SYSC4906 Introduction to Machine Learning Fall 2019

Assignment 2

You are provided with four collections of images: - Trump_train: 100 images of President Trump - Trump_test: 39 images of President Trump - Decoys_train: 100 randomly selected decoy images - Decoys_test: 41 randomly selected decoy images You are to develop a convolutional neural network (CNN) that can distinguish between the two image classes (Trump vs. Decoy). We will leverage transfer learning to retrain a pre-trained network for this new task. **Deliverables are marked in bold.**	 **Image credit:** https://twitter.com/deepdrumpf

Resources for this problem:

- SYSC4906_Assig2_Data.zip on the SYSC4906 GitHub repository under Assignments
- These instructions in PDF, also on GitHub repo
- Template for your solution at: https://github.com/jrgreen7/SYSC4906/tree/master/Assignments/Assignment2
- Description of all pre-trained models available in Keras: https://keras.io/applications/
- Tutorial on transfer learning and data augmentation using InceptionV3 model:
    - https://medium.com/abraia/first-steps-with-transfer-learning-for-custom-image-classification-with-keras-b941601fcad5
    - Note that there is a complete notebook associated with this tutorial that is ***very useful for this question*****...**
- Tutorial on transfer learning: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
- Tutorial on image data augmentation: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Step 0: Create a Jupyter notebook on Google Colab **based on the template provided**.

- Make sure you change your runtime type to set “Hardware Acceleration = GPU”
- Look at the resources above...

Step 1: Prepare the dataset.

- Load the four image sets. Note that they have already been resized to 299x299 pixels.
    - See SYSC4906_Assig2_Data.zip on the GitHub repository
- Normalize the pixel values across all images, from all sets.
- Augment your dataset using horizontal and vertical shifts only.
- Convert all images to numpy arrays.
- **Deliverable 1: Display the first image in each of the four sets. Each image's title should indicate which of the four sets it represents.**

Step 2: Load pre-trained CNN

- Load a pre-trained CNN network from Keras. Specifically, load the entire InceptionV3 network using ImageNet pre-trained weights.
- Once loaded, use `summary()` to display the structure of the network.
- **Deliverable 2: Report how many learnable parameters are in layer "`conv2d_1 (Conv2D)`" of the model.**
    - Try to figure out where this number comes from...

Step 3: Test pre-trained InceptionV3 CNN on our data. This model is trained on the ImageNet database, so it will predict up to 1000 class labels for an input image.

- Apply the pre-trained InceptionV3 CNN to classify the following images:
    - train/trump/100.jpeg
    - test/decoy/28.jpeg
- **Deliverable 3: What are the top-ranked predictions and their scores from the InceptionV3 original model for these two images?**

Step 4: Reload the InceptionV3 CNN, but with new dense layer. We are going to use transfer learning to re-train only the "top" application-specific layers (global average pooling followed by a fully-connected dense layer) and keep the pre-trained base model as a feature extractor.

- Re-load the ImageNet pre-trained InceptionV3 CNN model, but this time exclude the final ("top") GlobalAveragePooling and dense network layers.
- Re-add a new `GlobalAveragePooling` layer, followed by a Dropout layer (50% chance of dropout), then a Dense layer with two classes and terminating with a softmax activation
- Freeze all layers in the base model, so that training does not affect them.
- Compile the model using categorical_crossentropy as the loss function, accuracy as the performance metric, and Adam as the optimizer.
- The layers should match the original model (see original network structure from Step 2).
- **Deliverable 4: Examine the layers in the full model (see Step 2) and the new modified model using summary(). Which layer has the largest difference in the number of learnable parameters, when comparing the two models? What type of layer is it and why did the number of learnable parameters change?**

Step 5: Create training and validation image generators to augment image sets

- Create image generators for both training and validation using two `ImageDataGenerator` objects. Generators are useful when you want to create each batch of data as needed during training, rather than pre-compute all the augmented images prior to training. This saves storage/memory, at the expense of regenerating data at each epoch.
- Create `ImageDataGenerator` objects for training and for validation that:
    - Normalize the pixel values for each image, as required for input to the InceptionV3 pre-trained feature extractor layers (use `keras.applications.inception_v3.preprocess_input`.)
    - Augment your dataset using horizontal and vertical shifts only, of up to 0.3, and horizontal flips. Use the "nearest" fill mode.
    - Convert all images to numpy arrays.
- Use `flow_from_directory` on your `ImageDataGenerator` objects to create two `DirectoryIterator` objects.
    - Specify the image height and width of 299 and a batch size of 32.
    - Set `class_mode` to be 'categorical'
- **Deliverable 5: Invoke "next" on your training `ImageDataGenerator` to create a batch of 32 images and labels (where each label is a tuple). Create a 4rowx8col subplot matrix showing all 32 images. Each image should be titled with its label tuple.**

Step 6: Use transfer learning to train the new CNN

- Freeze the weights of all layers from the base model, except the new `GlobalAveragePooling and dense` layers you added.
- Train the model using `fit_generator()`.
    - Use the two `DirectoryIterator` objects, a batch size of 32, 10 epochs, with 3 training steps per epoch. The `validation_data` argument should be set to your validation `DirectoryIterator` object and `validation_steps` should be set to 1.
- Use 10% of the training data as a validation set to evaluate loss at each training epoch.
- **Deliverable 6: Plot the training and validation loss at each training epoch (i.e. the learning curve). Repeat for accuracy instead of loss.**

Step 7: Test the new CNN

- Apply the re-trained CNN to the two (non-augmented) test (validation) sets.
- **Deliverable 7: Report a confusion table and the total accuracy.**
    - Note that I've provided a `printCM` function