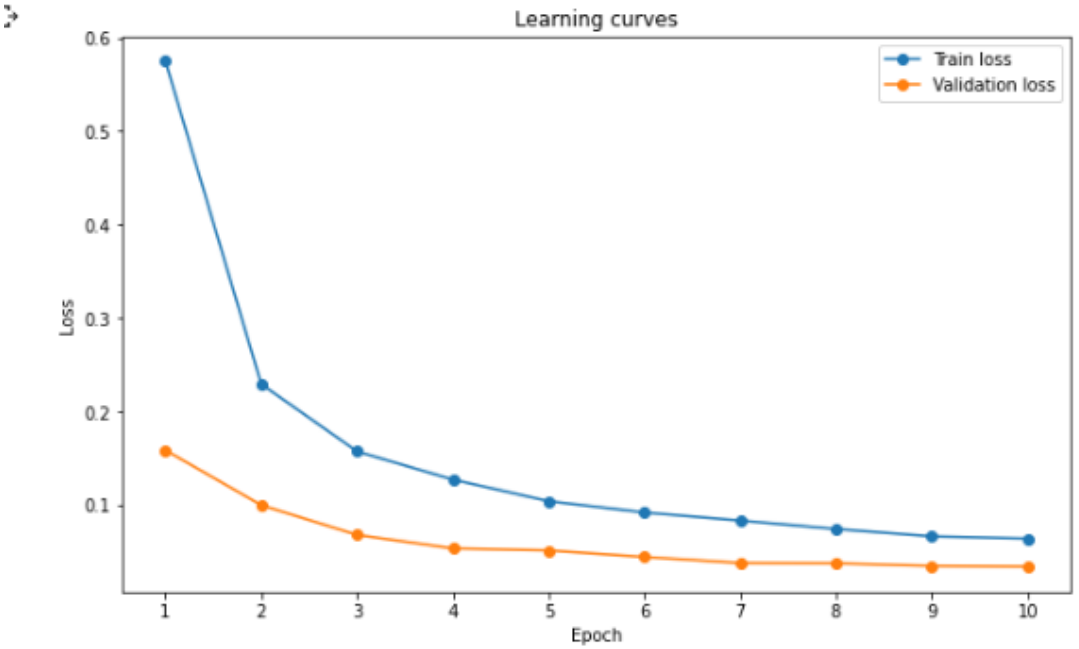
- Número de parámetros del modelo Simple CNN

	N.º parámetros entrenables
Simple CNN	813802

Podemos desglosarla en los parámetros de cada capa de la red tal que:

```
+--------------+------------+
|   Modules    | Parameters |
+--------------+------------+
| conv1.weight |    288     |
|  conv1.bias  |     32     |
| conv2.weight |    9216    |
|  conv2.bias  |     32     |
|  fc1.weight  |   802816   |
|   fc1.bias   |    128     |
|  fc2.weight  |    1280    |
|   fc2.bias   |     10     |
+--------------+------------+
Total Trainable Params: 813802
813802
```

- 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	98.9400%	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

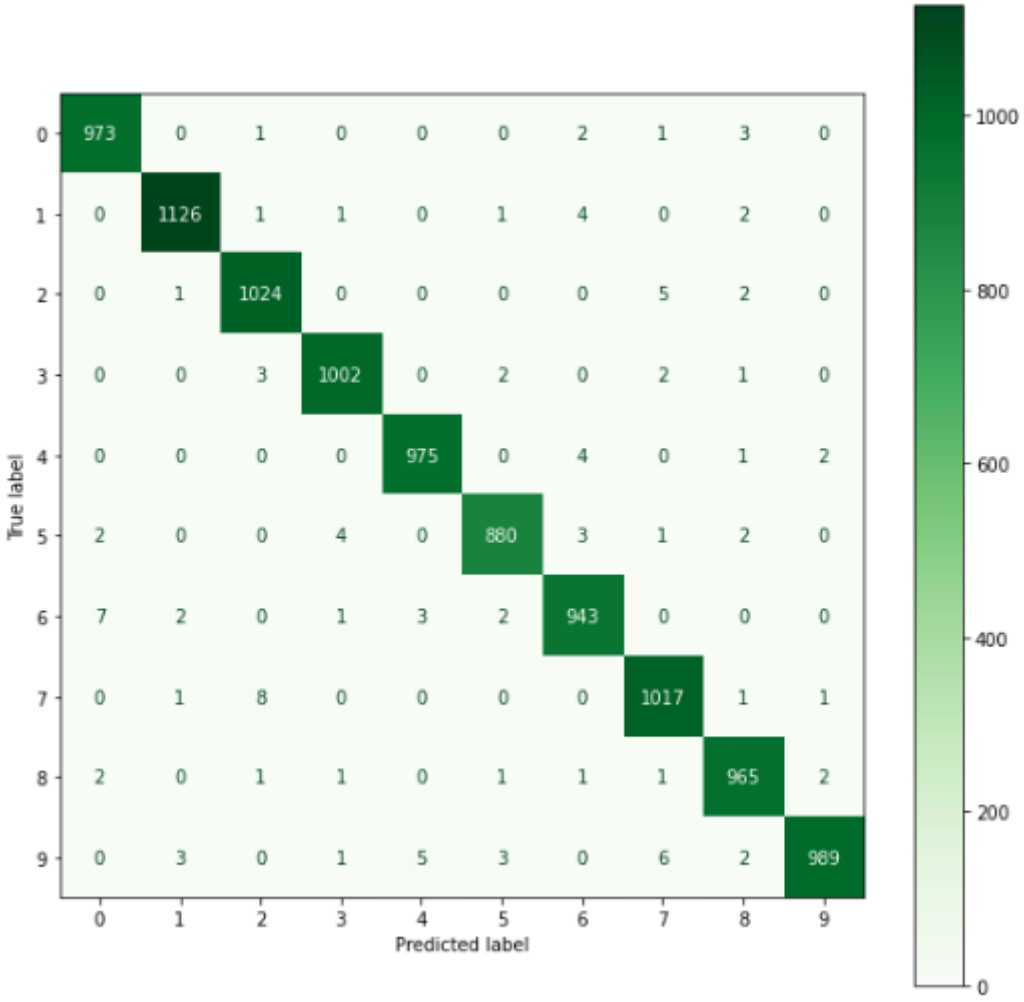
Ambas loss o pérdidas van disminuyendo a medida que nos aproximamos a la época 10, lo cual nos indica que nuestro modelo mejora y cada vez tiene más acierto. No se ven problemas

de sesgo, ya que el modelo va dismininuyendo su pérdida y hasta la época 10 se la pasa aprendiendo. Esto es probable que se de gracias a los dropouts que hacemos en la red, para que se generalice sobre los datos.

Tampoco podemos ver overfitting, ya que la curva de validación está siempre por debajo de la de loss mientras ambas disminuyen de valor con las épocas.

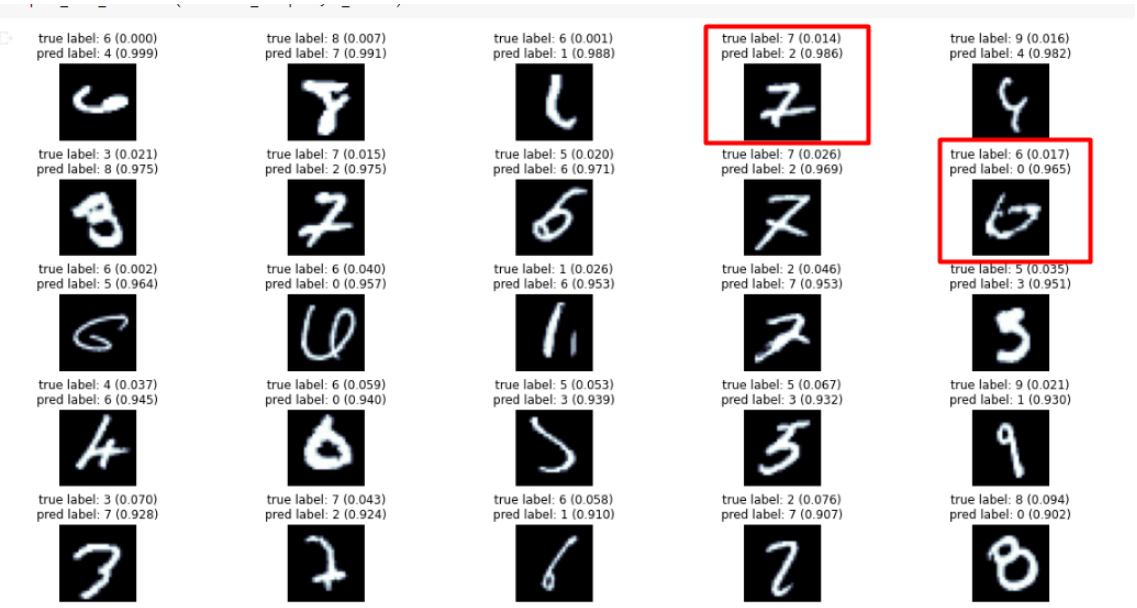
En un principio todo nos hace indicar dadas las curvas, que el model mejoraría con más épocas de entrenamiento. De hecho hemos entrenado el modelo otras 10 épocas más al final del notebook, y hemos visto que el error de validación sigue disminuyendo(es decir mejoramos el modelo). Aunque si que es cieto que cerca de la época 20, dicha curva tiende a elevarse, lo que nos indica que ya estaríamos haciendo overfitting y que no podremos mejorar mucho más el modelo.

- 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.

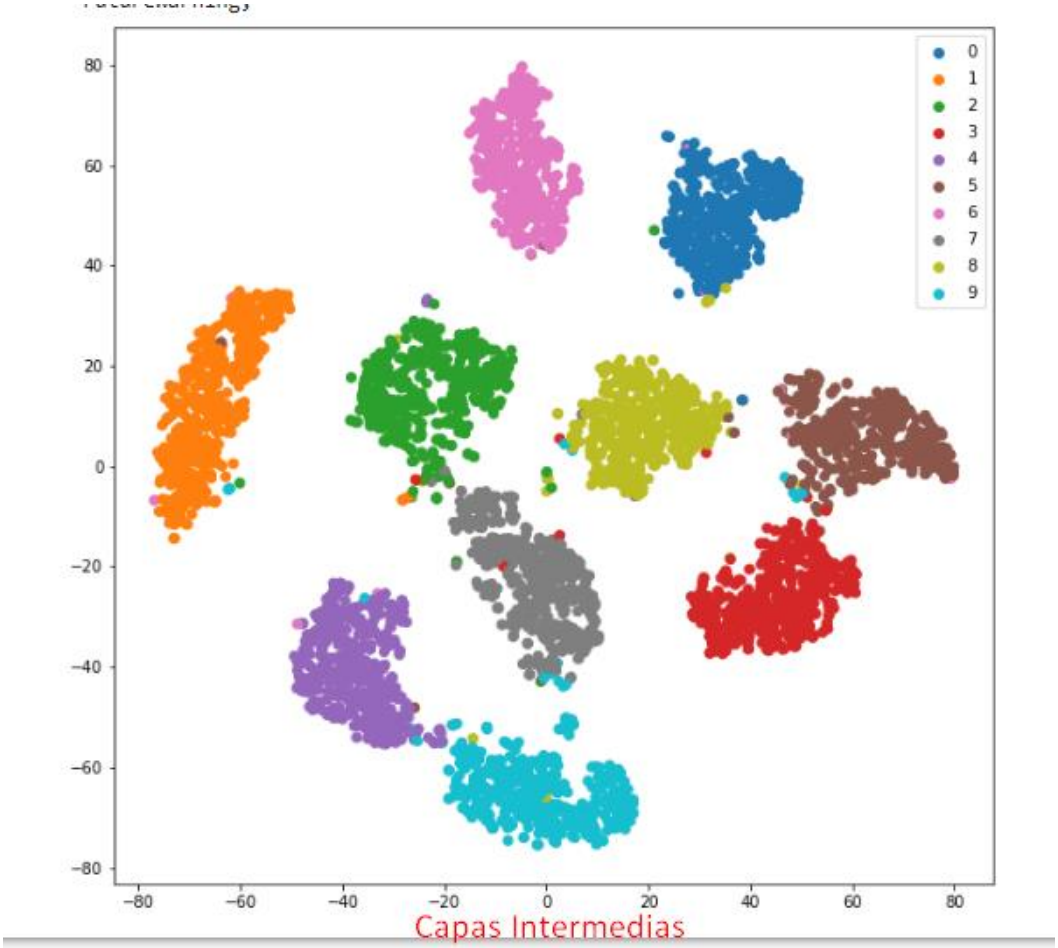


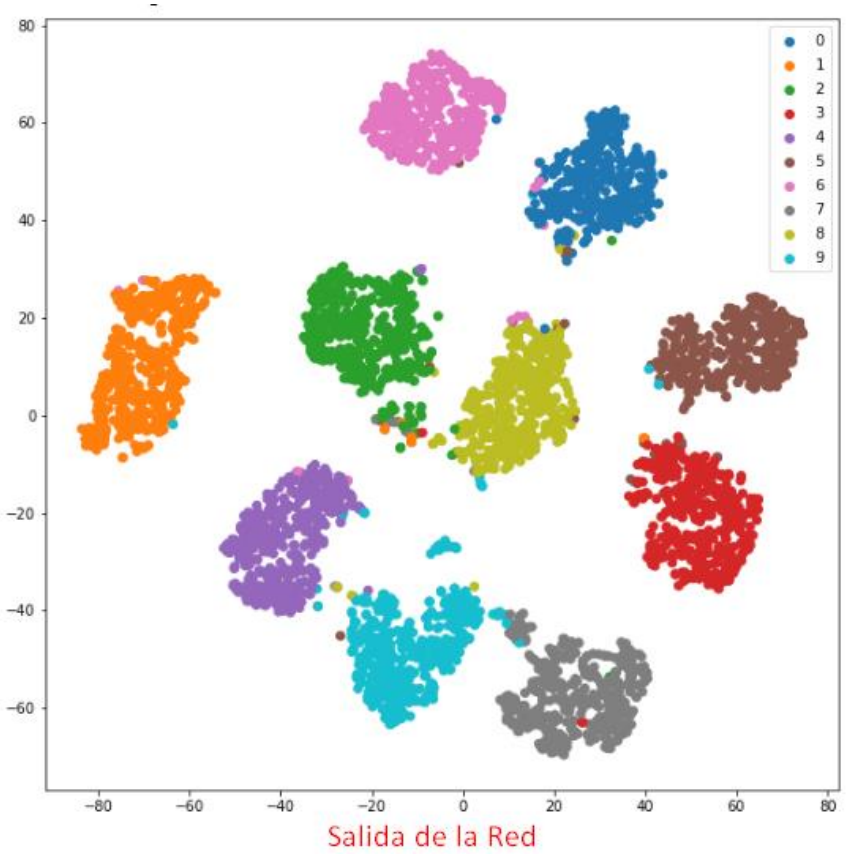
Vemos que el 7 real se confunde con el 2 en predicción (el caso de error más frecuente con 8 casos).
El segundo caso más frecuente(con 7 casos) 6 real se confunde con el 0 en predicción.

Se pueden ver estos casos en el gráfico siguiente:



- 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.





Observamos como ya en las capas intermedias disponemos de un cluster para cada dígito bien definido, lo que nos indica que a esas alturas de las capas la red ya dispone de buena información discrimitoria de los dígitos.

Sin embargo, vemos cierta tendencia al solapamiento entre algunos de ellos como el 7 y el 9 o el 3 y el 5. Para las capas finales, se puede ver que la dispersión intra-cluster es menor(los puntos del mismo color están mas cerca) y que la dispersión interclase es mayor también(es decir, que los clusters están mas separados entre ellos.) Lo que nos indica, que la salida de la red, es capaz de hacer una discriminación entre dígitos más fuerte y diferenciada que las capas intermedias.

- 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.

  Las primeras capas son las que extraen las características, en concreto, las dos capas de convoluciones(luego le palicamos max pooling). En la representación intermedia estamos representando la salida de primera capa fully conected, con lo que ya le hemos aplicados 2 capas convolucionales,un maxpooling y una capa densa con dropout.

  Dado que esta representación ya nos da unos clústeres bien diferenciados, una forma de reducir la complejidad de la red sin penalizar mucho la precisión podría ser quitar la ultima capa densa de la red, dejando únicamente la primera, solo que ésta con una dimensión de 10 en vez de 128. La razón de reducir la complejidad del modelo con estás capas, es que éstas se dedican a comprimir la información extraída, y podemos hacer una compresión un tanto más agresiva si

buscamos reducir la complejidad.

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**¡Error! No se encuentra el origen de la referencia.**

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

            # Second convolutional layer.
            nn.Conv2d(48, 128, 5, 1, 2), #in_channels, out_channels, kernel_size, stride, padding
            nn.MaxPool2d(2), #kernel_size
            nn.ReLU(inplace = True),

            # Third convolutional layer.
            nn.Conv2d(128, 192, 3, 1, 1), #in_channels, out_channels, kernel_size, stride, padding
            nn.ReLU(inplace = True),


            # 4 th convolutional layer.
            nn.Conv2d(192, 192, 3, 1, 1), #in_channels, out_channels, kernel_size, stride, padding
            nn.ReLU(inplace = True),


            # 5 th convolutional layer.
            nn.Conv2d(192, 128, 3, 1, 1), #in_channels, out_channels, kernel_size, stride, padding
            nn.MaxPool2d(2), #kernel_size
            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
```

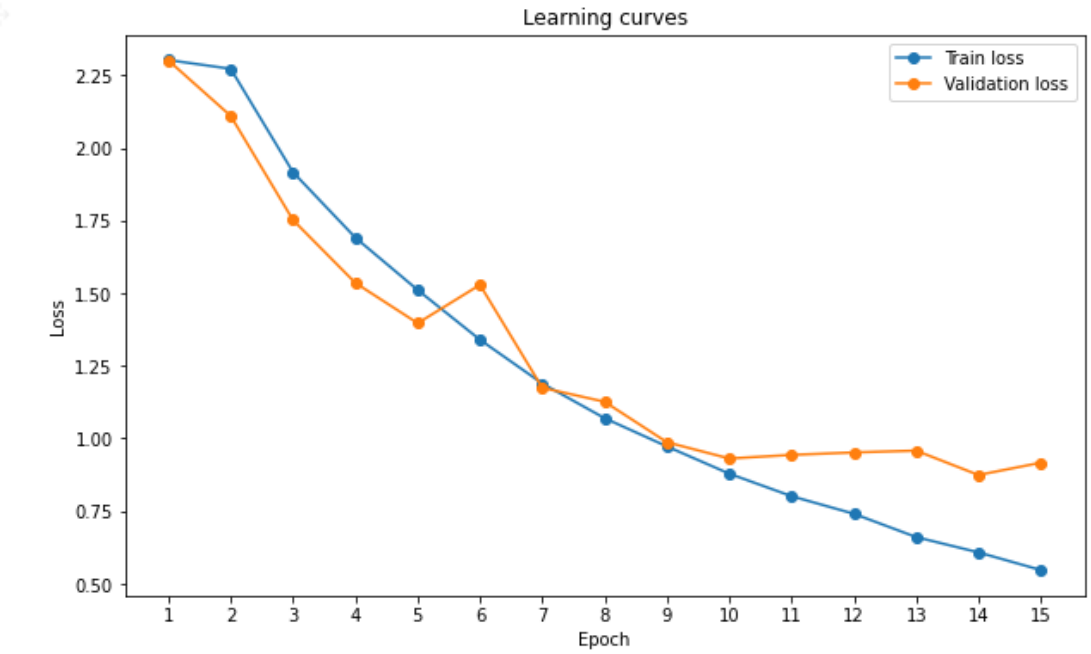
- Número de parámetros del modelo AlexNet

	Nº parámetros entrenables
AlexNet	6,199,498

De nuevo podemos ver este desglose por cada capa de la red:

```
+---------------------+------------+
|       Modules       | Parameters |
+---------------------+------------+
|  features.0.weight  |   23232    |
|   features.0.bias   |     64     |
|  features.3.weight  |   307200   |
|   features.3.bias   |    192     |
|  features.6.weight  |   663552   |
|   features.6.bias   |    384     |
|  features.8.weight  |   884736   |
|   features.8.bias   |    256     |
|  features.10.weight |   589824   |
|   features.10.bias  |    256     |
| classifier.1.weight |  37748736  |
|  classifier.1.bias  |    4096    |
| classifier.4.weight |  16777216  |
|  classifier.4.bias  |    4096    |
| classifier.6.weight |  4096000   |
|  classifier.6.bias  |    1000    |
+---------------------+------------+
Total Trainable Params: 61100840
61100840
```

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




```
[001] train loss: 2.302293  val loss: 2.300810  val acc: 10.0000%
[002] train loss: 2.272070  val loss: 2.109978  val acc: 18.4500%
[003] train loss: 1.916227  val loss: 1.751657  val acc: 30.7900%
[004] train loss: 1.691202  val loss: 1.535392  val acc: 41.8900%
[005] train loss: 1.511061  val loss: 1.396724  val acc: 47.7300%
[006] train loss: 1.339784  val loss: 1.528726  val acc: 44.3700%
[007] train loss: 1.189249  val loss: 1.174518  val acc: 58.0200%
[008] train loss: 1.069891  val loss: 1.126314  val acc: 60.4900%
[009] train loss: 0.972650  val loss: 0.987152  val acc: 64.1700%
[010] train loss: 0.878938  val loss: 0.931056  val acc: 67.0700%
[011] train loss: 0.801410  val loss: 0.943449  val acc: 67.5700%
[012] train loss: 0.740898  val loss: 0.951947  val acc: 66.8600%
[013] train loss: 0.660681  val loss: 0.958129  val acc: 68.9300%
[014] train loss: 0.607995  val loss: 0.874286  val acc: 70.9700%
[015] train loss: 0.547257  val loss: 0.916623  val acc: 71.2700%
```

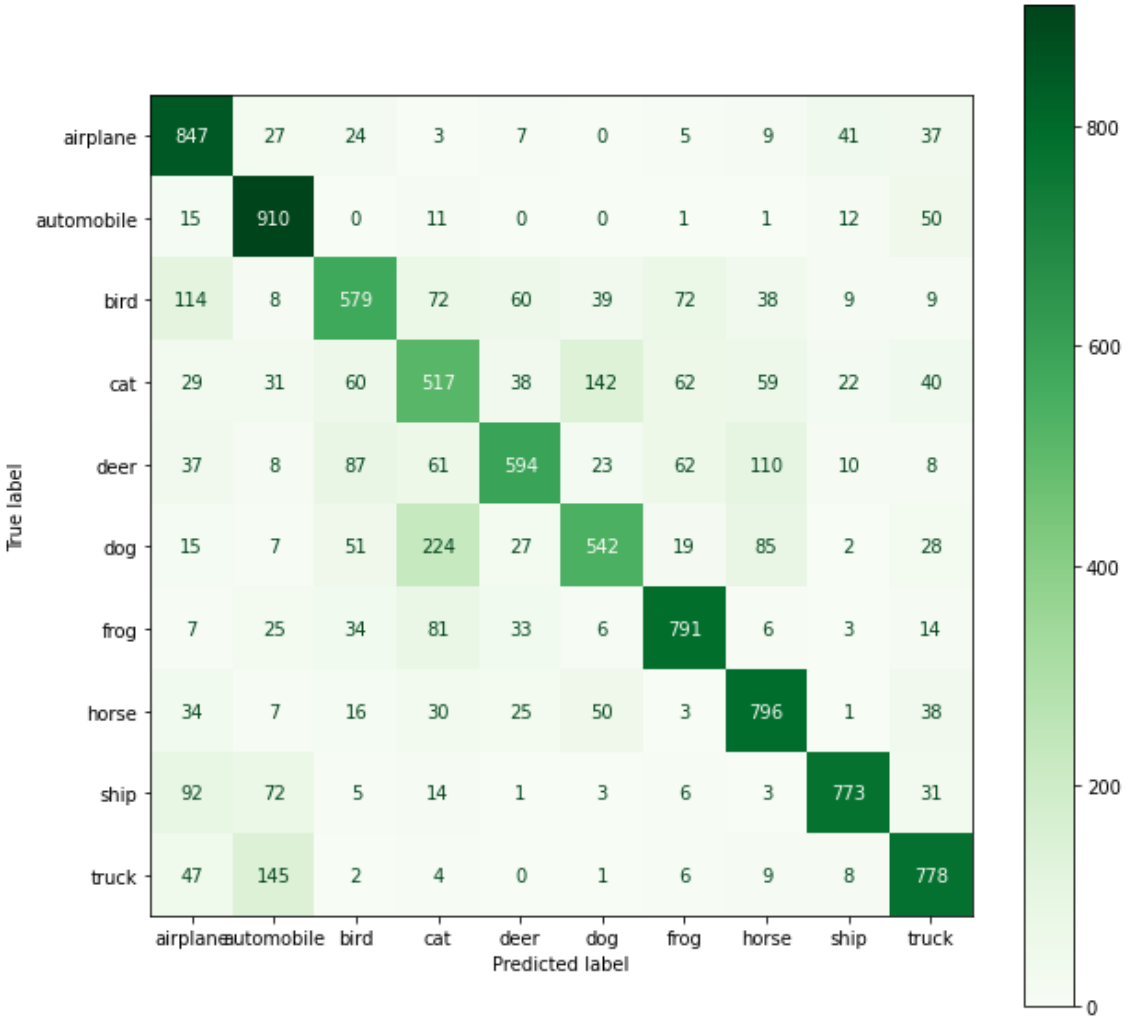
	Mejor precisión (validación)	Época con mejor precisión
AlexNet	71.2700%	15

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

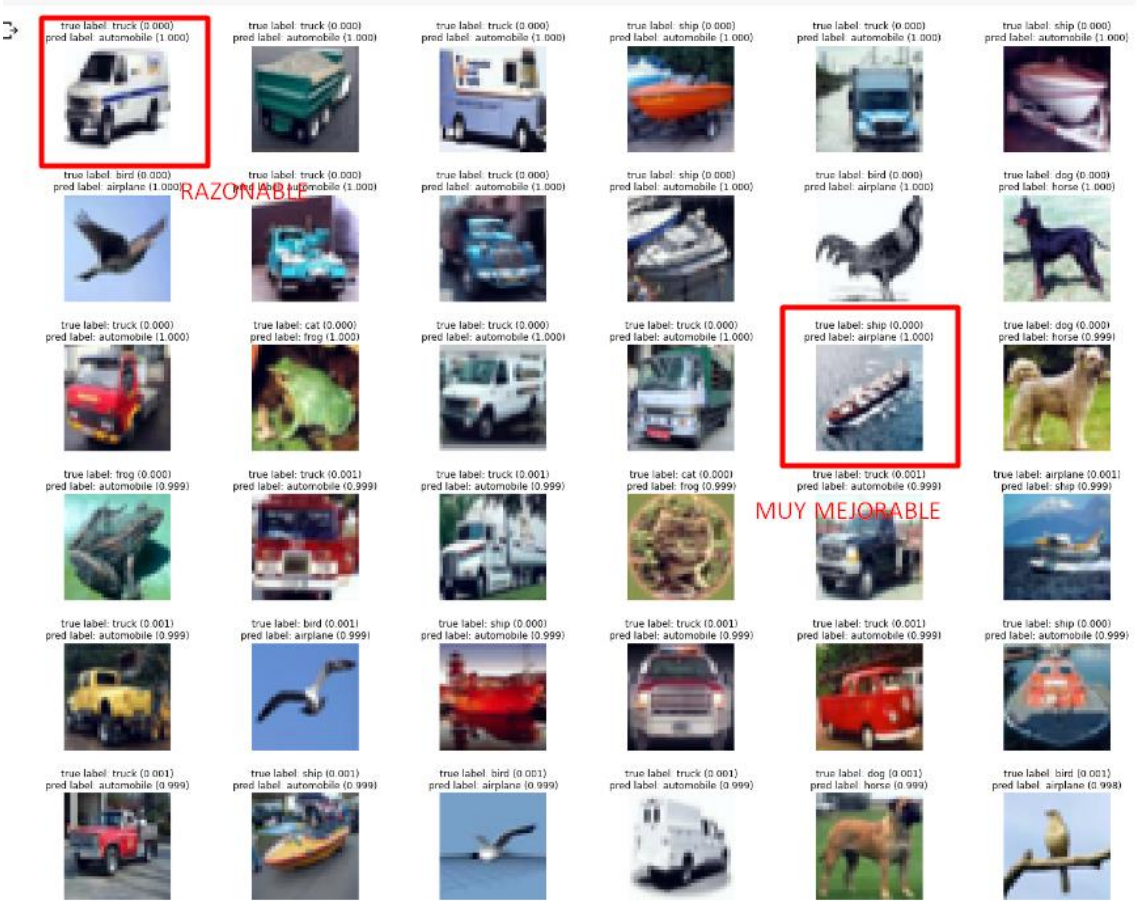
Vemos como en un principio las loss de training y validación decaen con el tiempo, lo cual es buena señal. Sin embargo, ya desde la época 6 la curva de validación empieza a querer estancarse, hasta que cuando llega a la época 10 y en adelante, vemos que empieza a mantenerse e incluso subir con las épocas, mientas la curva de training sigue disminuyendo. Es decir, en la época 10 no encontramos con un ***punto de retorno****, a partir del cual empezamos a obtener* ***overfitting****: la red captura bien los datos de entramiento, pero no los nuevos datos de validación, lo que quiere decir que ya no generaliza bien.*

Por lo tanto podemos afirmar que más épocas no mejorarían el rendimiento del modelo.

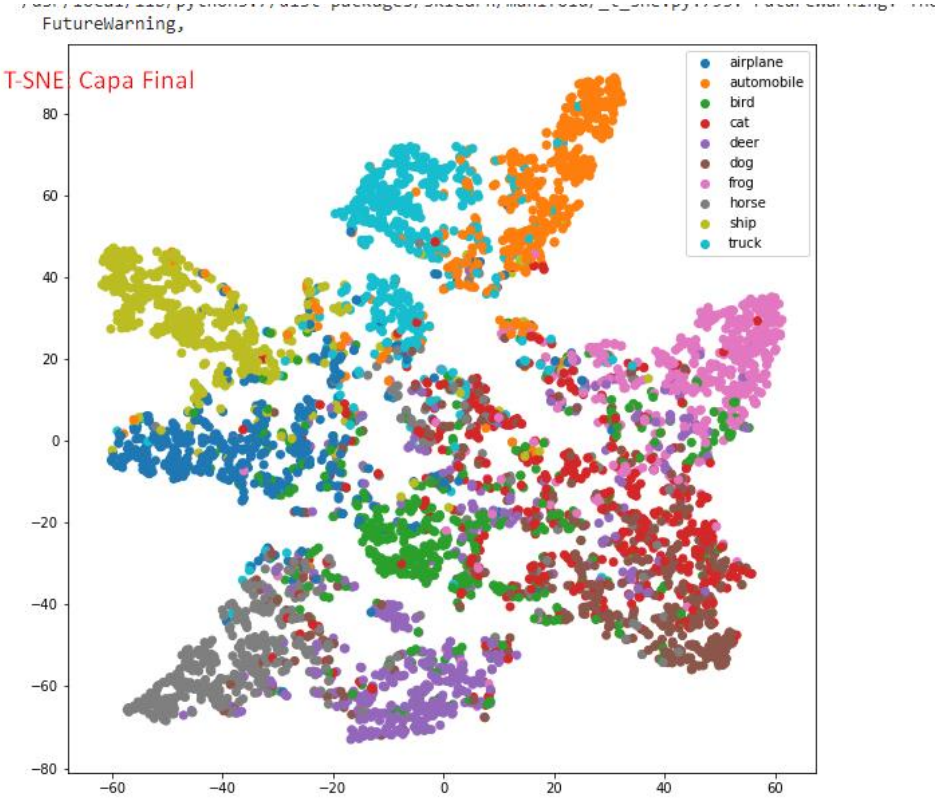
- Incluir la matriz de confusión. Comentar los resultados obtenidos atendiendo a las características de las imágenes de cada clase



Vemos que hay clases que confunde en la mayoría de casos son razonables, como dog y cat(la más frecuente con 224 casos), o truck y automobile, de acuerdo a la naturaleza de las mismas. Sin embargo si que es cierto que es bastante mejorable, tenemos el caso de 92 equivocaciones de ship con airplane, lo cual no guarda tanta similitud, por lo que sería recomendable mejor el modelo(ya sea con la arquitectura o con un dataset aún mayor).



- 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.



De entrada obsevamos que a diferencia del dataset de MNIST, aquí los clusterres están mas dispersos inter e intraclase. De hecho hay mucho solapamiento entre algunos de ellos. Por ejemplo, dog lo podemos encontrar disperso entre su cluster, el de cat y el de horse mayormente. Tiene sentido ya que en la vida real son los objetos que más similitud guardan entre ellos: todos tienen 4 patas, cola, hocico. En cambio vemos que como en la clase deear, que también cumple todas estas caráterísticas, no sucede lo mismo, muy probablemente porque estos disponen de cuernos, que es una característica discriminatoria muy fuerte y diferenciable. Además el contexto en el que probablemente encontremos horse y cat sea muy parecido al de dog: un césped, una granja,etc. Por lo que probablemente la red esté aprendiendo características del background de la imagen y clasificando en consecuencia.

Del mismo modo el background probablemente sea la característica que haga que ship,bird y plane tengan un valor alto en la matriz de confusión: un fondo azul haga que a veces el clasificador los confunda,ignorando el resto de características.

Los clústers comentados con relaciones de similitud por contexto o características comunes, se tienden a dispersar entre ellos como hemos visto.

Para mejorar el modelo necesitamos mejorar la información discriminatoria, intensificarla. Una forma de poder hacer esto sería hacer data augmentation: mantenemos el tamaño del datset, pero disponemos de más ejemplos para que la red mejore la clasificación. Aunque también existe la opción de variar parte(s) de la arquitectura de la red.

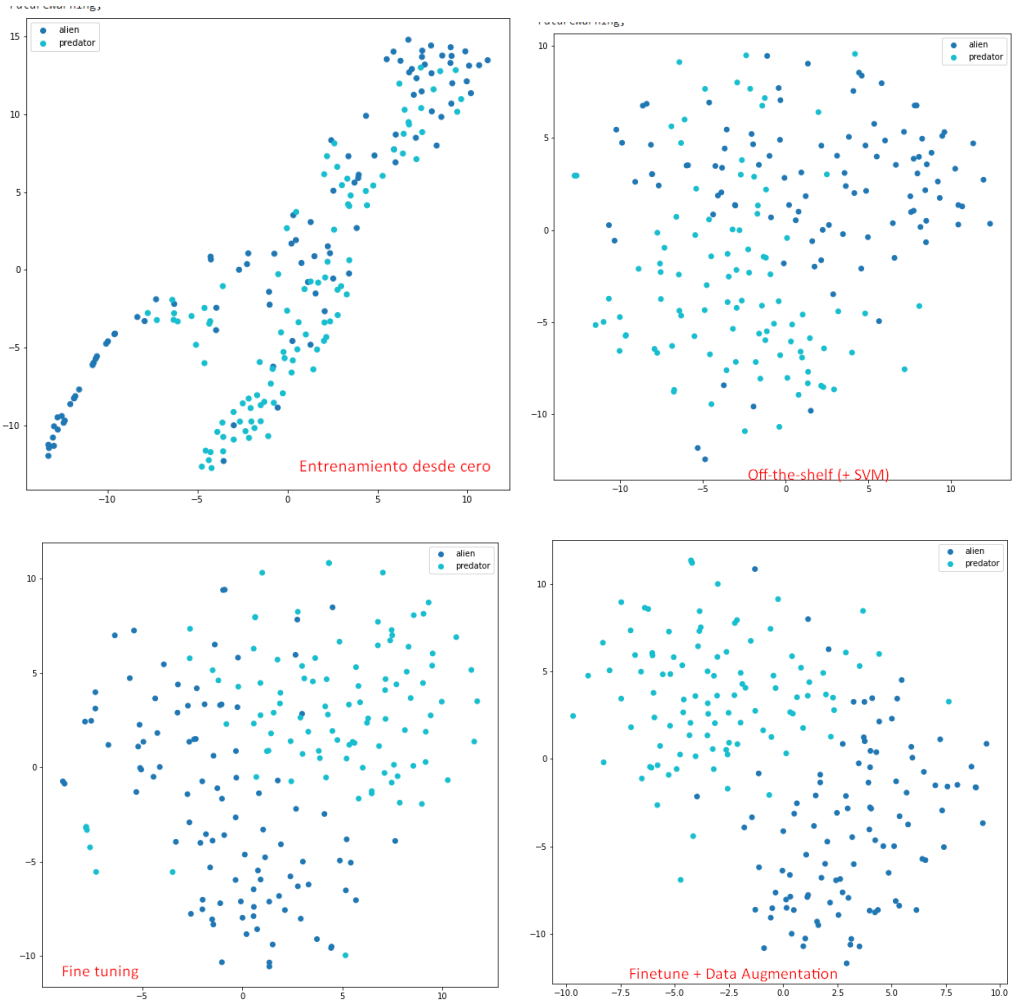
Puedes encontrar aquí el tercer *notebook.* Debe completar el código en el *notebook* y responder a las preguntas de la Sección 1.1.

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	72.00%	90.5%	92.50%	96.00%

- 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.



Para el método de entrenamiento desde cero vemos como hace la peor clasificación de clusters, ya que en la zona más baja la separación al principio es dispersa, pero enseguida se junta y se merclan demasiado los clusters. Por lo que la separación lineal hace que el porcentaje de muestras clasificadas erróneamente sea muy alto.

Para una cantidad de datos tan pequeña, vemos como la separación mejora enormemente cuando usamos preentrenamiento + calsificador lineal (y por tanto el accuracy, en un 20%). Esto es debido a que con unicamente los datos del datset, la red no aprende muy bien a extraer características.. Para estos 3 casos vemos como disponemos de una nube de puntos Tambien linealmente separable, pero mucho mayor.

Entre usar una SVM y una capa lineal (nn.Linear), vemos como obtenermos un resultado de accuracy y representación similares. Aunque ligeramente mejor con la capa linear(gráfica 3 de fine tuning). Con ambas la clasificación lineal errónea de muestras es acceptable pero mejorable.

Si a este mejor clasificador, le proporcionamos más datos, en este caso del mismo datset, mediante data augmentation, vemos como el accuracy crece un 4%. La representción y diferenciacion de los clusters también se hace muchos más clara, con ambos clusters más disperses entre sí, y permitiendo poder separar las clases linealmente de una forma muy eficiente.

Por ello como vemos la combinación de Fine Tuning + Data Augmentation, es la mejor opción para tener el menor porcentaje de muestras erróneamente clasificadas mediante separación lineal de las 2 clases.