Sriram Pallapothu
Udacity Capstone project
Machine Learning Nanodegree

Dog Breed Classifier

19th September 2020

PROJECT OVERVIEW

Dog breed classification is one of the popular Machine Learning image classification problems. Given an image of a dog, the ML model should be capable of identifying the breed, which is quite difficult even for a human as there could be breeds that resemble a lot with each other.

In this project, I have built a Machine learning model using CNN (Convolutional Neural Network) model.

PROBLEM STATEMENT

The goal is to create an application that will accept an image and classify it among 133 dog breeds. This application can also identify a human image and identify a closest looking breed of dog.

The steps followed in achieving our goal -

1. Download and import human and dog images
2. Process the data
3. Identify human images
4. Identify dog images
5. Train a model to classify dog breeds (from scratch)
6. Use transfer learning to train a pre-trained model
7. The trained model can be part of a bigger web app that can accept images
8. Display the results (dog breed)

METRICS

As most of the image classification problems, this model is also trained on CNN.

Training loss and validation loss are calculated after epoch and the model is saved only if the validation loss is also reducing. Thus, preventing overfitting.

Accuracy is calculated on the test data and the model is trained/ hyperparameters are tuned until the below criteria are met -

- **Model to detect human -** this is a pre-trained OpenCV's face detection model which is accurate in most of the cases. The accuracy of this model is almost 100%
- **Model to detect dog -** this is a pre-trained VGG-16 model, that gives output categories a number between 0-999. This model is also highly accurate with almost 100% accurate results
- **Model to detect breed (from scratch) -** this is a model built from scratch and using CNN and this can be acceptable if it has an accuracy of at least 10%
- **Model to detect breed (transfer learning) -** the model is acceptable if it has an accuracy of at least 60%

DATA EXPLORATION

The data is collected from Udacity's storage at AWS.

Human images

Downloaded from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip.

The dataset consists of a total of 13233 human images, divided into folders one per person. Some of the folders have multiple images, some of them have only one. The images are at various brightness levels, orientations, and sizes. We need to perform some transforms and make them all similar to test the accuracy of the model.

Dog images

Downloaded from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip.

The data set has three folders, each one for train, validation, and test. Each of these folders has 133 folders each folder corresponding to the breed of the dog. This data set also needs transformation before we can use them to train/test the models.

EXPLORATORY VISUALIZATION

Open CV's human identifier

The algorithm uses Edge features and identify various parts of a human face and puts them back together by Adaboost classifier

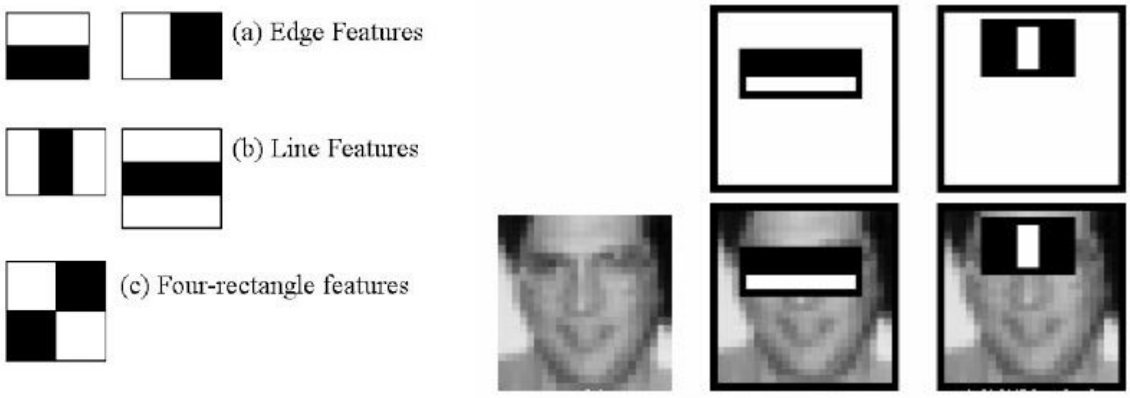
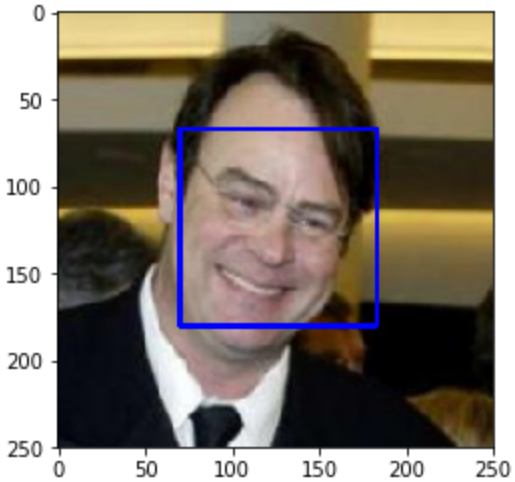


Image source: https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html

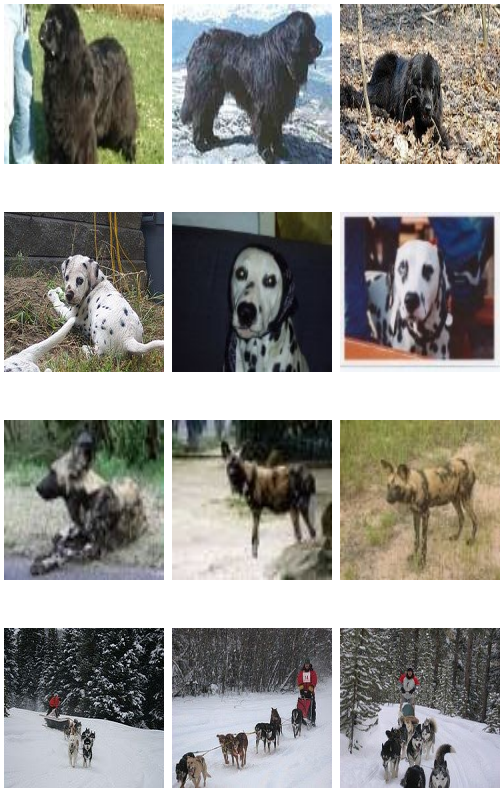
After identifying it then return us the coordinates of the corners of the face, which we can use to mark the face.



Dog identifier (VGG-16)

Trained on ImageNet's images to categorize an image into one of the 1000 categories.

ImageNet has a huge collection of images of every category



Dog breed classifier

To classify the correct dog breed is challenging as sometimes two breeds may look similar that even a human may find it difficult to identify the breed.

Brittany vs Welsh springer spaniel

CNN models separate the image into various layers and detect background, varying intensity, object, etc. a visualization of various layers -

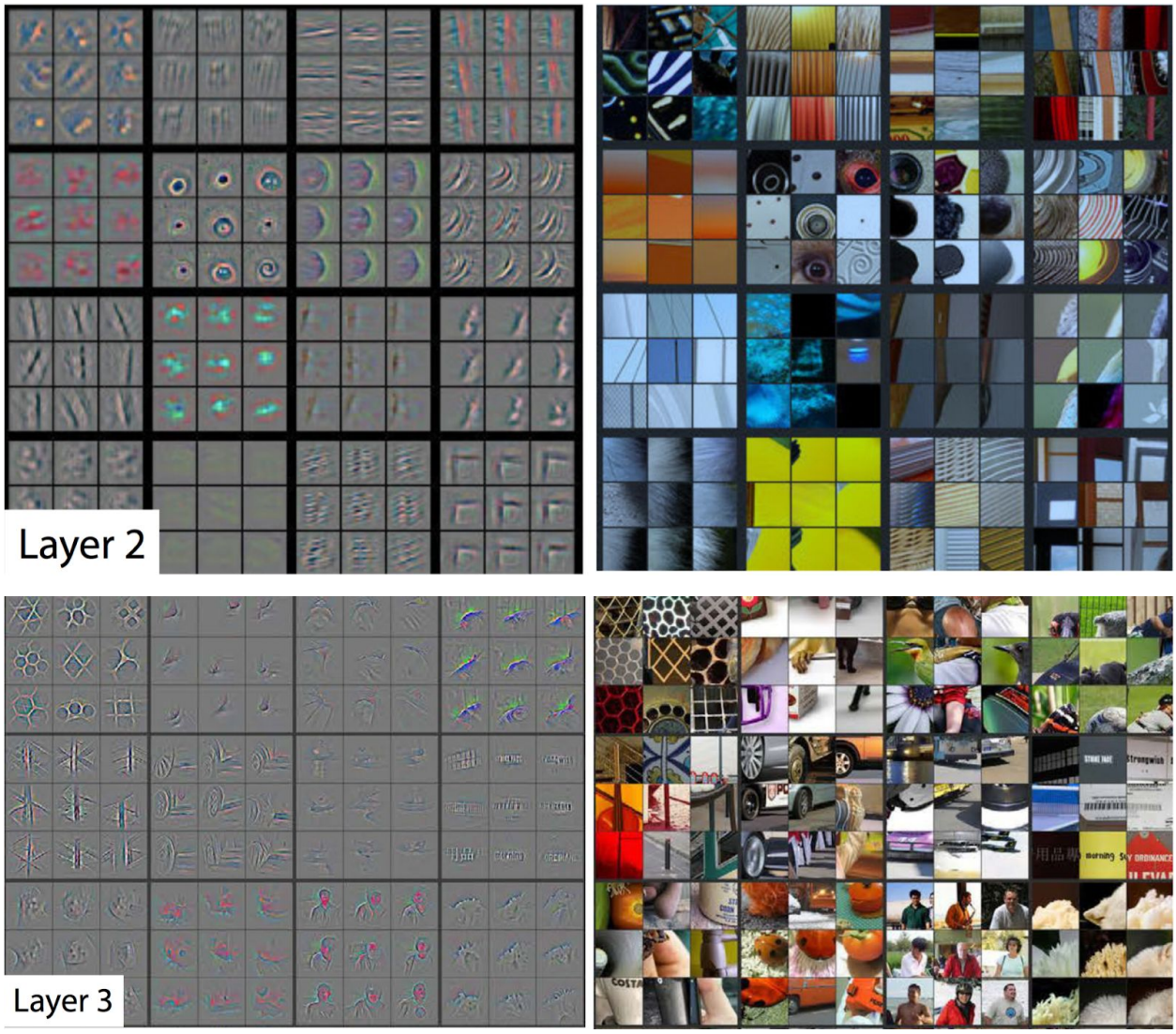


Image source - Udacity CNN course

ALGORITHMS AND TECHNIQUES

Human identifier

OpenCV's implementation of Haar feature-based cascade classifiers is used to detect Human image - http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

These pre-trained models are stored as XML files on Github - https://github.com/opencv/opencv/tree/master/data/haarcascades

This pre-trained model is highly accurate and the results are almost 100% correct in most of the cases

Dog identifier

VGG-16 model which is pre-trained on ImageNet's images is used for detecting if its a dog.

ImageNet is a very large and popular dataset used for image classification and vision tasks. Each image is categorized into one of the 1000 categories - https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

The dog breeds have indexes from 151 to 268 (both inclusive). so if VGG-16 returns a value between these two numbers, then the image in question is classified to be a dog

Dog breed classifier

Build a model from scratch

CNN model is built from scratch and tested on the test data set of dogs. CNN models perform well for image classification problems. We can analyze an image properly by dividing it into different layers, each analyzing a different feature like identifying boundaries, identifying background, etc. so, CNN is preferred in most of the image classification problems. The benchmark for this model is at least 10% accuracy.

Use transfer learning

The final model will be from transfer learning. There are some good models that are fairly accurate in image classification. Few of the notable classification algorithms -

- AlexNet
- VGGNet
- ResNet

More information on choosing the right algorithm - https://www.learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/

The benchmark in selecting a transfer learning model will be an accuracy of at least 60%

Final algorithm

1. Check if the image is human
2. If human then return resembling dog breed
3. If a dog return dog breed
4. If none then print error saying its neither

BENCHMARK

- Human detector - As it is a pre-trained model the accuracy should be close to 100%
- Dog detector - this is also a pre-trained model. Accuracy must be very high. Close to 100%
- Dog breed classifier (from scratch) - its a very challenging problem, as a wild guess has the chance of 1/133 to be correct, which is less than 1%. So, a fair model will be considered if the accuracy is at least 10%
- Dog breed classifier (transfer learning) - this should have an accuracy of at least 60%

DATA PREPROCESSING

Transformations must be performed on the images so that they are all uniform and acceptable by the training models.

Most of the pre-trained models accept 224*224 images so, it's a good idea to resize the images to 224*224.

Also, additional transformations like random rotation, horizontal flips will also help train the model better as it helps the model to identify objects even if they are differently oriented.

```
train_transforms = transforms.Compose([transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.RandomRotation(15),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],
                                                            [0.229, 0.224, 0.225])])

test_transforms = transforms.Compose([transforms.Resize(size=(224,224)),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],
                                                           [0.229, 0.224, 0.225])])
```

IMPLEMENTATION

Step 1: Detect humans

OpenCV's implementation Haar feature-based cascade classifiers are stored as XML files on GitHub at - https://github.com/opencv/opencv/tree/master/data/haarcascades

This pre-trained face detector is extracted using cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

We get the boundaries of the face when we pass a grayscale image through the face detector. The face detector does not return anything if no face is found.

Step2: Detect Dogs

A pre-trained VGG-16 model is used to detect dogs in images. It basically classifies an image into one of the 1000 categories - https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

Of all the categories; categories 151 to 268 (both inclusive) are different breeds of dogs. So, the image most likely has a dog in it if the model returns any value between these two numbers.

Step 3: Create a CNN to classify dog breeds (from scratch)

First, a CNN model's architecture is defined as

```
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 5, padding=2)
        self.conv3 = nn.Conv2d(64, 128, 5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)

        self.input = nn.Linear(28*28*128, 1024)
        self.output = nn.Linear(1024,133)

        self.drop = nn.Dropout(0.4)


    def forward(self, x):
        ## Define forward behavior

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        x = x.view(-1, 28*28*128)

        x = F.relu(self.input(x))
        x = self.drop(x)
        x = F.relu(self.output(x))

        return x
```

It is a good practice to increase the number of layers during CNN so that we will be able to distinguish most of the features like boundaries, background, varying intensity, etc. So, I have made the architecture in such a way that the layers are gradually increasing from 3 ->32 -> 64 -> 128.

At the same time, I also passed every convolution layer through Maxpool to keep the size of the image in check, not making the model too complex and to avoid overfitting.

Relu activation function is used after every layer.

This model is then trained on the training data, and saving the model only if the validation loss is decreasing. Thus, ensuring we are not overfitting.

Step 4: Create a CNN to classify dog breeds (using Transfer Learning)

There are many popular models that are good at image classification like LAexNet, ResNet, VGG. I picked the ResNet model as it has a low Top-1 error %.

First, the resnet50 model is imported -

model_transfer = models.resnet50(pretrained=True)

Then, we need to change the output features to suit our needs. In this case, it will be 133 outputs.

model_transfer.fc.out_features = 133

The model is trained as earlier and this time we should notice that it should already start with a low training loss. We save the model only if the validation loss is decreasing

Step 5: Final Algorithm

An algorithm to accept an image and -

- Detect if its a human. If its human return closest resembling breed
- Detect if its a dog, and return the breed prediction
- Indicate error if its neither dog nor human

REFINEMENT

The first model built from scratch has an accuracy of 12% with 20 epochs. This could be further improved with hyperparameters tuning, changing the architecture of the model and also increasing the number of epochs.

I have chosen a pre-trained model (ResNet50) as the final model for the classification. The number of outputs needed to be changed to 133 (number of breeds) and the model had comparatively very less training/validation loss after training only for 10 epochs. It gave an accuracy of 80% which is far better than the accuracy of the model built from scratch.

MODEL EVALUATION AND VALIDATION

Human detector

This is a pre-trained model and does very well. The accuracy should be close to 100%. I have tested with 100 images and it could correctly identify 98 humans which is very good.

Dog detector

This is also a pre-trained model. This returned a 100% accurate results for the dog data set. However, it also predicted 4 humans to be dogs. But, that is fine as the model is doing good enough for our classification problem.

Breed classifier (from scratch)

This gave an accuracy of 12%, which is also good enough as a random guess has a 1/133 chance of being correct which is less than 1%. But, the model from scratch is already doing quite well with 12% accuracy. This can be further fine-tuned by adjusting learning rates, optimizers, loss functions.

Breed classifier (transfer learning)

On using the ResNet50 model the accuracy was 80% with 10 epochs.

JUSTIFICATION

The final model (resnet50 transfer learning) is a good choice as we see a drastic improvement in accuracy from 12% to 80%.

This could be further fine-tuned by increasing epochs, adjusting learning rate, trying out different optimizers, and loss functions