Data Scientist Nanodegree

Convolutional Neural Networks

Project part 1: Write an Algorithm for a Dog Identification App

With this project I will finish my Udacity nanodegree in “Data Scientist”, sponsored by Misk & Sdaia in Saudi Arabia.
This is a real world capstone project, one of the projects proposed by Udacity. Some startup code has been provided. In this writeup I will go through it and clarify the most important parts of it.

Project Definition

The dog breed classifier project is a Deep Learning project that uses Convolutional Neural Networks (CNN) with the objective to create a web application where the user supplies images and the algorithm should detect human faces or dogs that are in those images. If a dog is found, it returns its most probable breed class and if a human face is detected it tells what its most resembling dog breed is.
The startup project provided by Udacity is to be found here. But I have been working in my own environment with the following requirements:

- ipython==8.0.1
- keras==2.7.0
- matplotlib==3.4.3
- numpy==1.20.3
- opencv_python==4.5.5.62
- pandas==1.3.4
- Pillow==9.0.1
- scikit_learn==1.0.2
- tensorflow==2.7.0
- tqdm==4.62.3

Problem statement

The objective is to go through a series of processes to read an image and using Tensorflow, computer vision and CNN models to achieve the highest accuracy possible to classify images of humans and dogs. And to be able to run the model with reasonable computer resources while

attaining a test accuracy of a minimum of 60% Transfer Learning will be used to create a CNN model.
In this project a test accuracy of about 79% is reached using the Inception model. This accuracy can be definitely improved through implementation of eg. augmentation and changing model parameters.

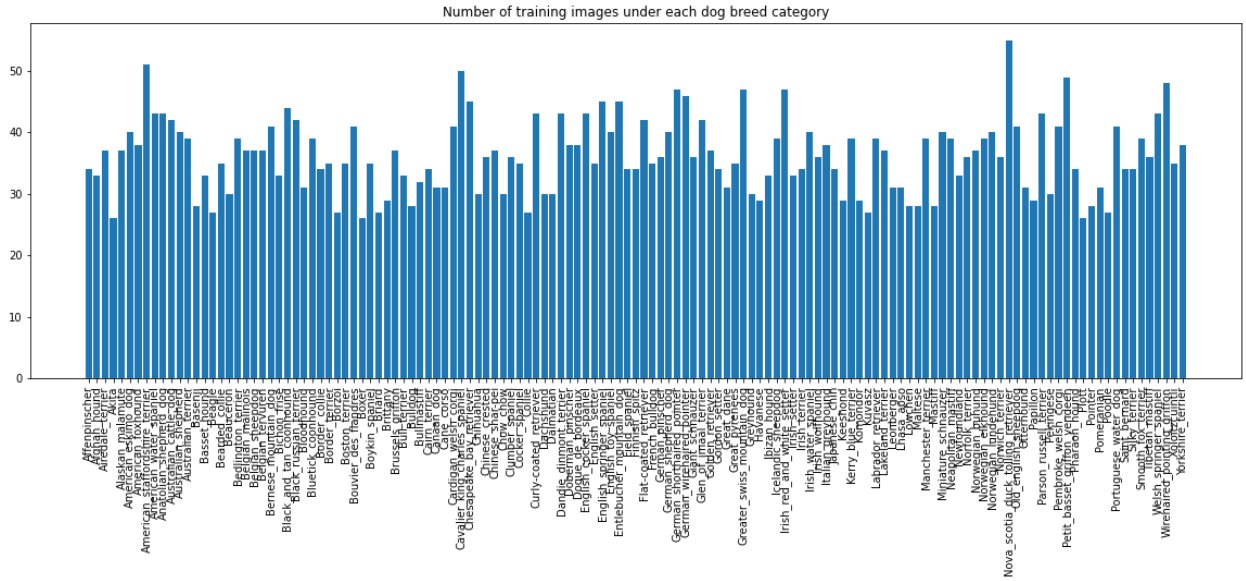
Metrics

Assigning a breed to dogs from images is considered exceptionally challenging even for humans because of the great difficulty in distinguishing between many of the breeds. For example, a Brittany and a Welsh Springer Spaniel are hard to be distinguished:

Also these two breeds (Curly-Coated Retrievers and American Water Spaniels) are for me the same:

Besides, when looking for human faces in dog images 11% of images are seen as humans using [Haar feature-based cascade classifiers](#).
In this multi-class classification project with 133 dog breeds a test accuracy of the “from scratch” created CNN model resulted in 1.07% test accuracy while the pre-trained VGG-16 model which we used as is with only adding a global average pooling layer and a dense layer with 133 classes, thus using “categorial_crossentropy” as a loss function and “accuracy” as a metric, a result of about 40% of test images are predicted properly.

Using accuracy as metric is reasonable here because most categories contain roughly 40 images per each meaning there is an acceptable balance between them



Data Exploration and visualization

The Dog Dataset (dogs images) that are used is to be downloaded from
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
There are 133 total dog categories with a total of 8351 dog images, split as follows:

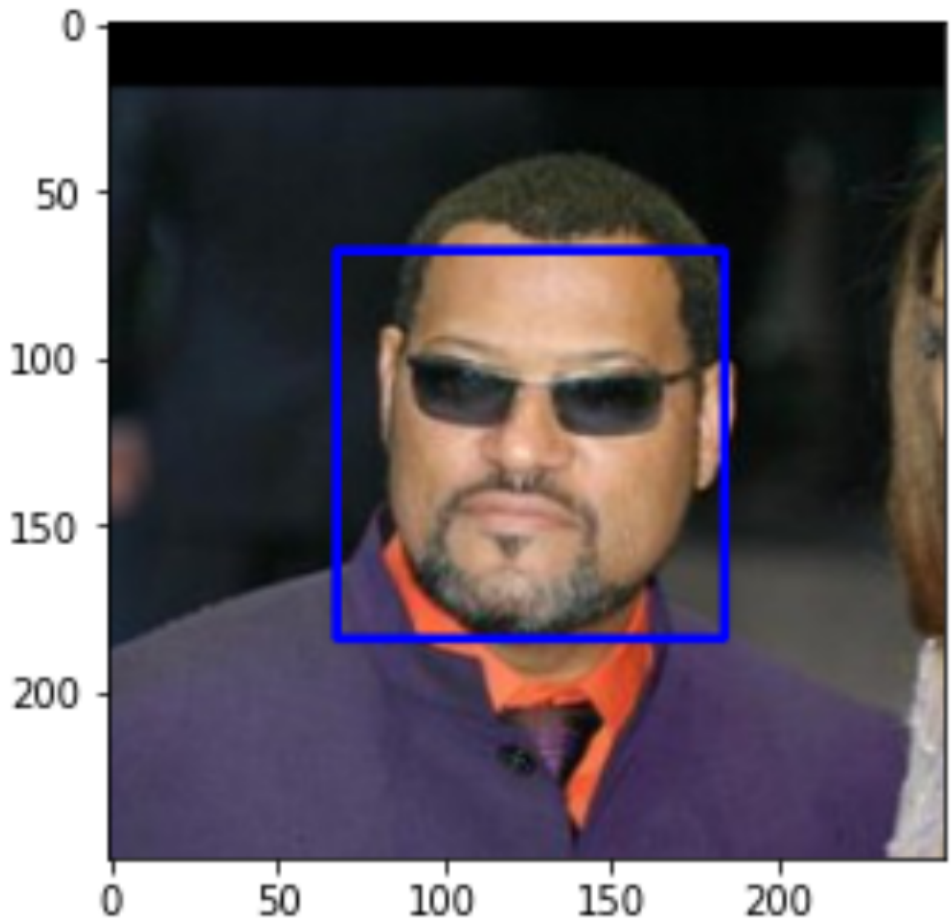
- 6680 training dog images.
- 835 validation dog images.
- 836 test dog images.

The Humans Dataset which contains 13233 images is found here:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

Data Preprocessing

In detecting humans a detector from “haarcascades” directory in Haar feature-based cascade classifiers is used to count the number of faces and draw a bounding box for each detected face. E.g.:

```
Number of faces detected: 1
```



When assessing the human face detector 100.0 % of the first 100 images in human images had a human face detected in them while 11.0 % of the first 100 images in dog_files had a human face detected in them. This is an acceptable performance for this project but we could look for other face detectors using deep learning and OpenCV.

For detecting dogs in images a pre-trained ResNet-50 model is used (to be found here) along with the weights trained on ImageNet. The ResNet-50 model returns a prediction for the object in the image based on the 1000 categories in ImageNet.

To use Keras CNN images should be converted to 4D tensor as follows (nb_samples,rows,columns,channels), where nb_samples is the number of images, rows and

columns are the 2 dimensional sizes of the image and channels are the 3 colors (RGB). RGB is re-ordered by the function preprocess_input found here to BGR
In this dictionary, the categories in imageNet corresponding to dogs are keyed from 151 to 268 which gives us 133 categories and using ResNet-50 model labels a dog detector is created to return True if the label predicted is between 151 and 258.

The results are: 0.0 % of the first 100 images in humans images had a dog detected in them while 100.0 % of the first 100 images in dogs images had a dog detected in them.

For the implementation of a CNN model (next) the images are rescaled by dividing every pixel in every image by 255

Implementation

As a first step a CNN simple model from scratch is created with the following summary:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_758 (Conv2D)         (None, 224, 224, 16)      208

 max_pooling2d_38 (MaxPoolin  (None, 112, 112, 16)     0
 g2D)

 dropout_5 (Dropout)         (None, 112, 112, 16)      0

 conv2d_759 (Conv2D)         (None, 112, 112, 32)      2080

 max_pooling2d_39 (MaxPoolin  (None, 56, 56, 32)       0
 g2D)

 dropout_6 (Dropout)         (None, 56, 56, 32)        0

 conv2d_760 (Conv2D)         (None, 56, 56, 64)        8256

 max_pooling2d_40 (MaxPoolin  (None, 28, 28, 64)       0
 g2D)

 dropout_7 (Dropout)         (None, 28, 28, 64)        0

 global_average_pooling2d_4   (None, 64)               0
 (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 133)               8645

=================================================================
Total params: 19,189
Trainable params: 19,189
Non-trainable params: 0
```

In this architecture the following steps were realized:

- 3 Conv2D layers were used to deal with 2D input matrices (images).
- the 3 layers have 16,32,64 filters respectively,
- each filter is a 2 by 2 matrix specified with kernel-size.
- Relu is the activation function to deal with nonlinearity in a neural network.
- Padding is added to ensure the output feature matches the input.
- Max pooling layer is added to each conv to reduce number of cells in filters, here a 2 by 2 matrix for all features is used

- data-format specifies the order of tensor parameters, here the channels are last
- at last we use GlobalAveragePooling2D to default the size to input size
- dropouts of 10% are used to avoid overfitting in each conv layer

With CNN it is much faster to train and avoid overfitting. It also doesn't need a lot of training data

With ModelCheckpoint the model that attains the best validation loss is saved and then used on the test dog images which resulted in a test accuracy rounded to less than 1% when I had dropouts of 25%. Because it is requested that accuracy should exceed 1% I changed the percentage of dropouts to 10% and retrieved an accuracy which is 1.07%

Refinement

In second step Transfer Learning is used to create a CNN model using DogVGG16Data.npz found here: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/DogVGG16Data.npz

Transfer learning enables us to use pre-trained models trained on ImageNet data for other tasks like our dog-breed classification.Here, we retain the layers corresponding to the feature extraction part from the pre-trained model and train only the final classification layers. This way, less computational resources are used within a short processing time.

The following architecture is used for our model which resulted in a test accuracy of 67.58%

```
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dense(133, activation='softmax'))

VGG16_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 global_average_pooling2d_1   (None, 512)              0
 (GlobalAveragePooling2D)

 dense_1 (Dense)             (None, 133)               68229

=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

In the third step another pre-trained model was used; 4 options were proposed. I've worked with Inception bottleneck features pre-computed by Udacity.

My architecture in this model is:

```
### TODO: Define your architecture.
InceptionV3_model = Sequential()
InceptionV3_model.add(GlobalAveragePooling2D(input_shape=(train_InceptionV3.shape[1:])))
InceptionV3_model.add(Dense(133, activation='softmax'))
InceptionV3_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 global_average_pooling2d_2   (None, 2048)             0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 133)               272517

=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

I've added a Global Average layer and a Dense layer with "softmax" as an activation function.

The model weights with the best validation loss are saved at the epoch with: "loss: 0.3776 - accuracy: 0.8861 - val_loss: 0.6388 - val_accuracy: 0.8335". It is used on test images to end up with a test accuracy of 81.22%

Results

The test accuracy that is achieved is way ahead of the requested accuracy of 60%. While the model can still be improved by using augmentation for example.

In the last step a function is created to read an image and predict whether there are humans or not in it and tell the most resembling dog breed for the detected human or dog:

```
def identify_dog_app(img_path):
    print ("Hello, let's identify your images of humans and dogs!")
    display(Image(img_path,width=200,height=200))
    breed = InceptionV3_prediction_breed(img_path)
    if dog_detector(img_path):
        print("This can be a dog and its breed can be: {}\n".format(breed))
    elif face_detector(img_path):
        print("This can be a human but looks like the dog breed: {}\n".format(breed))
    else:
        print("I have no idea what this is. Please upload another image! \n")
```

A sample output is:

```
Hello, let's identify your images of humans and dogs!
```

```
This can be a human but looks like the dog breed: Dachshund
```

```
Hello, let's identify your images of humans and dogs!
```

```
This can be a dog and its breed can be: Airedale_terrier
```

This model can be improved by

- using augmentation by random transformation and rescaling of images
- having more layers and more nodes in the neural network will improve the model
- adding just enough training epochs

Model Evaluation and Justification

One of the objectives of this project is to get a sense of using different CNN models for prediction. And as stated above, the best model resulted to be the model that uses transfer learning and specifically “Inception V3”. As this increased the test accuracy and could easily run it on a local machine.

“Transfer Learning will be the next driver of ML success” — Andrew Ng

Using a CNN model from scratch: 1%
Transfer learning using pre-trained model “VGGNet”: 40%
Transfer learning using pre-trained model “Inception”: 81%

According to this article, VGGNet has a higher number of parameters and a decreased accuracy. It takes more time to train a VGGNet with reduced accuracy. While Inception has much fewer parameters and a higher accuracy.

Reflection and Improvement

I would like to work further on this project and use augmentation to get more images. Also adding more layers and better tuning the parameters using grid-search to improve the performance.
I wanted to use the Xception model for comparison but couldn’t download its features file for its big size.

Project part 2: Design a web app

Using Gradio I made a web app using the image classification app template with the following tweaks:

- Changed the default theme to dark-huggingface
- Added a second out label (to display 1) whether the image contains human faces or dogs and 2) to display the dog breed)
- Added an html block for acknowledgment
- Removed examples images
- Set the num_to_classes to 1, to display the most probable classification only

For the app.py to run the following requirements are needed:

- gradio==2.7.5.2

- keras==2.7.0
- numpy==1.20.3
- opencv_python==4.5.5.62
- tensorflow==2.7.0
- tqdm==4.62.3

If needed to run on huggingface.com, the package `libgl1-mesa-glx` should be installed.

The application runs perfect on my local machine, but trying to host it on huggingface at: https://huggingface.co/spaces/fatsam/myapp I miss the output labels, which I should find out soon on how to fix this error.

To run the application locally first create an environment and download the following files:

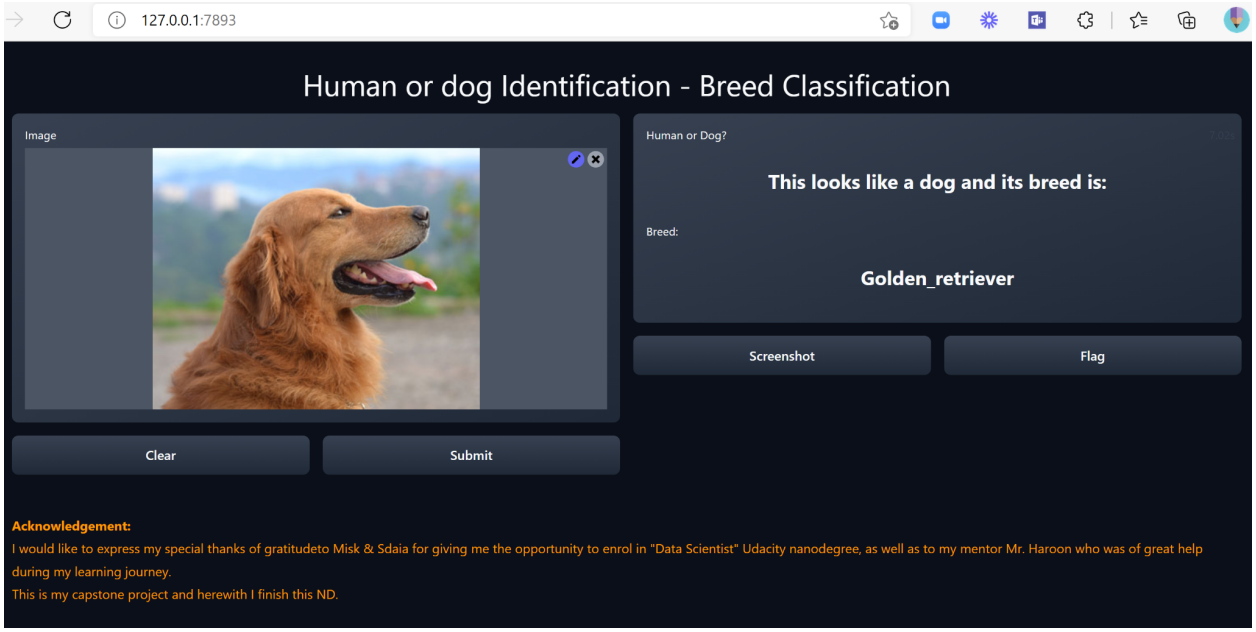
- app.py (contains the algorithm to return a gradio app all with functions)
- extract_bottleneck_features.py (contains the functions to extract bottleneck features from the models)
- haarcascade_frontalface_alt.xml (contains haarcascade classifier to detect human faces)
- requirements.txt (the libraries to install)
- weights.best.InceptionV3.hdf5 (saved model used to predict the dogs breed)

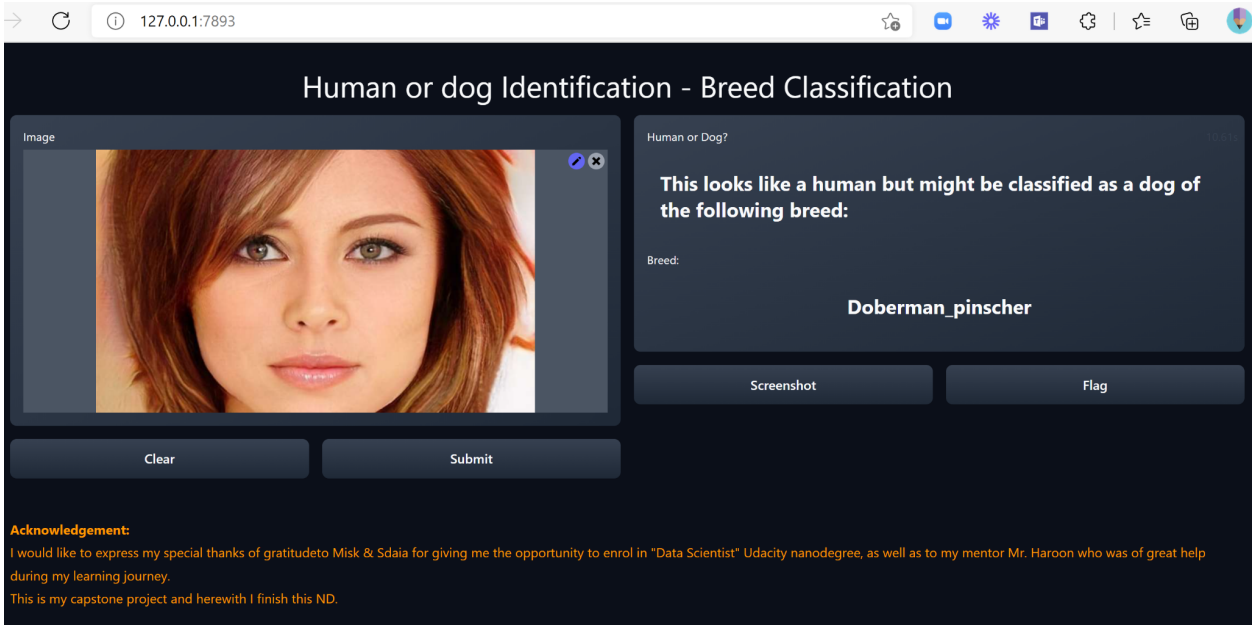
Adjust the file paths in app.py as needed.
Then install the requirements and run the app.py
A local URL will be returned to run the web application. A temporary public url is also provided if you set share=True in: *iface.launch(share=True)*

A sample output:

```
Running on local URL:  http://127.0.0.1:7893/
Running on public URL: https://41027.gradio.app
```

Your app will look like this:


127.0.0.1:7893
Human or dog Identification - Breed Classification
Image
Human or Dog?
This looks like a dog and its breed is:
Breed:
Golden_retriever
Screenshot
Flag
Clear
Submit
Acknowledgement:
I would like to express my special thanks of gratitudeto Misk & Sdaia for giving me the opportunity to enrol in "Data Scientist" Udacity nanodegree, as well as to my mentor Mr. Haroon who was of great help during my learning journey.
This is my capstone project and herewith I finish this ND.


127.0.0.1:7893
Human or dog Identification - Breed Classification
Image
Human or Dog?
This looks like a human but might be classified as a dog of the following breed:
Breed:
Doberman_pinscher
Screenshot
Flag
Clear
Submit
Acknowledgement:
I would like to express my special thanks of gratitudeto Misk & Sdaia for giving me the opportunity to enrol in "Data Scientist" Udacity nanodegree, as well as to my mentor Mr. Haroon who was of great help during my learning journey.
This is my capstone project and herewith I finish this ND.