GUÍA CLASIFICACIÓN DE IMÁGENES

1.- OBJETIVO

Se trata de clasificar imágenes completas utilizados técnicas de Aprendizaje Profundo. Para ello se disponen las imágenes distribuidas en clases o categorías.

Por ejemplo, si se trata de clasificar monumentos de Madrid, las imágenes se distribuyen en carpetas y dentro de ellas se ubica cada imagen según su categoría. Dentro de la carpeta Monumentos se contienen las siguientes subcarpetas:

- Felipe III
- Fuente de Cibeles
- Neptuno
- Oso y el Madroño
- Puerta de Alcalá
- Puerta de Toledo

Tal es el caso de las imágenes contenidas por ejemplo en la categoría **Fuente de Cibeles**

2.- REDIMENSIONADO DE LOS DATOS

Las imágenes originales contenidos bajo la carpeta original necesitan redimensionarse con arreglo a las dimensiones requeridas por la red correspondiente. Los modelos siguientes son las redes que admite Matlab y requieren las dimensiones que se especifican:

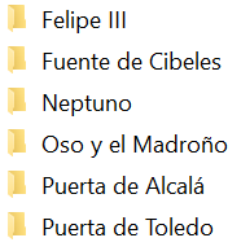
```
% Dimensión: 224x224x3: GoogleNet, VGG16, VGG19, ResNet18, ResNet50, ResNet101, densenet201
% Dimensión: 227x227x3: AlexNet, squeezenet
% Dimensión: 299x299x3: inceptionresnetv2, inceptionv3
```

Para llevar a cabo este redimensionado de forma automática, se utiliza el programa Matlab **ConvertirSizesRedes.m** cuyo contenido es el siguiente:

```
%% Redimensionado de las imágenes
% Dimensión: 224x224x3: GoogleNet, VGG16, VGG19, ResNet18, ResNet50, ResNet101, densenet201
% Dimensión: 227x227x3: AlexNet, squeezenet
% Dimensión: 299x299x3: inceptionresnetv2, inceptionv3


cd 'C:\Trabajo\Trabajos Fin de Grado\2023-24\FJFFerreiro-VPorras'
imds = imageDatastore('Monumentos',...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');

idx = size(imds.Files,1);

for i=1:1:idx
  D = cell2mat(imds.Files(i));
  Img = imread(D);
  I = imresize(Img, [224 224]);
  [a,b] = find(D =='\');
```

```
  %S2 = 'DATASET224x224';
  %S2 = 'DATASET227x227';
  S2 = 'DATASET299x299';
  S3 = D(b(6):size(D,2));
  fichero = [S1,S2,S3];
  imwrite(I,fichero);
end
```

Se toman las imágenes procedentes de la carpeta original, Monumentos y se ubican en una carpeta que se denomina DATASETYYYxYYY, la cual contiene la misma estructura que la carpeta original, con las mismas subcarpetas pero vacías. Tras la ejecución estas carpetas contendrán las imágenes redimensionadas.



3.- DESCRIPCIÓN DE LOS PROGRAMAS

Se dispone de programas que contienen partes específicas de Entrenamiento y Clasificación, a saber: **CNNTrainingMonumentos224x224, CNNTrainingMonumentos227x227 y CNNTrainingMonumentos299x299** todos ellos con extensión .m

3.1 Entrenamiento

La parte de entrenamiento contiene las siguientes estructuras, cuya explicación son las descripciones que aparecen en rojo.

```
clear all; close all;

% https://es.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html

Estructura del tipo Datastore, que redirecciona las imágenes contenidas en esa carpeta con las imágenes redimensionadas, según el modelo de red que se vaya a utilizar

imds = imageDatastore('DATASET224x224',...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');

Separación de las imágenes en conjunto de entrenamiento y test, con el parámetro 0.7 se especifica que el 70% sean de entrenamiento y el 30% para validación, las cuales se ubican en las estructuras imdsTrain y imdsValidation respectivamente.

[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');

Separación de las imágenes para ver una muestra de ellas, en total 16

numTrainImages = numel(imdsTrain.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsTrain,idx(i));
    imshow(I)
end
```

Carga de los modelos pre-entrenados. Los modelos disponibles son los que aparecen en la siguiente dirección: https://es.mathworks.com/help/deeplearning/ref/alexnet.html
En concreto son los siguientes
alexnet| vgg16|vgg19|resnet18|resnet50|densenet201|googlenet|inceptionresnetv2|squeezenet|

Hay que tener en cuenta que esos modelos están pre-entrenados con el Dataset Imagenet (https://www.image-net.org/update-mar-11-2021.php), para 1000 clases. Y en nuestros programas lo que se hace es volverlos a entrenar para nuestro dataset, por eso se necesitan relativamente pocas imágenes. Esto es lo que se conoce como transferencia de aprendizaje.

```
%% Redes con dimensiones de imagen 224x224x3: googlenet, vgg16, vgg19, resnet18, resnet50,
resnet101, densenet201

%red ='googlenet';
%red ='vgg16';
red ='vgg19';
%red ='resnet18';
%red ='resnet50';
%red ='resnet101';
%red ='densenet201';
```

Según el modelo de red seleccionado se carga el modelo pre-entrenado correspondiente, que queda almacenado en la variable net

```
switch red
    case 'googlenet'
        net = googlenet;
    case 'vgg16'
        net = vgg16;
    case 'vgg19'
        net = vgg19;
    case 'resnet18'
        net = resnet18;
    case 'resnet50'
        net = resnet50;
    case 'resnet101'
        net = resnet101;
    case 'densenet201'
        net = densenet201;
end
```

En estas dos líneas se determina la dimensión de entrada de los modelos de redes, a través de la variable inputSize, que será la dimensión requerida por las imágenes.

```
net.Layers(1)
inputSize = net.Layers(1).InputSize;
```

Análisis del modelo de red cargado para deterimar que es coherente y no contiene errores.

```
analyzeNetwork(net)
```

Transformación de la red a una representación determinada por layerGraph.

```
if isa(net,'SeriesNetwork')
  lgraph = layerGraph(net.Layers);
else
  lgraph = layerGraph(net);
end
```

Hemos dicho que los modelos cargados están preparados para identificar 1000 clases, por este motivo hay que modificar las últimas capas del modelo para adaptarlo a nuestro conjunto de clases. Identificando el número de clases, que se obtienen de las carpetas donde se han ubicado las imágenes.

```
[learnableLayer,classLayer] = findLayersToReplace(lgraph);

numClasses = numel(categories(imdsTrain.Labels));
```

```
if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end
```

Reemplazamiento de las capas, junto con las conexiones. Y visualización gráfica de los modelos de red.

```
lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

figure('Units','normalized','Position',[0.3 0.3 0.4 0.4]);
plot(lgraph)
ylim([0,10])

layers = lgraph.Layers;
connections = lgraph.Connections;
```

Estas sentencias, a través de la función freezeWeights, lo que hacen es fijar los pesos de las capas indicadas, concretamente en este caso las capas 1:10. Esto se puede variar para hacer pruebas por ejemplo 1:5 o 1:15, etc., indicando cuántas capas se congelan. Lo de la congelación significa que durante el entrenamiento en las capas indicadas no se actualizan los pesos, sólo se ajustan los pesos del resto de capas con nuestras propias imágenes.

```
layers(1:10) = freezeWeights(layers(1:10));
lgraph = createLgraphUsingConnections(layers,connections);
```

La siguiente sentencia analiza el modelo de red modificado para ver si tiene errores

```
analyzeNetwork(lgraph)
```

Las siguientes sentencias generan nuevas imágenes a partir de las ya existentes. Es lo que se llama “aumento de imágenes” (image augmentation) tanto para las imágenes de entrenamiento (augimdsTrain) como de validación (augimdsValidation). La operación image augmentation consiste en girar las imágenes, variar los colores, meterlas ruido, etc, con el fin de simular nuevos casos e incluso casos desfavorables, todo ello para favorecer el aprendizaje.

```
pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);

augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);

augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

En las siguientes líneas se muestran algunas (16) imágenes aumentadas.

```
numaugsValidationImages = augimdsValidation.NumObservations;
idx = randperm(numaugsValidationImages,16);

figure
```

```
for i = 1:16
    subplot(4,4,i)
    I = imread(augimdsValidation.Files{idx(i),1});
    imshow(I)
end
```

Las siguientes sentencias generan son de especial relevancia, ya que permiten definir, a través de la función trainingOptions, distintos hiperparámetros para el entrenamiento. Entre ellos los resaltados en verde. Son importantes porque es posible y conveniente modificarlos para ver su comportamiento y reflejarlo luego en la memoria del trabajo. Se puede ver qué opciones admite la función trainingOptions en el siguiente link:
https://es.mathworks.com/help/deeplearning/ref/trainingoptions.html

Por cierto, desde la línea de comandos de Matlab, siempre es posible ver la documentación de cualquier función de Matlab haciendo >> help nombre_de_la_función, por ejemplo:
>> help trainingOptions

```
miniBatchSize = 10;
valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'ValidationPatience',Inf, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

La siguiente función es la que realiza el entrenamiento propiamente dicho, mostrando gráficamente la evolución del mismo

```
netTransfer = trainNetwork(augimdsTrain,lgraph,options);
```

El modelo de red con todos los pesos ajustados (aprendidos) se encuentran en la variable netTransfer. Esta variable se puede guardar en un fichero, que en Matlab tiene la extensión .mat. Así, con el comando siguiente save, se guarda el contenido de la variable netTransfer en el fichero netTransferMonumentosvgg16.mat. Hay que tener cuidado, para no m

```
save netTransferMonumentosvgg16 netTransfer
```

Una vez que se tiene la variable el comando equivalente para cargarla desde el fichero es load, recuperando la misma variable guardada previamente.

```
load netTransferMonumentosvgg16
```

3.2 Clasificación

Esta parte realiza la clasificación de las imágenes disponibles en el conjunto de validación, incluyendo las imágenes previas en este conjunto y las aumentadas, según se ha explicado previamente. Es decir, las imágenes que se tienen augimdsValidation y para ello se utiliza el modelo de red ajustado que tenemos en la variable netTransfer

```
%% Validación del proceso de entrenamiento

[YValidationPred,probs] = classify(netTransfer,augimdsValidation);
validationAccuracy = mean(YValidationPred == imdsValidation.Labels)
validationError = mean(YValidationPred ~= imdsValidation.Labels)

YTrainPred = classify(netTransfer,augimdsTrain);
trainError = mean(YTrainPred ~= imdsTrain.Labels);
disp("Error Entrenamiento: " + trainError*100 + "%")
disp("Error Validacion: " + validationError*100 + "%")
```

A continuación se muestran los resultados a través de lo que se conoce como matriz de confusión.

```
%% Plot the confusion matrix. Display the precision and recall for each class by using column and row summaries. Sort the classes of the confusion matrix. The largest confusion is between unknown words and commands, up and off, down and no, and go and no.

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
cm = confusionchart(YValidationPred,imdsValidation.Labels, ...
    'ColumnSummary','column-normalized', ...
    'RowSummary','row-normalized');
A1 = 'Matriz Confusion Validacion: ';
A2 = red;
cm.Title = [A1,A2];
```

Este conjunto de sentencias muestra algunos resultados y las probabilidades de clasificación

```
idx = randperm(numel(imdsValidation.Files),8);
figure
for i = 1:8
    subplot(4,2,i)
    I = readimage(imdsValidation,idx(i));
    imshow(I)
    label = YValidationPred(idx(i));
    title(string(label) + ", " + num2str(100*max(probs(idx(i),:)),3) + "%");
end
```

Las siguientes líneas muestran en forma gráfica, los pesos aprendidos por la red, en sus capas. En este caso, los correspondientes a la capa dos

```
%% Visualización de los pesos

% Get the network weights for the second convolutional layer
w1 = netTransfer.Layers(2).Weights;

% Scale and resize the weights for visualization
w1 = mat2gray(w1);
w1 = imresize(w1,5);

% Display a montage of network weights. There are 96 individual sets of
% weights in the first layer.
figure
montage(w1)
title('Pesos primera capa convolucional')
```

3.3 Clasificación sobre imágenes nuevas

Dada una imagen nueva, por ejemplo, una imagen captada con un dispositivo móvil y una vez se tiene en un fichero, se puede proceder a su clasificación. Si la imagen está en la variable Img, que se obtiene a partir del siguiente comando

Img = imread ('C:\Trabajo\Trabajos Fin de Grado\2023-24\FJFFerreiro-VPorras\DATASETGoogleNet\Fuente de Cibeles\1cibeles.jpg');

```
[YValidationPred,probs] = classify(netTransfer,Img);
```