Project Report Dog Breed Classifier

Step 0: Import Datasets

For this part, there is nothing special, just download the required datasets as mentioned in the notebooks. There are total 13233 human images and 8350 dog images.

Step 1: Detect Humans

For this part we used the OpenCV Cascade Classifier which output the face locations in an image. This function can also be used as a classifier for human/non-human based on the number of detected faces (0 means non-human, everything else means human).

Applying the face detector to the images we have yield relatively good results: 96% of the first 100 images in human files have a detected human face, 18% of the first 100 images in dog files have a detected human face.

Step 2: Detect Dogs

For this section, we used the pretrained VGG16 model. This model takes an image of (3, 224, 224) as input and returns a label of 1000 common objects. Without any modification, this model can be used to detect dogs because label [155, 268] correspond different dog breeds. Thus, if the predicted label is in the above range, we can output 1 as dog detected, otherwise no dog detected.

Here, I used PIL to load image and apply transform with Pytorch to the required input and pass it to the VGG16 model. In the end, for the first 100 human files, non was classified as dogs. And among the first 100 dog files, 91% of them are classified as dogs.

Step 3: Create CNN to classify the dog breed

- **Preprocessing**: I created an CNN from scratch for the dog classifier. The input shape that I want to use is (3, 64, 64). For the training images, my code will first resize the image to h, w = 80, 80 and then use the random crop to adjust it to (64, 64) which is the input size for my neural network; for the test images, my code directly resizes the image to (64, 64). Besides that, I used random crop and random horizontal flip to augment the dataset which could increase the predictive power of the model. Finally, both train and test set will be normalized.
- **Model architecture:**
    - The input size for my CNN is (3, 64, 64). The first step is to use convolutional layers to reduce the image size while also increasing the number of channels. Here I use four convolution layers conv0-3 to transform the input into (24, 7, 7).
    - Now I get the flatten input of (24, 7, 7) and start to construct the fully connected layer. Here I have three fully connected layers fc1-3. And fc3 is the output layer that has 133 classes. Reason: Use fully connected layers to achieve the final classification.
    - I also applied dropout in fully connected layers to prevent overfitting.
    - The loss function I used is the cross-entropy function. For the optimizer, I choose the Adam optimizer with learning rate = 0.001.

- **Training and testing:** The training algorithm goes through the training data via data loader. For each mini batch, it first sets model as train mode, forward computes the network, calculates the loss, and uses optimizer to update the parameters through the gradients. After the end of each epoch, it computes the validation loss. If the validation loss decrease compares to the minimum validation loss before, it saves the model. After the training method finishes, I test the model against the test datasets.
- **Results:** For the training process, after 100 epochs, the training loss is about 2.8 and the best validation loss is around 3.8. The test result gives 11 percent accuracy.

Step 4 Create CNN to classify dog breed use transfer learning

- **Model to use**: The pretrained model I used is VGG16. The input transformation is similar to the previous step, except here I transform the image size to (3, 224, 224) to suit the input of VGG16.
- **Model architecture**:
    - VGG16 has two parts, the first part is convolution layers. This part is mainly used to detect patterns. This part can be directly applied to my use case without updating the parameters. So, I have the parameters in this part as untrainable.
    - The second part is fully connected layers which are the classifier part. I change the out features of the last layer to 133 to suit my use case. And set this part as trainable. Reason: it is good to train the fully connected layer as the last part of the transfer learning.
    - I think this architecture is suitable for my use case because VGG16 is a successful model in image classifications and it is also the main goal of my project. I could use the pre-trained weights of VGG16 efficiently with the last few layers trainable.
- **Results**: The training and testing are similar as before, after 25 epochs, the training loss is around 1.7 and the lowest validation loss is around 1.0. The test accuracy on the same testing set is 72%.

Step 5,6 Write the final algorithm

- **Predict the breed with the model**: The output of the neural network is an integer. I first get all dog breed names from the ImageFolder class that I used to create the data loader. In the end I get a list of names, the predicted breed name is the value correspond to the predicted label. For example, if the output from a model is 5, then the breed name is class_names[5].
- **Final Algorithm:** Now we have the human face detector, dog detector and a dog breed classifier. I can write the final application. The algorithm first tests the image against face detector, if it found human faces, the image will be passed to the dog breed classifier and the output breed is the dog breed that resembles the human face. If human face is not detected, the algorithm tests the image against dog detector, if a dog is detected. It will use the dog breed classifier to classify the dog breeds. Finally, if the image is neither human nor dog, return a message that indicator no human or dog found.
- **Results:** I test 10 human files and 10 dog files. For the human files, 9 out of 10 images are classified as human and outputs dog breed that resembles the face. For the dog files, 8 out of 10 images are classified as dogs and outputs the correct breed for 6 of them.

Conclusion:

Overall, I am satisfied with my results. I create a CNN from scratch that has 11 percent accuracy which is significantly better than guessing. I believe given more training data and more complexed model architecture; the result will be better. The transferring learning with VGG16 gives 72 percent accuracy which is amazing.