UDACITY MACHINE LEARNING PROJECT CAPSTONE

June 2021

1. Project overview

The project goal is to build a Dog Breeds Classifier app that uses an image classification deep learning model to perform the Dog breed identification. To achieve this goal is necessary to review different models based on CNN architectures (Convolutional Neuronal Networks) mainly. The project development includes to try models built from scratch and models using transfer learning approach on pretrained image classification architectures. The main tool used to develop the project is Pytorch and the models available in the torchvision.models library.

2. Project description

Project starts with a data exploration to review the image data organization and classes defined. Data formats need to be transformed to fit the pretrained model's requirements building data loaders function that process de images and add data augmentation to improve the training process. Different libraries and frameworks are suggested to use for the task required in the app. Face detection, Dog's detection, and Dog Breed´s detection. The app to be developed must me perform these tasks:

- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither is detected in the image, provide output that indicates an error.

Some models are suggested by the Udacity Team, but other models will be considered and tested to achieve best app performance.

3. Data exploration

Dataset used for the training process consist in face images and dog's images classified by name and dog breed, respectively. There are 13.233 human images and 8.351 dog images. For the Dogs breed classifier, the dog images provided by Udacity are organized by train, test and validation sets as subfolders.

001.Affenpinscher
002.Afghan_hound
003.Airedale_terrier
004.Akita
005.Alaskan_malamute
006.American_eskimo_dog
007.American_foxhound
008.American_staffordshire_terrier
009.American_water_spaniel
010.Anatolian_shepherd_dog
011.Australian_cattle_dog
012.Australian_shepherd
013.Australian_terrier
014.Basenji
015.Basset_hound
016.Beagle
017.Bearded_collie
018.Beauceron
019.Bedlington_terrier
020.Belgian_malinois
021.Belgian_sheepdog
022.Belgian_tervuren
023.Bernese_mountain_dog
024.Bichon_frise

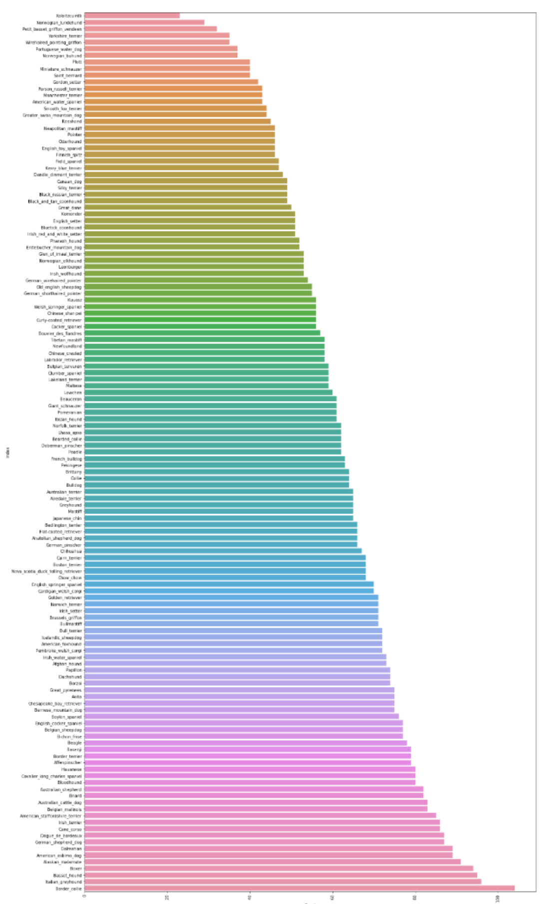
056.Dachshund
057.Dalmatian
058.Dandie_dinmont_terrier
059.Doberman_pinscher
060.Dogue_de_bordeaux
061.English_cocker_spaniel
062.English_setter
063.English_springer_spaniel
064.English_toy_spaniel
065.Entlebucher_mountain_dog
066.Field_spaniel
067.Finnish_spitz
068.Flat-coated_retriever
069.French_bulldog
070.German_pinscher
071.German_shepherd_dog
072.German_shorthaired_pointer
073.German_wirehaired_pointer
074.Giant_schnauzer
075.Glen_of_imaal_terrier
076.Golden_retriever
077.Gordon_setter
078.Great_dane
079.Great_pyrenees

111.Norwich_terrier
112.Nova_scotia_duck_tolling_retriever
113.Old_english_sheepdog
114.Otterhound
115.Papillon
116.Parson_russell_terrier
117.Pekingese
118.Pembroke_welsh_corgi
119.Petit_basset_griffon_vendeen
120.Pharaoh_hound
121.Plott
122.Pointer
123.Pomeranian
124.Poodle
125.Portuguese_water_dog
126.Saint_bernard
127.Silky_terrier
128.Smooth_fox_terrier
129.Tibetan_mastiff
130.Welsh_springer_spaniel
131.Wirehaired_pointing_griffon
132.Xoloitzcuintli
133.Yorkshire_terrier

025.Black_and_tan_coonhound
026.Black_russian_terrier
027.Bloodhound
028.Bluetick_coonhound
029.Border_collie
030.Border_terrier
031.Borzoi
032.Boston_terrier
033.Bouvier_des_flandres
034.Boxer
035.Boykin_spaniel
036.Briard
037.Brittany
038.Brussels_griffon
039.Bull_terrier
040.Bulldog
041.Bullmastiff
042.Cairn_terrier
043.Canaan_dog
044.Cane_corso
045.Cardigan_welsh_corgi
046.Cavalier_king_charles_spaniel
047.Chesapeake_bay_retriever
048.Chihuahua
049.Chinese_crested
050.Chinese_shar-pei
051.Chow_chow
052.Clumber_spaniel
053.Cocker_spaniel
054.Collie
055.Curly-coated_retriever

080.Greater_swiss_mountain_dog
081.Greyhound
082.Havanese
083.Ibizan_hound
084.Icelandic_sheepdog
085.Irish_red_and_white_setter
086.Irish_setter
087.Irish_terrier
088.Irish_water_spaniel
089.Irish_wolfhound
090.Italian_greyhound
091.Japanese_chin
092.Keeshond
093.Kerry_blue_terrier
094.Komondor
095.Kuvasz
096.Labrador_retriever
097.Lakeland_terrier
098.Leonberger
099.Lhasa_apso
100.Lowchen
101.Maltese
102.Manchester_terrier
103.Mastiff
104.Miniature_schnauzer
105.Neapolitan_mastiff
106.Newfoundland
107.Norfolk_terrier
108.Norwegian_buhund
109.Norwegian_elkhound
110.Norwegian_lundehund

There are 133 breeds categories where Border Collie is the most frequent class and Xoloitscuintli the less one.



Images are in RGB format and different sizes. These are an images sample:

4. Metrics

Metric used for model training was accuracy to assess the model classification quality. The accuracy formula is:

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

This is a percentage between 0 to 1. Higher values mean better classification quality of the model.

5. Data Pre-processing

The input images have different sizes, for that reason is necessary do some pre-processing. Using torchvision.transforms from PyTorch framework these transformation were applied to the data input:

- transforms.Resize(IMAGE_RESIZE): Resized the images to 256 px as is required by the Image classification Models.
- transforms.RandomRotation(28): Apply a random rotation between 0-28 grades
- transforms.RandomHorizontalFlip(): Generate random horizontal flip for data augmentation
- transforms.CenterCrop((IMAGE_SIZE, IMAGE_SIZE)): Center cropped to 224 px
- transforms.Normalize: Apply normalizarion with 0.5 mean and 0.5 std

6. Reference architectures

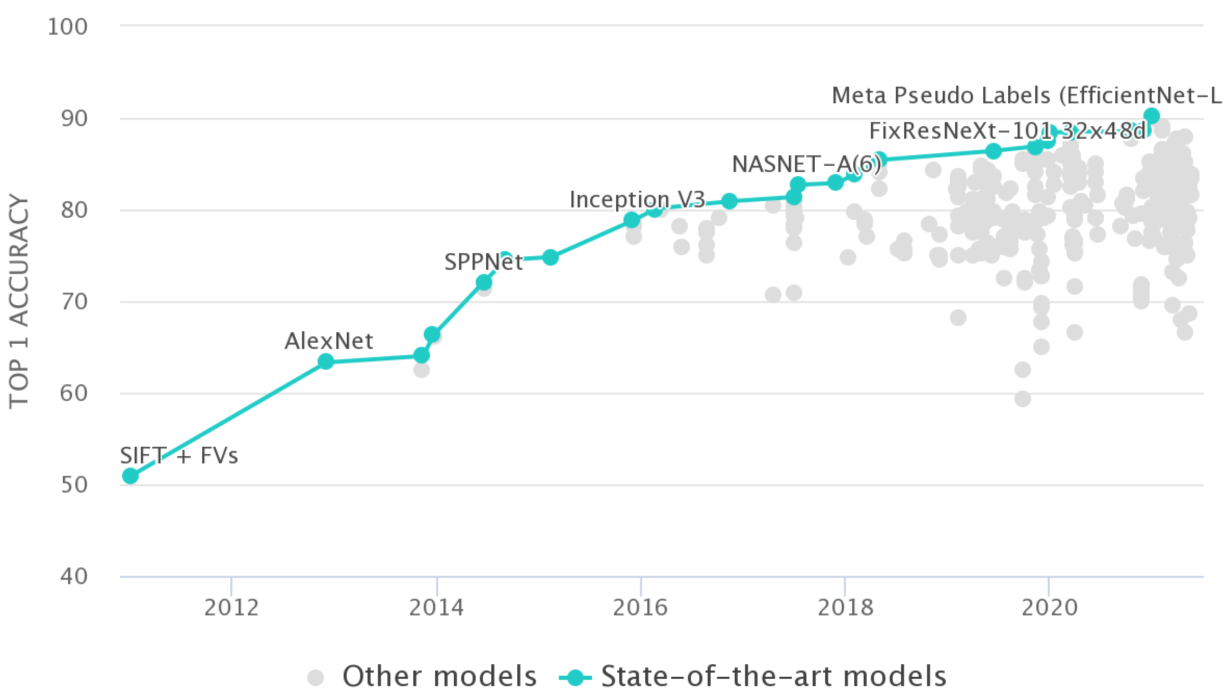
Reviewing the state-of-the-art image classification models, the site presents the recent models proposed and the accuracy achieved using the ImageNet dataset. https://paperswithcode.com/sota/image-classification-on-imagenet. The models available in the standard torchvision.models (Last Torchvision version 0.8):

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet

Models available that perform better are:

- **resnext101_32x8d:** The model is the same as ResNet except for the bottleneck number of channels which is twice larger in every block.
- **resnet152:** Residual Networks with 152 layers.
- **Inception v3:** convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with the use of batch normalization for layers in the side head).

Resnext101 uses over 80 million parameters, Inception v3 require over 20 million parameters, but state-of-the-art that performs better on the Imagenet dataset uses over 600 million parameters, that can use more time, inclusive for retraining the last layer for transfer learning. For that reason, I am reviewing the models pre-trained in Torhcvision and validate that best performing model is resnext101_32x8d, but this one is only available in torchvision 0.8.



Models with to performance are:

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year
1	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels			2021
2	**Meta Pseudo Labels** (EfficientNet-B6-Wide)	90%	98.7%	390M	✓	Meta Pseudo Labels			2021
3	**NFNet-F4+**	89.2%		527M	✓	High-Performance Large-Scale Image Recognition Without Normalization			2021
4	**ALIGN** (EfficientNet-L2)	88.64%	98.67%	480M	✓	Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision			2021
5	**EfficientNet-L2-475** (SAM)	88.61%		480M	✓	Sharpness-Aware Minimization for Efficiently Improving Generalization			2020
6	**ViT-H/14**	88.55%		632M	✓	An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale			2020
7	**FixEfficientNet-L2**	88.5%	98.7%	480M	✓	Fixing the train-test resolution discrepancy: FixEfficientNet			2020
8	**NoisyStudent** (EfficientNet-L2)	88.4%	98.7%	480M	✓	Self-training with Noisy Student improves ImageNet classification			2020
9	**Mixer-H/14** (JFT-300M pre-train)	87.94%			✓	MLP-Mixer: An all-MLP Architecture for Vision			2021

resnext101_32x8d looks like the best performance model in the list of available models, but still far from the lasted state-of-the-art models. Despite requires over 80 million parameters. I am trying to use this model for the transfer learning process. I decided to use resnet_150 because the number of parameters still low and is better for the resources available. This model requires image input with size 224x224 and has up to 2040 output classes. I also added to layers more for the transfer learning.

7. Model experimentation from scratch

First, I read some references about CNN for image classification with some examples and recommendations. The main reference used was: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/. Some recommendation is to define different models with different complexity (Layer numbers and pooling operations) and check for normalization, regularization, and dropout to improve results and avoid overfitting. All models get an input of 224 x 224 x 3 (Image size and RGB channels) and output for 133 classes. I defined three different models:

Base model (Small)

This is the base model, which has basic convolutional layers and a fully connected layer to map the tensors to the output size.

- 3 convolutional layers
- Max pooling
- Batch normalization
- Dropout
- 2 fully connected layers

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 112, 112]	896
MaxPool2d-2	[-1, 32, 56, 56]	0
Conv2d-3	[-1, 64, 28, 28]	18,496
MaxPool2d-4	[-1, 64, 14, 14]	0
Conv2d-5	[-1, 32, 14, 14]	18,464
MaxPool2d-6	[-1, 32, 7, 7]	0
Dropout-7	[-1, 1568]	0
Linear-8	[-1, 133]	208,677
BatchNorm1d-9	[-1, 133]	266

- Total params: 910,437
- Trainable params: 910,437
- Non-trainable params: 0

For this model was used 30 epochs

Medium size model

This model a couple more convolutional layers trying to improve the classification task and add batch normalization between convolutional layers.

- 5 convolutional layers
- Max pooling
- Batch normalization
- Dropout
- 3 fully connected layers

Layer (type)	Output Shape	Param #

Conv2d-1	[-1, 128, 112, 112]	3,584
BatchNorm2d-2	[-1, 128, 112, 112]	256
MaxPool2d-3	[-1, 128, 56, 56]	0
Conv2d-4	[-1, 64, 56, 56]	73,792
BatchNorm2d-5	[-1, 64, 56, 56]	128
Conv2d-6	[-1, 32, 56, 56]	18,464
MaxPool2d-7	[-1, 32, 28, 28]	0
Conv2d-8	[-1, 64, 28, 28]	18,496
MaxPool2d-9	[-1, 64, 14, 14]	0
Conv2d-10	[-1, 32, 14, 14]	18,464
MaxPool2d-11	[-1, 32, 7, 7]	0
Dropout-12	[-1, 1568]	0
Linear-13	[-1, 784]	1,230,096
Dropout-14	[-1, 784]	0
Linear-15	[-1, 382]	299,870
Dropout-16	[-1, 382]	0
Linear-17	[-1, 133]	50,939
BatchNorm1d-18	[-1, 133]	266

- Total params: 1,714,355
- Trainable params: 1,714,355
- Non-trainable params: 0

Big size model

This mode adds more convolutional layer trying to capture more image characteristics but increase the number of the parameters over 3 million making the model take more than 10 hours to be trained. The model is basically like the base model but adds 7 more convolutional layers.

- 8 convolutional layers
- Max pooling
- Batch normalization
- Dropout
- Fully connected layers

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
BatchNorm2d-2	[-1, 64, 224, 224]	128
Conv2d-3	[-1, 128, 224, 224]	73,856
BatchNorm2d-4	[-1, 128, 224, 224]	256
Conv2d-5	[-1, 128, 224, 224]	147,584
MaxPool2d-6	[-1, 128, 112, 112]	0
Conv2d-7	[-1, 256, 112, 112]	295,168
BatchNorm2d-8	[-1, 256, 112, 112]	512
Conv2d-9	[-1, 256, 112, 112]	590,080
MaxPool2d-10	[-1, 256, 56, 56]	0
Conv2d-11	[-1, 128, 56, 56]	295,040
MaxPool2d-12	[-1, 128, 28, 28]	0
Conv2d-13	[-1, 64, 28, 28]	73,792
MaxPool2d-14	[-1, 64, 14, 14]	0
Conv2d-15	[-1, 32, 14, 14]	18,464
MaxPool2d-16	[-1, 32, 7, 7]	0
Linear-17	[-1, 784]	1,230,096
Dropout-18	[-1, 784]	0

Linear-19	[-1, 392]	307,720
Dropout-20	[-1, 392]	0
Linear-21	[-1, 133]	52,269
BatchNorm1d-22	[-1, 133]	266

- Total params: 3,087,023
- Trainable params: 3,087,023
- Non-trainable params: 0

For this model, the result running only 20 epochs executed due to the long training time required.

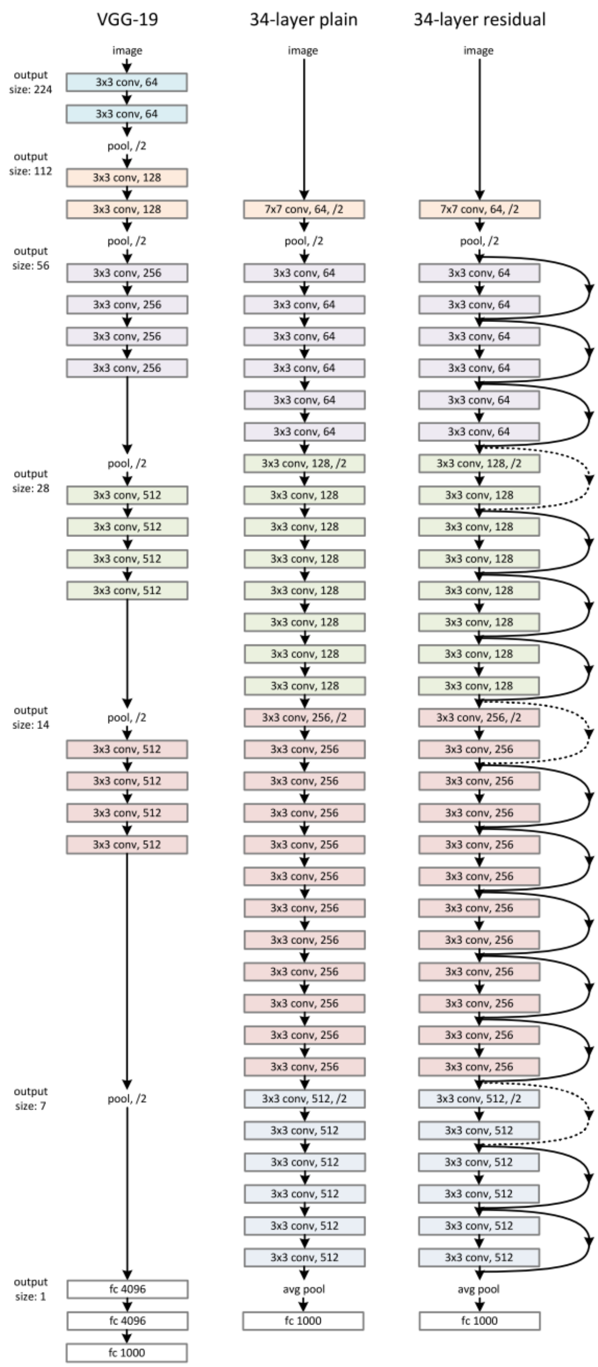
After try the different models, there results were obtained:

Model	Test Loss	Parameters number	Epochs	Test Accuracy
Small size	3.232469	246,799	50	21% (183/836)
Medium size	3.131659	1,714,355	50	25% (215/836)
Big size	3.917502	3,087,023	20	11% (94/836)

8. Model implementation using Transfer Learning.

To improve the results, transfer Learning approach was used using pretrained models provide by torchvision library. I tried different models like Resnet-50 and Inception v2 but the accuracy was not the best and moved to Resbet152. I needed to add an additional layer for the transfer learning and with different Learning rates and optimizers. Finally, the best performance model achieved over 85% accuracy with 15 epochs.

The resnet152 architecture has 59,261,125 training parameters and looks like this:



Two fully connected layers were added to the model to perform the final classification with the breed samples. Relu activation functions was used and dropout regularization with 0.3.

```python
import torchvision.models as models
import torch.nn as nn
from collections import OrderedDict


## TODO: Specify model architecture
INPUT_FEATURES = 2048
OUPUT_FEATURES = 133

# check if CUDA is available
use_cuda = torch.cuda.is_available()

# Load the model from torch models to perform the transfer learning using inception_v3
model_transfer_learning = models.resnet152(pretrained=True)

# freeze all model parameters
for parameter in model_transfer_learning.parameters():
    parameter.require_grad = False

# Get the input features size for the model resnet 152
input_features = model_transfer_learning.fc.in_features

# This is recommended to run deep models to stabilize the training process
# https://discuss.pytorch.org/t/why-auxiliary-logits-set-to-false-in-train-mode/40705
model_transfer_learning.aux_logits=False


# Define the additional layer for the transfer learning process
model_transfer_learning.fc = nn.Sequential(OrderedDict(
                                [
                                  ('fc1', nn.Linear(INPUT_FEATURES, 512)),
                                  ('relu', nn.ReLU()),
                                  ('dropout', nn.Dropout(p=0.3)),
                                  ('fc2', nn.Linear(512, OUPUT_FEATURES))
                                ]))

# Verify if cuda GPU is available
if use_cuda:
    model_transfer_learning = model_transfer_learning.cuda()
```

Loss Function and Optimizer

```python
# criterion_transfer = None
# optimizer_transfer = None

# Define the learning rate parameter
LEARNING_RATE = 0.02

# Define the Loss function
criterion_transfer_learning = nn.CrossEntropyLoss()

# Define the optimizer
# Adma optimizer
optimizer_transfer_learning = optim.SGD(model_transfer_learning.fc.parameters(),lr=LEARNING_RATE)
```

9. App implementation and results

With the final model trained using transfer learning, a prediction function was implemented to use in the final app.

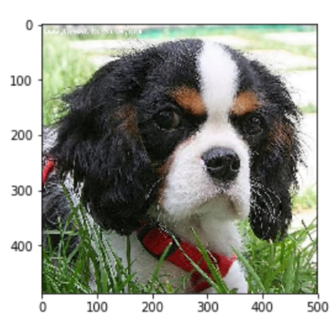
Prediction function:

```
def predict_breed_transfer(img_path):
    # load the image and return the predicted breed

    # Open the image related to he img_path
    img=Image.open(img_path).convert("RGB")

    # Define image transformation
    img_transform = transforms.Compose([transforms.Resize(256),
                                        transforms.CenterCrop(224),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                             std=[0.229, 0.224, 0.225])
                                       ])


    # Create the reshaped image tensor
    image_tensor = img_transform(img).unsqueeze(0)

    # Validate the cuda availability
    if use_cuda:
        image_tensor = image_tensor.cuda()

    # Performing inference
    model_transfer_learning.eval()

    # Execute the inference and apply softmax function and extract the final value
    output = F.softmax(model_transfer_learning(image_tensor),dim=1).cpu().data.numpy().squeeze()

    return class_names[np.argmax(output)]
```

Testing function:

```
# Test the function
dog_files_test = np.array(glob("/data/dog_images/test/*/*"))
image_index = int(np.random.randint(0,len(dog_files_test),1))
breed_detected = predict_breed_transfer(dog_files_test[image_index])
print("Image file Input : {}",dog_files_test[image_index])
image = Image.open(dog_files_test[image_index])
plt.imshow(image)
plt.show()
print('Predicted dog breed: {}', breed_detected)
```

```
Image file Input : {} /data/dog_images/test/046.Cavalier_king_charles_spaniel/Cavalier_king_charles_spaniel_03258.jpg
```




```
Predicted dog breed: {} Cavalier king charles spaniel
```

Final App implementation

```python
def run_app(img_path):
    '''
    Define a function to execute inference process to detect faces and fogd bred
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to RESNET-120 model's prediction
    '''

    ## handle cases for a human face, dog, and neither

    # First look for human face
    if face_detector(img_path):
        print("Face Detected")
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
        print(f"Looks like: \n The {predict_breed_transfer(img_path)}\n")

    # If not face detectec check for dog breed
    elif dog_detector(img_path):
        print("Dog Detected")
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
        print(f"Predicted breed:... \n{predict_breed_transfer(img_path)}\n")

    # Not face or dog detected
    else:
        print("Couldn't detect dogs or faces.\n")
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
```

Results

Example 1

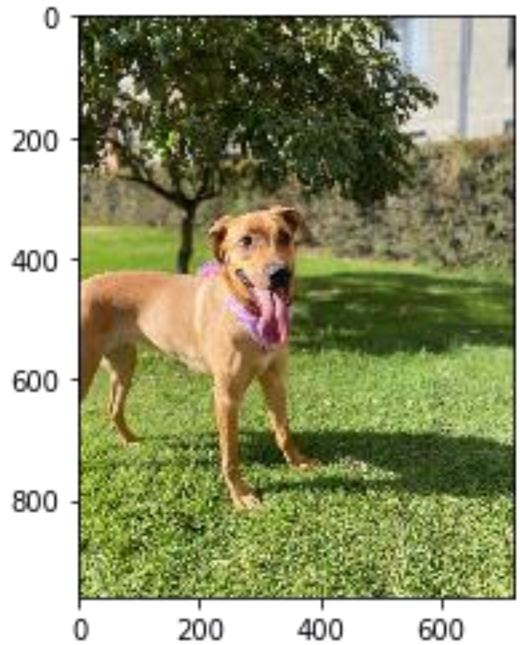
```
Face Detected
```


```
Looks like:
 The Anatolian shepherd dog
```

In this case the classifier failed, detecting a human face from the dog picture.

Example 2

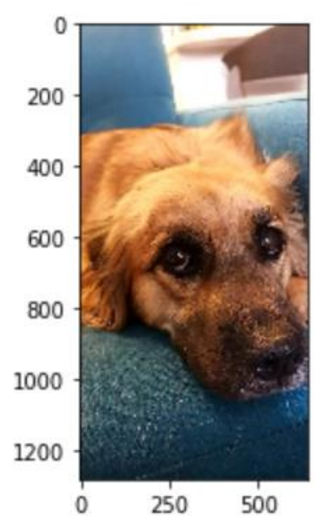
Dog Detected



Predicted breed:...
Canaan dog

Example 3

Dog Detected



Predicted breed:...
Cocker spaniel

Example 4

Couldn't detect dogs or faces.

Bad luck for the Grinch with the breed dog classifier

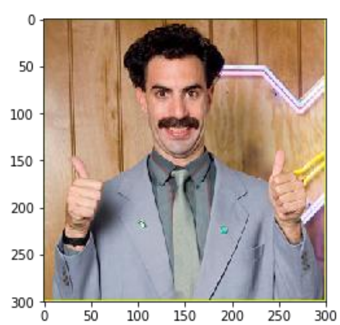
Example 5

Dog Detected

Predicted breed:...
Lowchen

Example 6

Face Detected



Looks like:
The Black russian terrier
Smart classifer!

Predicted breed:...
Lhasa apso

My Shih tzu doggy coco is sad for my classifier results!

10. Conclusion

The model trained using transfer learning with resnet152 pre-trained model achieved around 85% accuracy. I tried with some additional images, for example, two ShihZu images, and the model does not work well. and with dogs with mixed bred the model was confused. That means that there is a lot of improvement because 85% accuracy sounds ok but in practice still far from a good performance. Is clear that using a pre-trained model offers a fast and better way to train models for a specific task like dog breed detection. As I suggested when chose the pre-trained model, there more state-of-the-art models with more than 600 million parameters.

Possible improvements

1. As was reviewed before, state-of-the-art models offer higher accuracy for the image classification tasks, can also provide better performance for the transfer learning task. These models require more resources for the training process.
2. Aditional optimizations can be used to improve performance, like hyperparameter tuning and network pruning to accelerate inference time response.

3. Increasing data with more sources using web scrapping and data augmentation using GAN networks to enrich the training process can provide better input for the training and validation process.
4. New models based on a generative approach can be used to improve the transfer learning process.