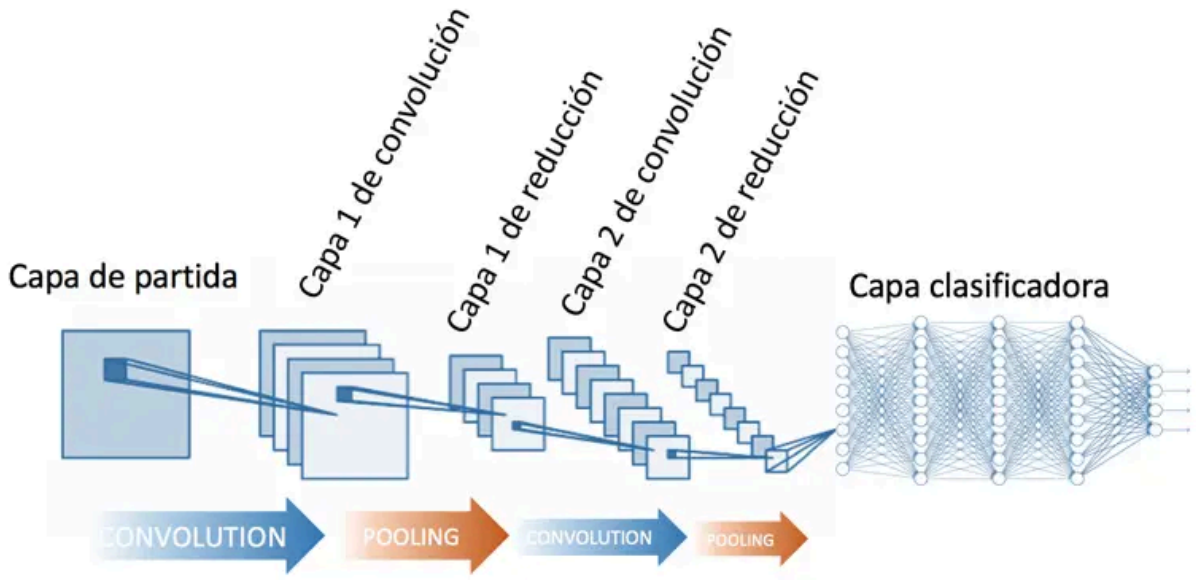
Práctica 1: Clasificador de imágenes con redes convolutivas



Asignatura: Aprendizaje Automático 2
Curso: 2024/25
Institución: EII-ULPGC

Autores:
Luis Perera Pérez
Nicolás Trujillo Estévez

Índice:

1. Introducción

Las redes neuronales convolutivas (CNN, por sus siglas en inglés) han sido un pilar fundamental en el campo del aprendizaje automático, especialmente en tareas de clasificación de imágenes. Gracias a su capacidad para capturar características jerárquicas en los datos visuales, las CNN han revolucionado áreas como la visión por computadora, desde el reconocimiento de objetos hasta la segmentación de imágenes. Este tipo de red es particularmente eficaz para procesar datos que tienen una estructura de cuadrícula, como las imágenes.

El objetivo de esta práctica es desarrollar una red neuronal convolutiva utilizando PyTorch para la clasificación de un conjunto de datos de imágenes. A lo largo de este proyecto, se explorarán diferentes configuraciones de la red, ajustando los hiperparámetros para obtener el mejor rendimiento posible. Además como aplicación especial añadida, se implementará el uso de transfer learning (aprendizaje por transferencia), aprovechando una seccion de la vgg-16 para comparar los resultados del aprendizaje por transferencia con los de la red neuronal planteada por nosotros.

2. Objetivos

El objetivo principal de esta práctica es diseñar y entrenar una red neuronal convolutiva para la clasificación de imágenes. Para ello, se deben alcanzar los siguientes objetivos específicos:

- **Selección y Preparación del Conjunto de Datos**: Elegir un conjunto de datos adecuado para el problema de clasificación, que cuente con al menos 5 categorías y 50 imágenes por categoría.
- **Implementación de la Red Neuronal Convolutiva**: Crear una red neuronal convolutiva en PyTorch, adaptándola al conjunto de datos elegido y utilizando arquitecturas adecuadas.
- **Búsqueda de Hiperparámetros**: Realizar una búsqueda exhaustiva de los hiperparámetros más adecuados, como el número de capas, el tamaño de los filtros, la regularización (como dropout), los optimizadores, entre otros, con el fin de obtener el mejor rendimiento del modelo. Primero a mano y de manera empírica y luego por medio de una libreria que se encarga de la seleccion aleatoria de esos hiperparametros.
- **Uso de Transfer Learning**: Implementar y evaluar el uso de modelos preentrenados (vgg-16) buscand mejorar la clasificación, aplicando la técnica de transfer learning.
- **Evaluación del Rendimiento del Modelo**: Evaluar el rendimiento de la red mediante métricas clave como la precisión y las curvas de aprendizaje (pérdida y precisión).

3. Metodología

3.1. Dataset

Para esta práctica se ha elegido un conjunto de datos compuesto por fotografías de diversas banderas nacionales. Este conjunto de datos contiene un total de 24 clases, correspondientes a banderas de países de Europa. Las clases son las siguientes: Austria, Belgium, Bulgaria, Croatia, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Holland, Hungary, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Slovakia, Slovenia, South Cyprus, Spain y Sweden.

El tamaño del conjunto de entrenamiento es de 796 imágenes, mientras que el conjunto de prueba consta de 200 imágenes. Las imágenes están etiquetadas según el país que representa la bandera, lo que permite realizar una tarea de clasificación supervisada. Este dataset es adecuado para poner a prueba la capacidad de una red neuronal convolutiva en la clasificación de imágenes de distintos patrones visuales puesto que cuenta con diferentes tipos de imagenes desde muchos angulos y no las típicas imagenes bien encuadradas.

3.2. Modelo

```python
class ConvNet(nn.Module):
    def __init__(self, num_filters_conv1=16, num_filters_conv2=32, kernel_size_conv1=5, kernel_size_conv2=5,
                 fc1_units=120, fc2_units=24, dropout_rate=0.0, activation=nn.ReLU):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, num_filters_conv1, kernel_size=kernel_size_conv1, stride=1, padding=kernel_size_conv1 // 2)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv2d(num_filters_conv1, num_filters_conv2, kernel_size=kernel_size_conv2, stride=1, padding=kernel_size_conv2 // 2)

        self.fc1 = nn.Linear(num_filters_conv2 * 32 * 32, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.dropout = nn.Dropout(dropout_rate)
        self.activation = activation()  # Instancia de la función de activación

    def forward(self, x):
        x = self.pool(self.activation(self.conv1(x)))
        x = self.pool(self.activation(self.conv2(x)))
        x = x.view(-1, x.size(1) * x.size(2) * x.size(3))  # Aplanar
        x = self.dropout(self.activation(self.fc1(x)))
        x = self.fc2(x)
        return x
```

Planteamos un modelo basico de red convolucional a partir de un codigo de partida con dos capas convolucionales, un maxpooling y dos capas fully connected. Por consiguiente probamos cambiando a mano algunos hiperparametros como el ratio de aprendizaje, la funcion de activacion, el optimizador optimizado y el numero de neuronas de las capas lineales y nos dimos cuenta que los hiperparametros del codigo de partida daban mas que buenos resultados en un entrenamiento basico con pocas epocas y es que sin cambiar numeros de neuronas predeterminado y eligiendo un funcion de activacion entre relu y leaky relu (para evitar neuronas muertas) y eligiendo optimizadores con utilizacion de momentum (termino de inercia que suavizara la curva de aprendizaje) y un ratio de aprendizaje pequeño entre 0.001 y 0.00001 el modelo ya se comportaba super bien, resultados que corroboraremos mas adelante con la implementacion de optuna.

3.3. Hiperparametros

La implementacion de optuna esta en el codigo pero aqui disponemos un resumen de los hiperparametros que se tratan y un ejemplo resultado para no tener que proceder a su ejecución si no se requiere.

Hiperparámetros del modelo

1. **`num_filters_conv1`**
   - Tipo: Entero (8 a 64, con pasos de 8)
   - Descripción: Número de filtros para la primera capa convolucional.
2. **`num_filters_conv2`**
   - Tipo: Entero (16 a 128, con pasos de 16)
   - Descripción: Número de filtros para la segunda capa convolucional.
3. **`kernel_size_conv1`**
   - Tipo: Entero (3 a 7, con pasos de 2)
   - Descripción: Tamaño del kernel (filtro) de la primera capa convolucional.
4. **`kernel_size_conv2`**
   - Tipo: Entero (3 a 7, con pasos de 2)
   - Descripción: Tamaño del kernel (filtro) de la segunda capa convolucional.
5. **`fc1_units`**
   - Tipo: Entero (64 a 512, con pasos de 64)
   - Descripción: Número de unidades (neuronas) en la primera capa completamente conectada (fully connected).
6. **`dropout_rate`**
   - Tipo: Float (0.0 a 0.5)
   - Descripción: Tasa de Dropout utilizada para regularización, ayuda a prevenir el sobreajuste.
7. **`activation`**
   - Tipo: Categórico ([`ReLU`, `LeakyReLU`, `ELU`, `Sigmoid`, `Tanh`])
   - Descripción: Función de activación utilizada en las capas.

Hiperparámetros del optimizador

8. **`lr`**
   - Tipo: Log-uniforme (0.00001 a 0.001)
   - Descripción: Learning rate o tasa de aprendizaje del optimizador.
9. **`optimizer`**
   - Tipo: Categórico ([`Adam`, `SGD`, `RMSprop`, `Adadelta`, `Adagrad`, `ASGD`, `NAdam`])
   - Descripción: Algoritmo de optimización para actualizar los pesos del modelo.

Hiperparámetros de entrenamiento

10. **`batch_size`**
    - Tipo: Entero (16 a 128, con pasos de 16)
    - Descripción: Tamaño del lote de datos procesados en cada iteración de entrenamiento.

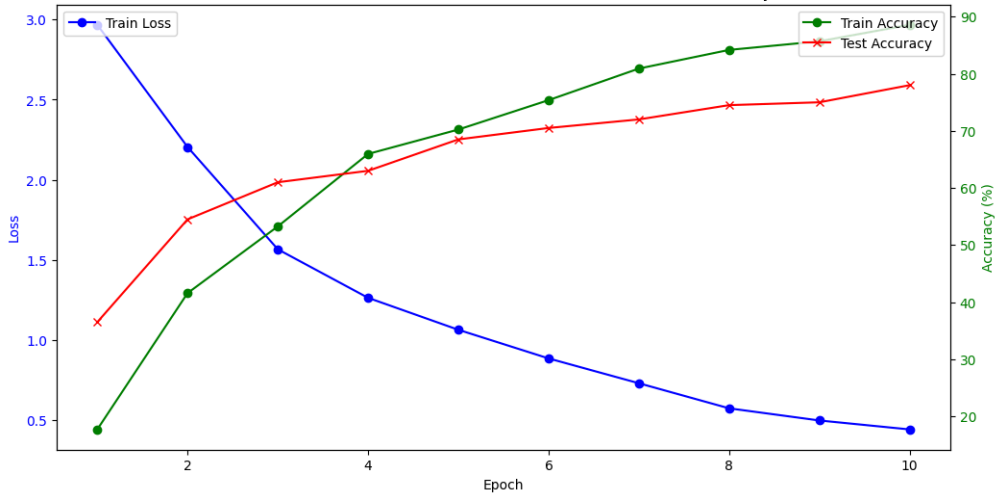
Constante del entrenamiento

11. **`epochs`**
    - Tipo: Entero (Constante, 5 en este caso)
    - Descripción: Número de veces que el modelo verá el conjunto de datos completo durante el entrenamiento.

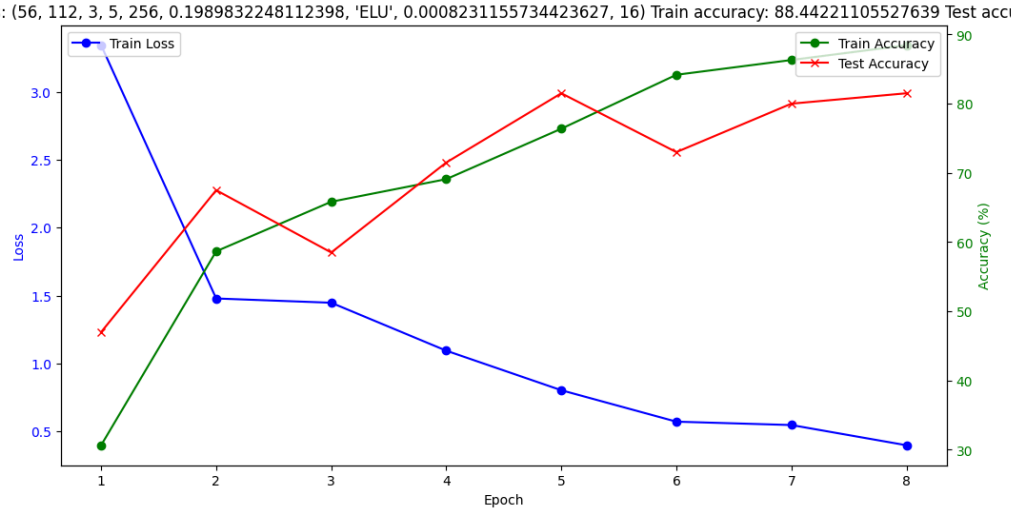
3.4. Entrenamiento y graficacion

Posterior a la obtencion aleatoria de los hiperparametros se procede al entrenamiento, testeo y graficacion del modelo con los hiperparametros utilizando early stop, los resultados se disponen aqui por si no se quiere proceder a su ejecución:

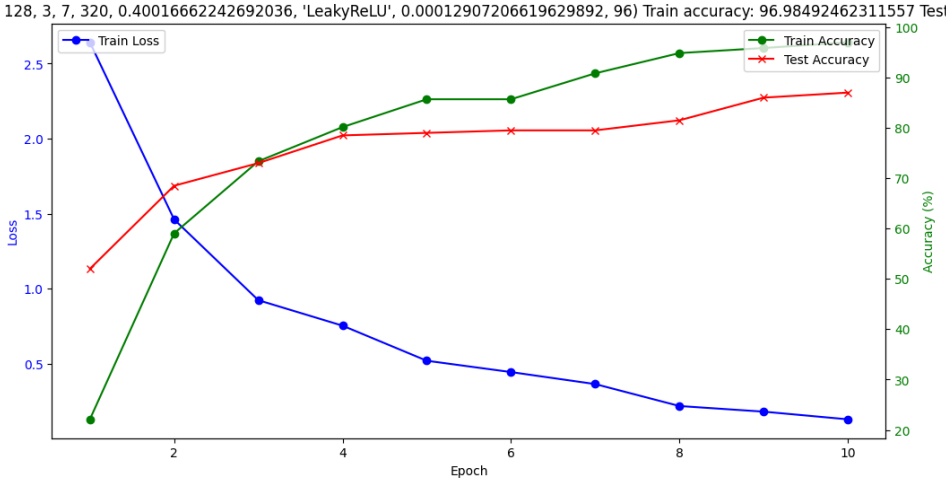
Parametros: (16, 48, 7, 5, 128, 0.2171782483869571, 'ReLU', 0.0001367225863360829, 96) Train accuracy: 88.5678391959799 Test accuracy: 78.0



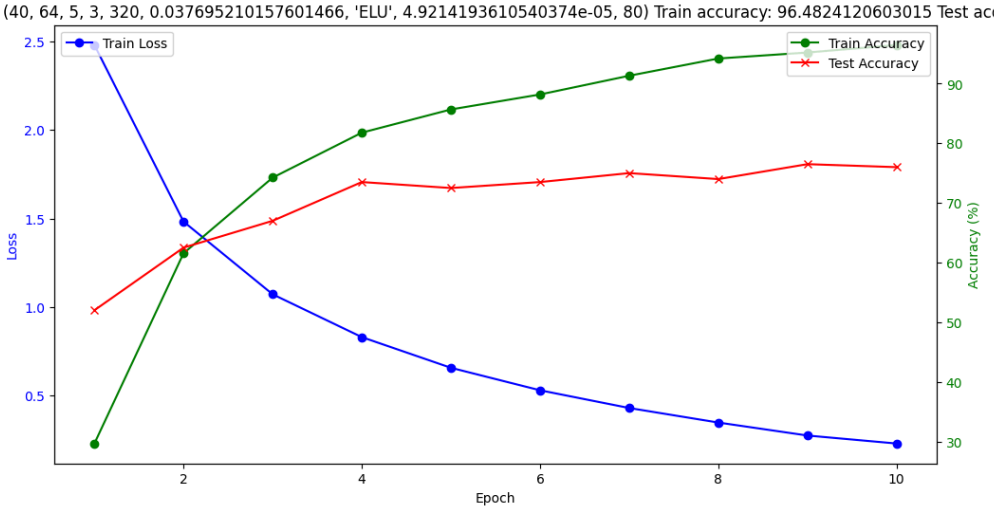
Parametros: (56, 112, 3, 5, 256, 0.1989832248112398, 'ELU', 0.0008231155734423627, 16) Train accuracy: 88.44221105527639 Test accuracy: 81.5



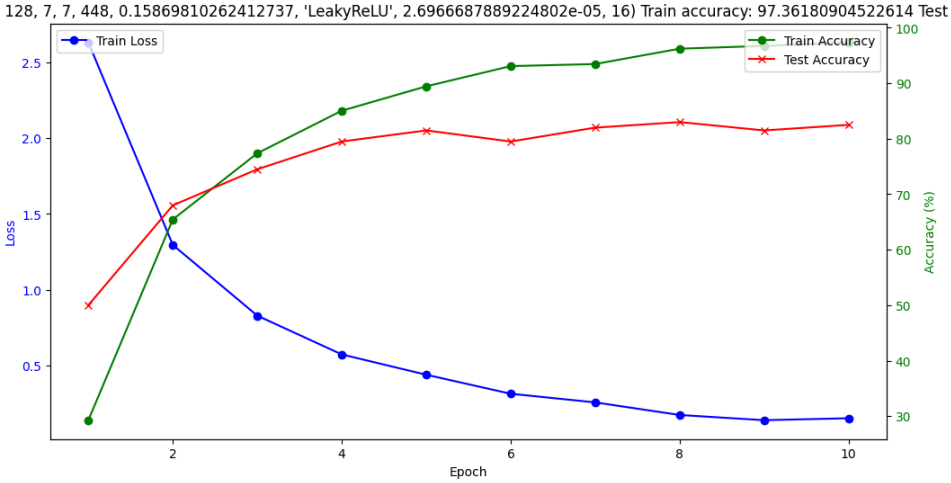
Parametros: (48, 128, 3, 7, 320, 0.40016662242692036, 'LeakyReLU', 0.00012907206619629892, 96) Train accuracy: 96.98492462311557 Test accuracy: 87.0



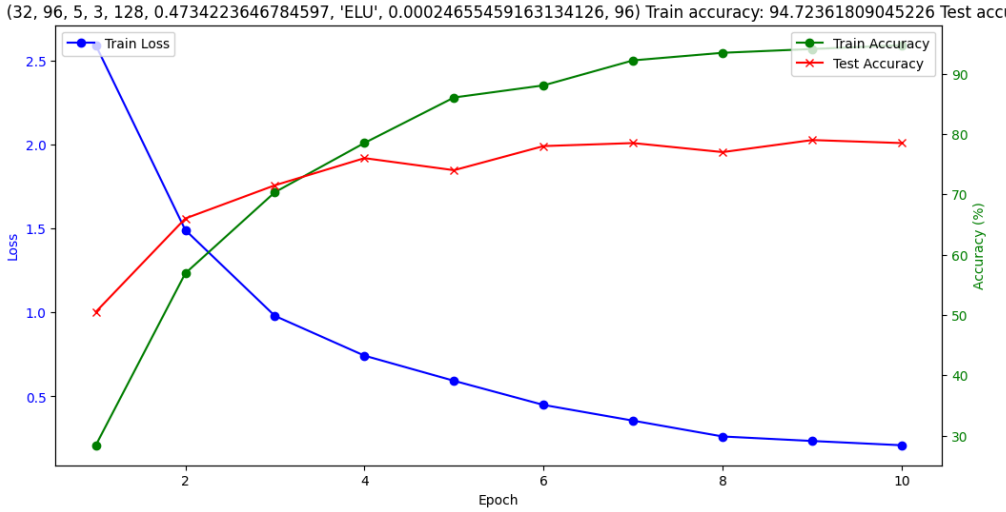
Parametros: (40, 64, 5, 3, 320, 0.037695210157601466, 'ELU', 4.9214193610540374e-05, 80) Train accuracy: 96.4824120603015 Test accuracy: 76.0



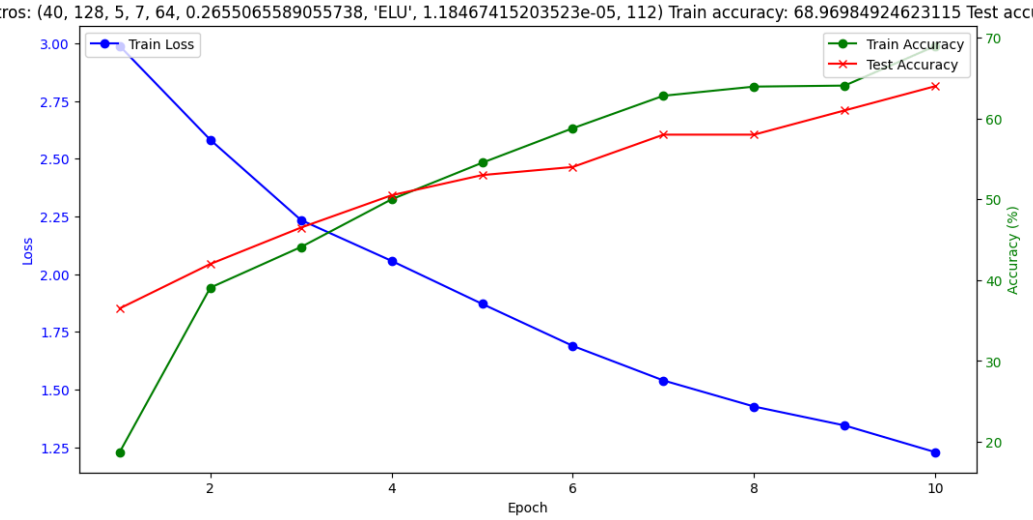
Parametros: (32, 128, 7, 7, 448, 0.15869810262412737, 'LeakyReLU', 2.6966687889224802e-05, 16) Train accuracy: 97.36180904522614 Test accuracy: 82.5



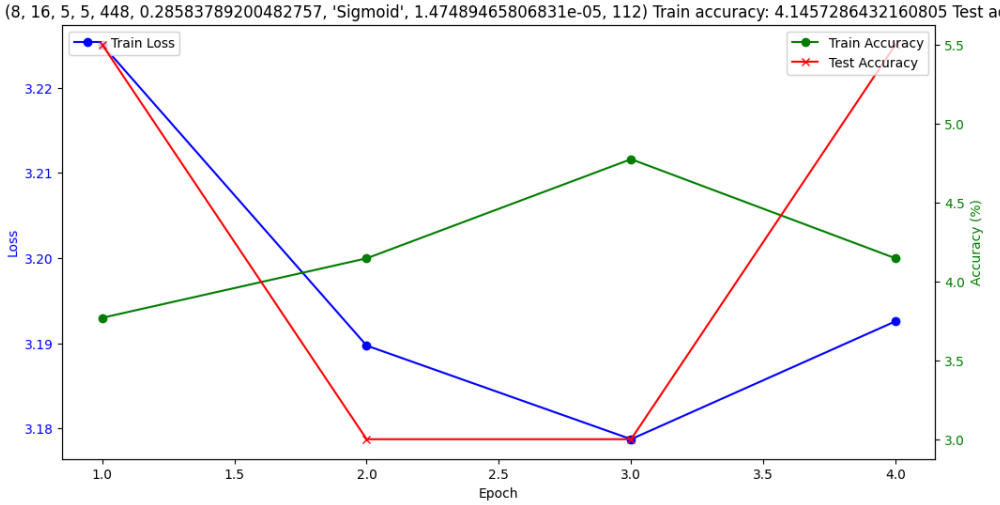
Parametros: (32, 96, 5, 3, 128, 0.4734223646784597, 'ELU', 0.00024655459163134126, 96) Train accuracy: 94.72361809045226 Test accuracy: 78.5



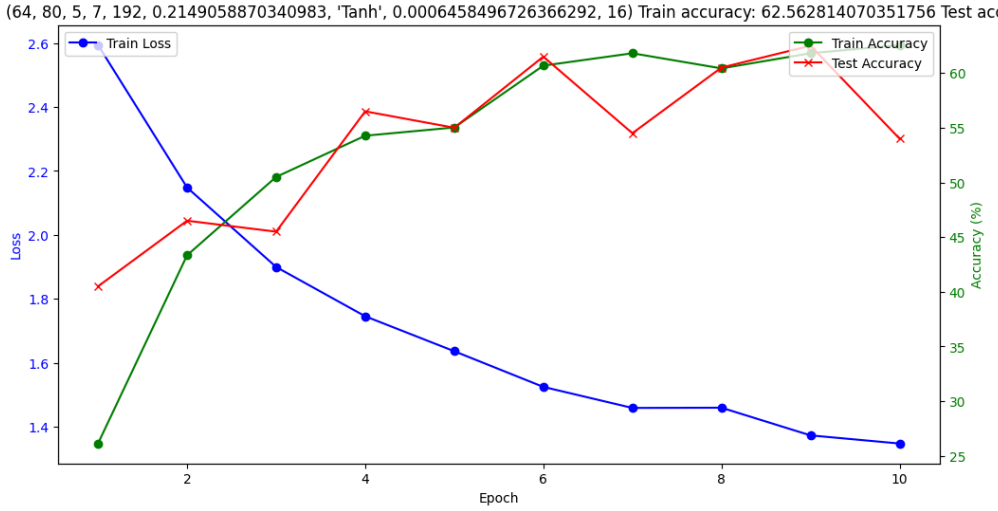
Parametros: (40, 128, 5, 7, 64, 0.2655065589055738, 'ELU', 1.18467415203523e-05, 112) Train accuracy: 68.96984924623115 Test accuracy: 64.0



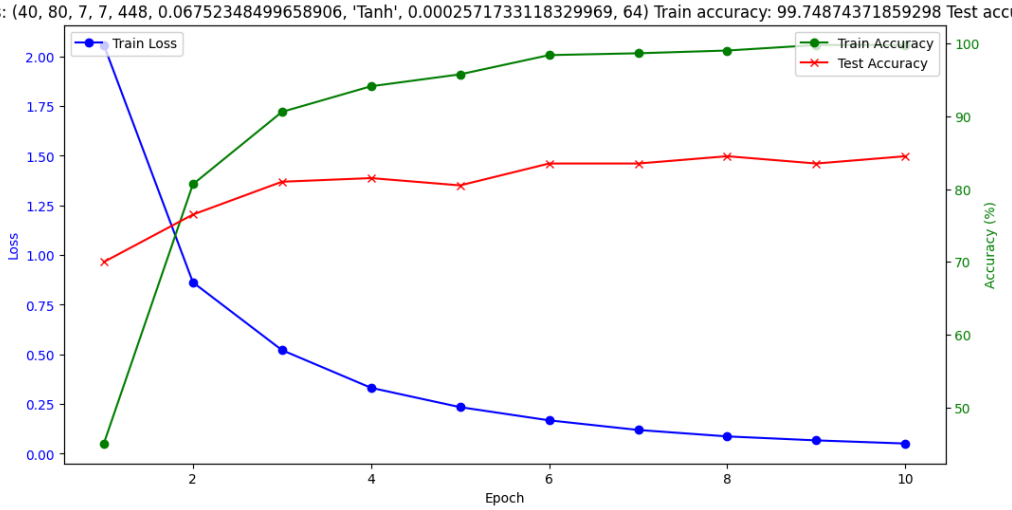
Parametros: (8, 16, 5, 5, 448, 0.28583789200482757, 'Sigmoid', 1.47489465806831e-05, 112) Train accuracy: 4.1457286432160805 Test accuracy: 5.5



Parametros: (64, 80, 5, 7, 192, 0.2149058870340983, 'Tanh', 0.0006458496726366292, 16) Train accuracy: 62.562814070351756 Test accuracy: 54.0



Parametros: (40, 80, 7, 7, 448, 0.06752348499658906, 'Tanh', 0.0002571733118329969, 64) Train accuracy: 99.74874371859298 Test accuracy: 84.5



3.5. Aprendizaje por transferencia

Esta parte del trabajo se puede clasificar como simbolica puesto que por falta de tiempo no indagamos mucho pero nos pareció raro observar que tenia peores resultados que el modelo nuestros a la hora de testear consideramos que es porque ha habido un sobre ajuste en el entrenamiento.

Epoch 1/5, Loss: 3.2025, Accuracy: 17.71%
Epoch 2/5, Loss: 1.6907, Accuracy: 52.26%
Epoch 3/5, Loss: 1.2454, Accuracy: 65.83%
Epoch 4/5, Loss: 0.9740, Accuracy: 74.37%
Epoch 5/5, Loss: 0.8100, Accuracy: 79.02%
Test Accuracy: 51.00%

4. Resultados

Ejemplo resultado del entrenamiento basico para no tener que proceder a su ejecucion:

Epoch 1, Loss: 2.2154, Accuracy: 34.42%

Epoch 2, Loss: 0.8506, Accuracy: 75.25%
Epoch 3, Loss: 0.4617, Accuracy: 87.69%
Epoch 4, Loss: 0.3009, Accuracy: 91.96%
Epoch 5, Loss: 0.1536, Accuracy: 95.98%

Ejemplo resultado del testeo basico para no tener que proceder a su ejecucion:

Accuracy of the model: 82.0%

Ejemplo resultado de optuna para no tener que proceder a su ejecucion:

[I 2024-12-02 17:35:58,622] Trial 0 finished with value: 0.865 and parameters: {'num_filters_conv1': 24, 'num_filters_conv2': 96, 'kernel_size_conv1': 5, 'kernel_size_conv2': 5, 'fc1_units': 384, 'dropout_rate': 0.023528276899695444, 'activation': <class 'torch.nn.modules.activation.LeakyReLU'>, 'lr': 0.00017908960602750068, 'optimizer': 'NAdam', 'batch_size': 16}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:36:54,623] Trial 1 finished with value: 0.34 and parameters: {'num_filters_conv1': 48, 'num_filters_conv2': 48, 'kernel_size_conv1': 3, 'kernel_size_conv2': 3, 'fc1_units': 64, 'dropout_rate': 0.139046109074281, 'activation': <class 'torch.nn.modules.activation.ReLU'>, 'lr': 3.7301156970022335e-05, 'optimizer': 'NAdam', 'batch_size': 128}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:38:09,402] Trial 2 finished with value: 0.045 and parameters: {'num_filters_conv1': 64, 'num_filters_conv2': 32, 'kernel_size_conv1': 3, 'kernel_size_conv2': 5, 'fc1_units': 448, 'dropout_rate': 0.023738204059223555, 'activation': <class 'torch.nn.modules.activation.ELU'>, 'lr': 3.294818804734115e-05, 'optimizer': 'Adadelta', 'batch_size': 96}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:41:11,023] Trial 3 finished with value: 0.83 and parameters: {'num_filters_conv1': 24, 'num_filters_conv2': 112, 'kernel_size_conv1': 5, 'kernel_size_conv2': 5, 'fc1_units': 448, 'dropout_rate': 0.018649782725007058, 'activation': <class 'torch.nn.modules.activation.LeakyReLU'>, 'lr': 7.649764783102111e-05, 'optimizer': 'NAdam', 'batch_size': 16}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:42:35,755] Trial 4 finished with value: 0.03 and parameters: {'num_filters_conv1': 16, 'num_filters_conv2': 80, 'kernel_size_conv1': 3, 'kernel_size_conv2': 5, 'fc1_units': 512, 'dropout_rate': 0.45175556968336106, 'activation': <class 'torch.nn.modules.activation.ReLU'>, 'lr': 1.7957831800836656e-05, 'optimizer': 'ASGD', 'batch_size': 32}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:43:43,132] Trial 5 finished with value: 0.065 and parameters: {'num_filters_conv1': 56, 'num_filters_conv2': 16, 'kernel_size_conv1': 5, 'kernel_size_conv2': 5, 'fc1_units': 512, 'dropout_rate': 0.4443473021750981, 'activation': <class 'torch.nn.modules.activation.ReLU'>, 'lr': 0.0003606253250741577, 'optimizer': 'SGD', 'batch_size': 48}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:45:50,486] Trial 6 finished with value: 0.75 and parameters: {'num_filters_conv1': 8, 'num_filters_conv2': 64, 'kernel_size_conv1': 3, 'kernel_size_conv2':

3, 'fc1_units': 512, 'dropout_rate': 0.37368966204101295, 'activation': <class 'torch.nn.modules.activation.ELU'>, 'lr': 6.076991430560391e-05, 'optimizer': 'NAdam', 'batch_size': 16}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:47:05,357] Trial 7 finished with value: 0.045 and parameters: {'num_filters_conv1': 64, 'num_filters_conv2': 32, 'kernel_size_conv1': 5, 'kernel_size_conv2': 5, 'fc1_units': 320, 'dropout_rate': 0.44788959927280814, 'activation': <class 'torch.nn.modules.activation.Tanh'>, 'lr': 1.9215908923392973e-05, 'optimizer': 'ASGD', 'batch_size': 128}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:47:53,031] Trial 8 finished with value: 0.065 and parameters: {'num_filters_conv1': 8, 'num_filters_conv2': 48, 'kernel_size_conv1': 5, 'kernel_size_conv2': 5, 'fc1_units': 256, 'dropout_rate': 0.29740269367268685, 'activation': <class 'torch.nn.modules.activation.Sigmoid'>, 'lr': 2.4710158117383795e-05, 'optimizer': 'NAdam', 'batch_size': 32}. Best is trial 0 with value: 0.865.

[I 2024-12-02 17:48:38,601] Trial 9 finished with value: 0.65 and parameters: {'num_filters_conv1': 8, 'num_filters_conv2': 64, 'kernel_size_conv1': 7, 'kernel_size_conv2': 5, 'fc1_units': 192, 'dropout_rate': 0.31135295165888865, 'activation': <class 'torch.nn.modules.activation.LeakyReLU'>, 'lr': 2.804932600022528e-05, 'optimizer': 'RMSprop', 'batch_size': 96}. Best is trial 0 with value: 0.865.

Resultados del entrenamiento del modelo con los hiperparametros (Se despliega uno par ano ocupar el documento con contenido reiterado y pesado de observar):

Epoch 1/10, Loss: 2.9667, Train Accuracy: 17.71%, Test Accuracy: 36.50%

Epoch 2/10, Loss: 2.2046, Train Accuracy: 41.58%, Test Accuracy: 54.50%

Epoch 3/10, Loss: 1.5665, Train Accuracy: 53.27%, Test Accuracy: 61.00%

Epoch 4/10, Loss: 1.2639, Train Accuracy: 65.95%, Test Accuracy: 63.00%

Epoch 5/10, Loss: 1.0646, Train Accuracy: 70.23%, Test Accuracy: 68.50%

Epoch 6/10, Loss: 0.8859, Train Accuracy: 75.38%, Test Accuracy: 70.50%

Epoch 7/10, Loss: 0.7311, Train Accuracy: 80.90%, Test Accuracy: 72.00%

Epoch 8/10, Loss: 0.5747, Train Accuracy: 84.17%, Test Accuracy: 74.50%

Epoch 9/10, Loss: 0.4989, Train Accuracy: 85.68%, Test Accuracy: 75.00%

Epoch 10/10, Loss: 0.4431, Train Accuracy: 88.57%, Test Accuracy: 78.00%

5. Conclusión

Hiperparámetros más relevantes para la precisión

1. `num_filters_conv1` **(Número de filtros en la primera capa convolucional):**
   - Valores altos (24) parecen funcionar mejor, como en el Trial 0 (precisión: 0.865).
   - Valores bajos (8) son menos efectivos en general, como en el Trial 9 (precisión: 0.65).
2. `num_filters_conv2` **(Número de filtros en la segunda capa convolucional):**
   - Valores intermedios a altos (96, 112) tienden a obtener mejores resultados, como en los Trials 0 y 3.
3. `kernel_size_conv1` **y** `kernel_size_conv2` **(Tamaño del kernel de las capas convolucionales):**
   - Tamaños de kernel medianos (5x5) tienen mejor rendimiento, como en los Trials 0 y 3.
   - Tamaños pequeños (3x3) no funcionan tan bien en la mayoría de los casos.
4. `fc1_units` **(Número de unidades en la capa totalmente conectada):**
   - Valores intermedios a altos (384, 448) ofrecen mejores resultados (Trials 0 y 3).
   - Valores más bajos (64, 320) reducen significativamente la precisión.
5. `dropout_rate` **(Tasa de dropout):**
   - Tasas bajas (0.018 a 0.023) son más efectivas, como en los Trials 0 y 3.
   - Tasas altas (0.44 y 0.45) disminuyen la precisión.
6. `activation` **(Función de activación):**
   - La función `LeakyReLU` se destaca como la mejor, usada en los Trials 0 y 3.
   - Otras funciones como `ReLU`, `ELU`, `Sigmoid` y `Tanh` no logran resultados comparables.
7. `lr` **(Learning rate):**
   - Valores intermedios en escala logarítmica (como **0.00018** en el Trial 0) son los mejores.
   - Tasas extremadamente bajas (como **1.79e-05**) tienden a ser ineficaces.
8. `optimizer` **(Optimizador):**
   - `NAdam` tiene el mejor rendimiento general (Trials 0, 3).
   - Otros optimizadores, como `ASGD` y `Adadelta`, generan bajas precisiones.
9. `batch_size` **(Tamaño del lote):**
   - Lotes pequeños (16) son los más efectivos, como en los Trials 0 y 3.
   - Lotes grandes (96, 128) tienden a reducir la precisión.

Resumen de los mejores valores observados:

- **num_filters_conv1**: 24
- **num_filters_conv2**: 96-112
- **kernel_size_conv1**: 5
- **kernel_size_conv2**: 5
- **fc1_units**: 384-448
- **dropout_rate**: 0.018-0.023
- **activation**: LeakyReLU
- **lr**: ~0.00018
- **optimizer**: NAdam
- **batch_size**: 16

6. Repositorio GitHub

El código completo de este proyecto, junto con los archivos necesarios para la implementación de la red neuronal convolutiva, está disponible en el repositorio de GitHub.

Enlace al repositorio: https://github.com/LuisPereraPerez/AA2_practicas