Capstone Project **Seongjae Ryu**

Machine Learning Engineer Nanodegree **August 17, 2021**

CNN Project: Dog Breed Classifier

Domain Background

Dog breed classifier is one of the popular CNN(convolutional neural network) projects [1]. The main problem is to identify a dog breed from an input image. It also needs to be identified whether it is an image of a dog or not. In case when a given input is identified as an image with a human face, a similar dog breed will be provided.

This idea is based on supervised machine learning with a multi-class classification problem. The intention of this model is to deploy APIs and build an application with the APIs.

Problem Statement

The problem is to build a dog breed classification model and deploy this model for an application.

- **Dog-image identifier**: Inferring whether it is a dog image or not from an input image.
- **Dog-breed classifier**: Inferring a dog breed from an input image.
- **Human-face-image identifier**: Inferring whether it is a human face image or not from an input image.
- **Human-face-image Resemble-dog classifier**: Inferring a dog breed, resembling a human face input image.

Datasets and Inputs

Dog images and human-face images are included in the dataset, which is labeled, organized, and provided by Udacity [2].

1. **Dog image dataset**: It contains 8,351 RGB images of dogs with each labeled file name. The images are organized in each directory; train, test, and valid. There are 6,680 train images, 836 test images, and 835 valid images. Each breed of 133 is grouped in each folder with label number and breed name; i.e. '/dog_images/train/103.Mastiff/Mastiff_06826.jpg'. Image sizes are varied.

2. **Human-face dataset**: It contains 13,233 RGB human-face images with each labeled file name. Images are grouped by each person in each folder with label number and person's name; i.e. '/Daniele_Bergamin/Daniele_Bergamin_0001.jpg'. There are 5,750 folders and

the data is imbalanced, such as one has few images and the other has many images. The image size is 250 by 250 px.

Solution Statement

A convolutional neural network is used for multi-class classification, dog breed classifier. Haar Cascade Detection is used for a face detection algorithm.

1. **Dog-image identifier**: Pre-trained VGG-16 torchvision model [3].
    a. Load pre-trained VGG-16 model via torchvision.
    b. Load image and convert into tensor image.
    c. Infer the image and check whether the output is in 133 breeds or not.

2. **Dog-breed classifier**: Fitted pre-trained VGG-16 torchvision model with custom classifier for 133 breeds.
    a. Load pre-trained VGG-16 model via torchvision.
    b. Define a new custom classifier to fit the model into the dataset with 133 breed categories.
    c. Load image and convert into tensor image with transforms.
    d. Train the model with train dataset and valid dataset.
    e. Test the model with a test dataset.
    f. Deploy the model and infer the most likely breed.

3. **Human-face-image identifier**: OpenCV with pre-trained face detector, haarcascade_frontalface_alt [4].
    a. Load pre-trained OpenCV model with haarcascade_frontalface_alt.xml.
    b. Load and convert RGB image into grayscale.
    c. Infer the image.

4. **Human-face-image Resemble-dog classifier**: Combination of human-face-image identifier and dog-breed classifier.
    a. Infer the image via human-face-image identifier and if the inference tells it is a human face image, infer the breed via dog-breed classifier.

Benchmark Model

1. Human-face-image identifier must have 70% or above true inference and vice versa.
    - The percentage of human face images inferred as a human face:
        - $\frac{Number\ of\ predictions\ as\ human-face}{Total\ number\ of\ prediction\ with\ human\ face\ dataset}$
    - The percentage of dog images inferred as not a human face:
        - $\frac{Number\ of\ predictions\ as\ not-human-face}{Total\ number\ of\ prediction\ with\ dog\ dataset}$

2. Dog-image identifier must have 70% or above true inference and vice versa.
    - The percentage of dog images inferred as a dog:
        - $\frac{Number\ of\ predictions\ as\ dog}{Toral\ number\ of\ prediction\ with\ dog\ dataset}$
    - The percentage of human face images inferred as not a dog:

        - $\frac{Number\ of\ predictions\ as\ not-dog}{Total\ number\ of\ prediction\ with\ human\ face\ dataset}$

3. Dog-breed classifier must have 60% or above accuracy.
    - $Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$

Evaluation Metrics

1. Human-face-image identifier must have 70% or above true inference and vice versa.
    - The percentage of human face images inferred as a human face:
        - The result is acceptable. The percentage is 98% which is above 70%.
    - The percentage of dog images inferred as not a human face:
        - The result is acceptable. The percentage is 17% which is below 30%.

2. Dog-image identifier must have 70% or above true inference and vice versa.
    - The percentage of dog images inferred as a dog:
        - The result is acceptable. The percentage is 80% which is above 70%.
    - The percentage of human face images inferred as not a dog:
        - The result is acceptable. The percentage is 0% which is below 30%.

3. Dog-breed classifier must have 60% or above accuracy.
    - The result is acceptable. The percentage is 74% which is above 60%.

Project Design

- Step 0: **Import dataset**
    - Create numpy file lists.
        - Dog image file list from Udacity dogImages.zip file [5].
        - Human image file list from Udacity lfw.zip file [5].
    - Check the number of images in each file list.
        - There are 8,351 total dog images.
        - There are 13,233 total human images.

- Step 1: **Detect Humans**
    - Create OpenCV model with a pre-trained face detector to detect human faces.
        - Create cv2 Cascade Classifier with haarcascade_frontalface_alt.xml.
        - Test the created classifier.
            1. Load RGB image from numpy image-file list
            2. Convert RGB image into grayscale and do prediction
            3. Show the prediction result and check the result.
        - Define face detector function with the classifier
            1. Load RGB image from numpy image-file list
            2. Convert RGB image into grayscale and do prediction
            3. Show the prediction result and check the result.
    - Evaluate the face detector with a part of each dataset.

- Step 2: **Detect Dogs**
    - Create torchvision model with pre-trained VGG16 model.

        - Download and create pre-trained torchvision VGG16 model, and set the device for the model.
        - Define VGG16 prediction function.
            1. Create a Pillow image object from image path.
            2. Transform the image into 224 by 224 size with adding randomness.
            3. Do prediction of the transformed image via VGG16 model.
            4. Return the index of the maximum value among predicted classes.
        - Evaluate VGG16 prediction and check the result is in 133 breeds.
            1. Define dog detector test function to count the number of dog detections and get the percentage.
            2. Test the function with a part of human dataset.
                a. Check the percentage is below 30%.
            3. Test the function with a part of dog dataset.
                a. Check the percentage is above 70%.

- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
    - Create a custom CNN model via Pytorch
        - Define the CNN architecture.
            1. Initialize layers.
                a. Convolutional layer with 3 channels(RGB) as an input.
                b. Max-pooling to dimension reduction.
                c. Classifier with 133 outputs setting.
            2. Define forward method.
    - Define a criterion.
    - Define a optimizer.
    - Define a train function.
        - Looping for epochs
            1. Initialize loss for each train and validation.
            2. Set the model into train mode.
            3. Start training the model.
                a. Move the train data into GPU if possible.
                b. Zero the gradient of the optimizer.
                c. Predict the result.
                d. Calculate the loss via the criterion.
                e. Calculate the gradient.
                f. Update the weights of the optimizer.
            4. Set the model into evaluation mode.
            5. Start validating the model.
                a. Move the validation data into GPU if possible.
                b. Predict the result.
                c. Calculate the loss via the criterion.
            6. Save the model parameters if the validation loss is the lowest ever.
        - Return a trained model.
    - Load the model parameters if needed.
    - Train the custom CNN model via the train function with a part of data of each train and validation.
    - Evaluate the custom model with a part of test data.

- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
    - Create a CNN model, using pre-trained torchvision model, VGG16.
        - Download and create pre-trained torchvision VGG16 model, and set the device for the model.
        - Turn off all gradient requirement options off.
        - Modify classifier of the pre-trained model.
            1. Check the architecture of the loaded model.
            2. Modify the output size into 133.
        - Move the model into GPU if possible.
    - Define a criterion.
    - Define a optimizer, includes parameters of which each gradient requirement option is on.
    - Train the model via the train function with train data and validation data.
    - Load the model parameters if needed.
    - Evaluate the custom model with test data.
        - Check the accuracy is above 60%.
    - Define a dog breed classifier.
        - Create a list of class names by index.
        - Define classifier function.
            1. Create a Pillow image object from image path.
            2. Convert the image into RGB.
            3. Transform the image into 224 by 224 size with adding normalization.
            4. Move the image into GPU if possible.
            5. Do prediction via a CNN model, using Transfer Learning.
            6. Move the model into CPU if it was in GPU.
            7. Get the most likely breed index among the prediction result
            8. Return a breed name via the index and the class names list.

- Step 5: Write your Algorithm
    - Define an app function.
        - Display input file name and graphic.
        - Display prediction output result for each case.
            1. Case(1): Human face
            2. Case(2): Dog
            3. Case(3): Neither

- Step 6: Test Your Algorithm
    - Test the app function
        - Looping for a part of each human data and dog data.

Reference

[1] Dog Breed Identification, Kaggle - Kaggle: https://www.kaggle.com/c/dog-breed-identification
[2] Dog Project, Udacity - Github: https://github.com/udacity/dog-project/blob/master/dog_app.ipynb
[3] torchvision.models - Pytorch: https://pytorch.org/vision/stable/models.html
[4] haarcascades, OpenCV - Github: https://github.com/opencv/opencv/tree/master/data/haarcascades

[5] Dog image dataset, Dog App - Udacity:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
[6] Human image dataset, Dog App - Udacity:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip