I. Project Definition

Overview

One of, if not the most important, concept of a data science is the classification algorithm. You can predict the target class by analyzing the training dataset and optimize and optimize the whole thing to the desired result.

In this Udacity project we will develop a classifier that can predict the dog breed of a picture. We will do this by using different deep learning techniques. First of all let's face the problems.

Problem

The goal of the project is to determine the dog breed of an image. Based on the following premises a picture should be categorized:

- If a human is detected in the image, return the information that's it is a human and the most similar breed
- If a dog is detected in the image, return the information that it's a dog and the predicted breed.
- If it's neither a dog nor human, return the information that it must be something else.

Due to this requirements we can divide the project in different steps.

Metrics

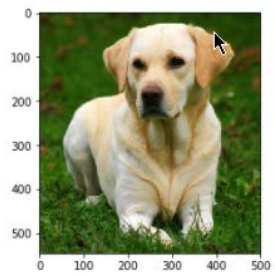
Target is a web application where the user can upload an image. By analyzing the picture the output should be the uploaded image as well as the following messages based on the above premises.

For the dog or face detector, the type of measurement is quite simple. The datasets in the dogs and dog images are used for both functions. Then you check in which images the respective detector has recognized what. In the ideal case, the face detector only recognizes a face in the human images and the dog detector does not recognize dogs in the same data set.

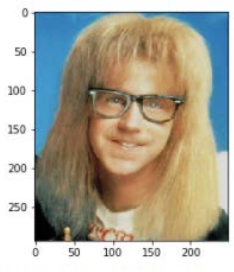
For the CNN we are checking different pre-trained networks by their accuracy in the test and validation dataset as well as the loss function.

- The loss function show the prediction of error in the CNN, the lower the loss the better.
- The test accuracy shows as how many percent of the images have been predicted right.

These two values and the overall test accuracy for each model give us a good overview of how well the respective model performed.



This is a dog and it's breed is Labrador retriever.



This is not a dog but it's look like a Cavalier king charles spaniel.

II. Analysis

Data Exploration and Visualization

Following dataset have been provided:

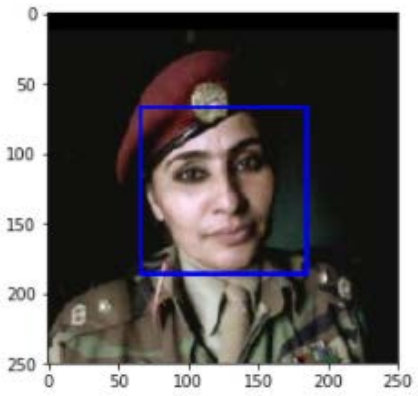
Dog images **i**n total 8351 images divided in datasets for:

- training with 6,680 images,
- validation with 835 images and
- testing with 836 images.

Dog names dataset contains 133 different dog breed names, we will use the as basis for the prediction.

Human images in total there are 13233 total human images for classify a human in a picture.

Haarcascades is pre-trained model to detect human faces in images



Bottleneck features will be used to find the best pre-trained model for our dog breed classification. In this case: VGG16, VGG19, Resnet50, InceptionV3 and Xception.

III. Methodology

Data preprocessing

Converting images

For further using the images with a Keras CNN, we need to take care that a function convert a image into a 4D tensor.

```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Implementation

Human detector

First step we are writing a human face detector with the help of “haarcascade”.

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

```
100.0% faces are detected in human files
11.0% faces are detected in dog files
```

By passing the training data for dogs and human images trough, we can see that in all human files faces are detected. On the negative side this happened also in 11% of the dog images. We keep that in mind, because we need to find a way to identify exactly if a dog or human is in the picture.

Dog detector

With the help of ResNet50 a pre-trained model weighted by “imagenet”. During investigate the dictionary we can enclose that dogs labeled between 151 and 268.

```
from keras.applications.resnet50 import preprocess_input, decode_predictions
from keras.applications.resnet50 import ResNet50

# define ResNet50 model
ResNet50_model = ResNet50(weights='imagenet')


def ResNet50_predict_labels(img_path):
    # returns prediction vector for image located at img_path
    img = preprocess_input(path_to_tensor(img_path))
    return np.argmax(ResNet50_model.predict(img))

def dog_detector(img_path):
    prediction = ResNet50_predict_labels(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

```
0.0%  are detected as dogs in human files
100.0%  are detected as dogs dog files
```

By passing the training data through the image converter and the dog detector we can identify a dog in 100% of the cases. Furthermore, no dogs were recognized in the human images.

This enables us to clearly identify a human face and a dog with the help of the two detectors. By first using the dog detector we avoid in advance that humans are wrongly recognized as dogs.

Dog breed classification

Because of the need of a CNN that can predict dog breed reliable, I started to create my own model architecture:

```
model = Sequential([
Conv2D(filters=16, kernel_size=(5,5), padding='same', activation='relu', input_shape=(224, 224, 3)),
MaxPooling2D(pool_size=2),
Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu'),
MaxPooling2D(pool_size=2),
Conv2D(filters=64, kernel_size=(5,5), padding='same', activation='relu'),
MaxPooling2D(pool_size=2),
Flatten(),
Dense(256, activation='relu'),
Dropout(0.5),
Dense(133, activation='softmax'),
])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      1216
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      12832
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        51264
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               12845312
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               34181
=================================================================
Total params: 12,944,805
Trainable params: 12,944,805
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

The models was trained in five epochs with a batch size of 20. The Test accuracy with a bit over 1% was unsatisfying.

Refinement

This is where transfer learning comes in.

We are loading a pre-training CNN that has already been trained on large data sets. Freeze the weighting, add individual layers and parameters that are necessary for our needs and can thus achieve a very good result in a shorter time. The following models have been extracted and weighted with ImageNet.

Where the last convolutional output of the respective model fed as input to our model. We only add a global average pooling layer and a fully connected layer, where the latter contains one node for each dog category and is equipped with a softmax.

```
# model architecture
print('...loading model architecture...')
trans_model = Sequential()
trans_model.add(GlobalAveragePooling2D(input_shape=train_bottle.shape[1:]))
trans_model.add(Dense(133, activation='softmax'))
```

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Layers
Xception	88 MB	0.790	0.945	22,910,480	71
VGG16	528 MB	0.713	0.901	138,357,544	16
VGG19	549 MB	0.713	0.900	143,667,240	19
ResNet50	98 MB	0.749	0.921	25,636,712	50
InceptionV3	92 MB	0.779	0.937	23,851,784	48

Source: https://keras.io/api/applications/

Example for ResNet 50

During the training and validation evertime the val_loss improved the model weights have been saved. If we decide to use a ResNet50 we can load the best weighted model.

```
...compiling...
...train model with Resnet50...
Train on 6680 samples, validate on 835 samples
Epoch 1/20
 - 1s - loss: 1.6501 - acc: 0.5915 - val_loss: 0.8110 - val_acc: 0.7425

Epoch 00001: val_loss improved from inf to 0.81104, saving model to saved_models/weights.best.Resnet50.hdf5
Epoch 2/20
 - 1s - loss: 0.4413 - acc: 0.8644 - val_loss: 0.6940 - val_acc: 0.7868

Epoch 00002: val_loss improved from 0.81104 to 0.69403, saving model to saved_models/weights.best.Resnet50.hdf5
Epoch 3/20
 - 1s - loss: 0.2607 - acc: 0.9186 - val_loss: 0.6573 - val_acc: 0.7940

Epoch 00003: val_loss improved from 0.69403 to 0.65733, saving model to saved_models/weights.best.Resnet50.hdf5
Epoch 4/20
 - 1s - loss: 0.1725 - acc: 0.9469 - val_loss: 0.7189 - val_acc: 0.7940
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_6 ( (None, 2048)              0
_________________________________________________________________
dense_6 (Dense)              (None, 133)               272517
=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.models import load_model
>>> model = load_model('saved_models/weights.best.Resnet50.hdf5')
>>> for layer in model.layers:
...     if len(layer.weights) > 0:
...         print(layer.name, layer.weights[0].shape)
```

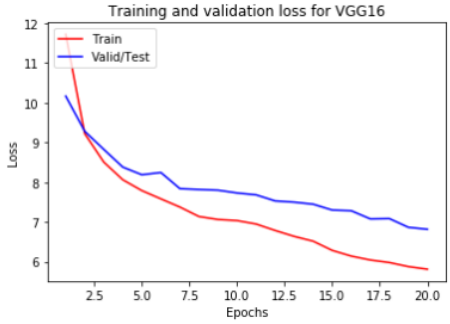
These are the saved weights for the ResNet 50 CNN: Dense_6 (2048, 133)

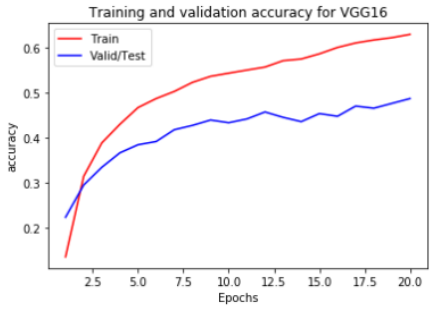
By downloading the bottleneck features running the models one after the other. I also decided to create two plots, each showing the “loss” and “val-loss” as well as “acc” and “vall acc” during the 20 epochs. The test accuracy is also determined

```
import time

# define testing fuction for the networks

def trans_net_test(network):

    # bottleneck features
    print('...loading bottleneck features...')
    bottleneck_features = np.load('bottleneck_features/Dog{}Data.npz'.format(network))

    # transfer bottleneck into train, test and validation
    train_bottle = bottleneck_features['train']
    test_bottle = bottleneck_features['test']
    valid_bottle = bottleneck_features['valid']

    # model architecture
    print('...loading model architecture...')
    trans_model = Sequential()
    trans_model.add(GlobalAveragePooling2D(input_shape=train_bottle.shape[1:]))
    trans_model.add(Dense(133, activation='softmax'))

    # model compiling
    print('...compiling...')
    trans_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

    #train model
    print('...train model with {}...'.format(network))
    checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.{}.hdf5'.format(network),
                                   verbose=1, save_best_only=True)


    hist_model = trans_model.fit(train_bottle, train_targets,
                validation_data=(valid_bottle, valid_targets),
                epochs=20, batch_size=20, callbacks=[checkpointer], verbose=2)

    #load best model
    print('...loading best model...')
    trans_model.load_weights('saved_models/weights.best.{}.hdf5'.format(network))

    # testing the model
    print('...testing model...')
    trans_predictions = [np.argmax(trans_model.predict(np.expand_dims(feature, axis=0))) for feature in test_bottle]

    #  test accuracy
    test_accuracy = 100*np.sum(np.array(trans_predictions)==np.argmax(test_targets, axis=1))/len(trans_predictions)
    print('Test accuracy: %.4f%%' % test_accuracy)

    #define plot for visualistion of the loss between test and train/validation datasets
    #def plotLoss(mdl):
    loss = hist_model.history['loss']
    val_loss = hist_model.history['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, color='red', label='Training loss')
    plt.plot(epochs, val_loss, color='blue', label='Validation loss')
    plt.title('Training and validation loss for {}'.format(network))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Valid/Test'], loc='upper left')
    plt.show()

    #plotloss(hist_model)

#define plot for visualistion of accuracy between test and train/validation datasets

    #def plotaccuracy(mdl):
    acc = hist_model.history['acc']
    val_acc = hist_model.history['val_acc']
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, color='red', label='Training acc')
    plt.plot(epochs, val_acc, color='blue', label='Validation acc')
    plt.title('Training and validation accuracy for {}'.format(network))
    plt.xlabel('Epochs')
    plt.ylabel('accuracy')
    plt.legend(['Train', 'Valid/Test'], loc='upper left')
    plt.show()



    return print('Test accuracy for {}: %.4f%%'.format(network) % test_accuracy)
```

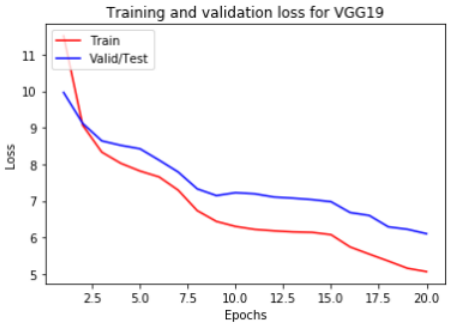
To check the robustness of the models we measure the loss function with categorical cross entropy. Also the metric accuracy is used. As an optimizer we are using rmsprop with the default weights.
The models will run 20 epochs with a batchsize of 20 on the training and validation dataset.

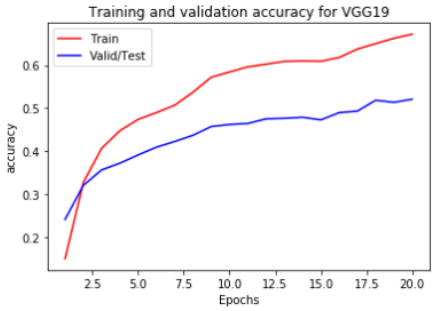
VGG-16
Test accuracy: 49.64%



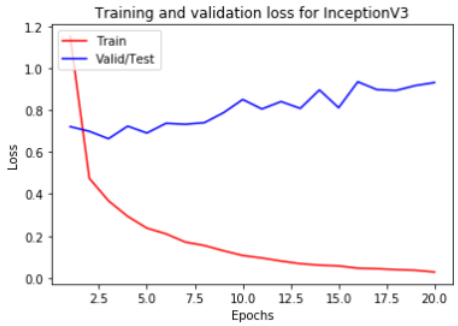


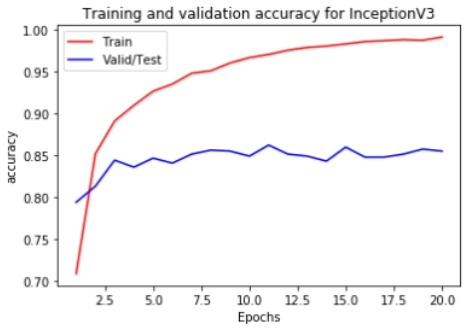
VGG-19
Test accuracy: 52.51%



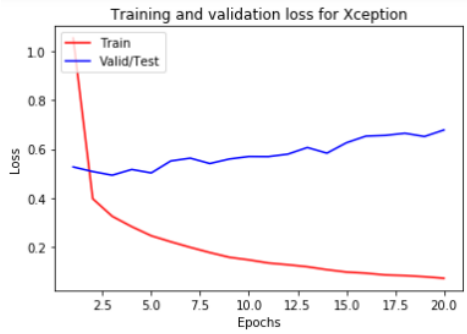


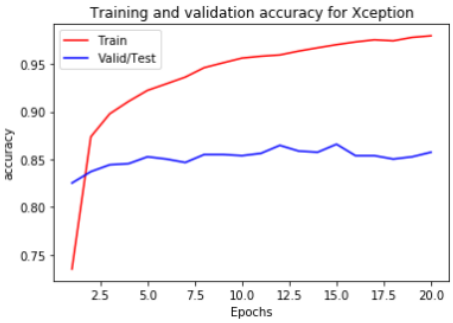
Inception V3
Test accuracy: 77.99%



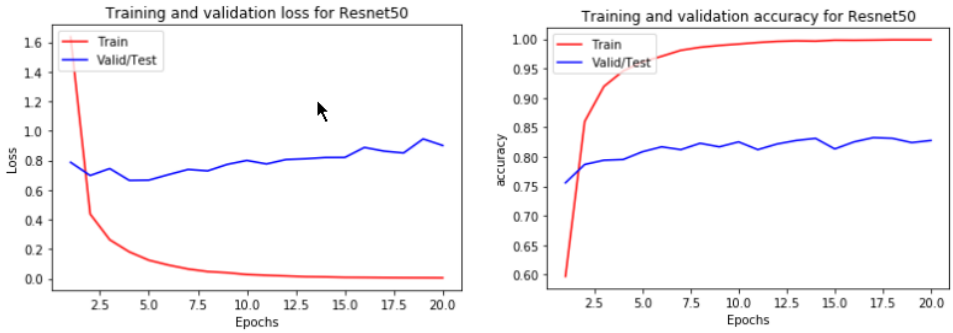


Xception
Test accuracy: 83.85%





ResNet 50
Test accuracy: 79.90%



To find the best model, let's first compare the loss function. We used the c

Let's start with VGG16 and 19 both models start with a loss of over 10 in test and validation dataset. After 20 epochs, this value can be reduced to approx. 6 for VGG16 and 5 for VGG19 Inception, Xception and ResNet show significantly better results with this value. All three models quickly achieve an error value of less than 1 per example.

To select a final model here, we look at the test accuracy of the three models:

Model	Test Accuracy
Inception	77,9%
Xception	83,8%
ResNet 50	79,9%

Test accuracy means if we test 1000 images with an accuracy of 79.9%, 799 images are correctly classified.

After testing the networks, I decided to use Resnet 50. While Xeption supplies a slightly better test accuracy the size of the bottleneck features is too unwieldy for me. ResNet as a pre-trained network delivers a similarly good performance with 79.90% test accuracy. Since nearly 80% is not enough, we try to improve the model by adjusting the hyperparameters.

IV. Results

Model Evaluation and Validation

After a few small adjustments

- Added a Dense layer with activation 'relu'. Due to the fact that relu not activate all neurons at the same time it can help us to detect patterns in our dataset.
- Drop out to minimize overfitting.
- I also decided to change the optimizer to sgd instead of adam or rmsprop, it seems to take advantage of its learning rate and momentum.

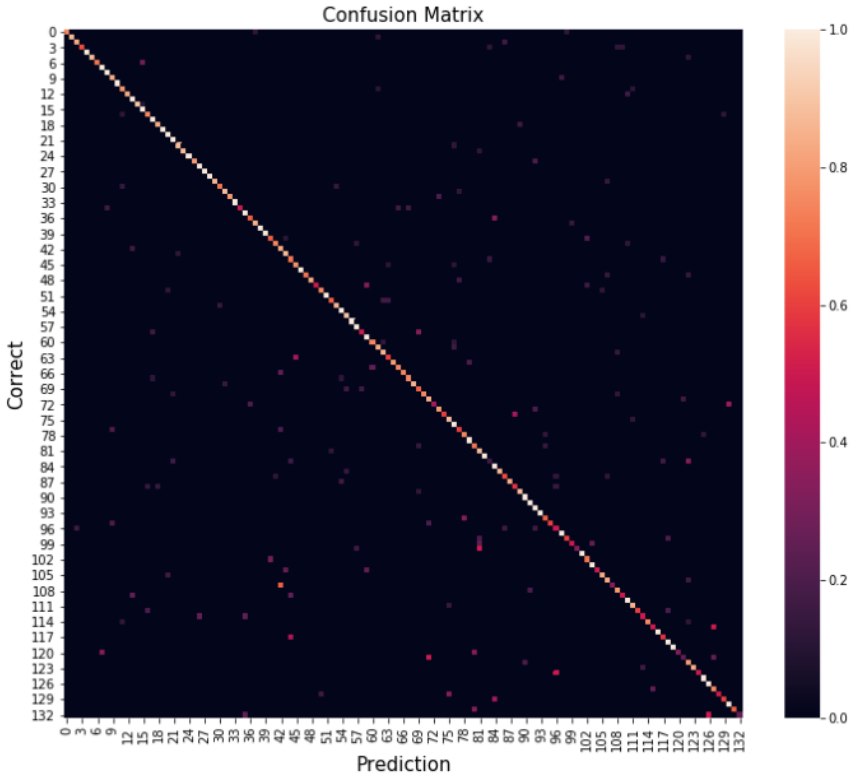
The following model resulted:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               34181
=================================================================
Total params: 558,725
Trainable params: 558,725
Non-trainable params: 0
```

```
Resnet50_model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

The model was trained in 200 epochs and resulted in a Test accuracy of 84.68%. Since we want to use this model for the web application, it was saved for later use.

If the result of the model is viewed as a confusion matrix, it looks like this:



A large part of the dog breeds is recognized perfectly.

Individual outliers are closely related breeds of dogs that are very similar, the respective picture depends on which breed is determined.

V. Conclusion

Reflection

Implementing the bottleneck features of ResNet we created a function that predicts the breed of an image

```python
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.

from extract_bottleneck_features import *

def Resnet50_predict_breed(img_path):
    # extract bottleneck features
    bottleneck_feature = extract_Resnet50(path_to_tensor(img_path))
    # obtain predicted vector
    predicted_vector = Resnet50_model.predict(bottleneck_feature)
    # return dog breed that is predicted by the model
    return dog_names[np.argmax(predicted_vector)]
```

We have come to the last step. All functions developed so far must be combined in order to create the final algorithm.

Let's go back to the problem:

The target was to develop an algorithm where the user can upload an image. By analyzing the picture the output should be the uploaded image as well as the following messages based on the above premises.

We have built a face detector that recognizes a face in all continuous images that contain a person.

When we convert the image into a 4D tensor we can, with help of ResNet50, say with a probability of 100% whether the image contains a dog.

We also tested neural networks and with the help of transfer learning built a network that recognizes the respective breed of dog in a picture with a probability of over 84%.

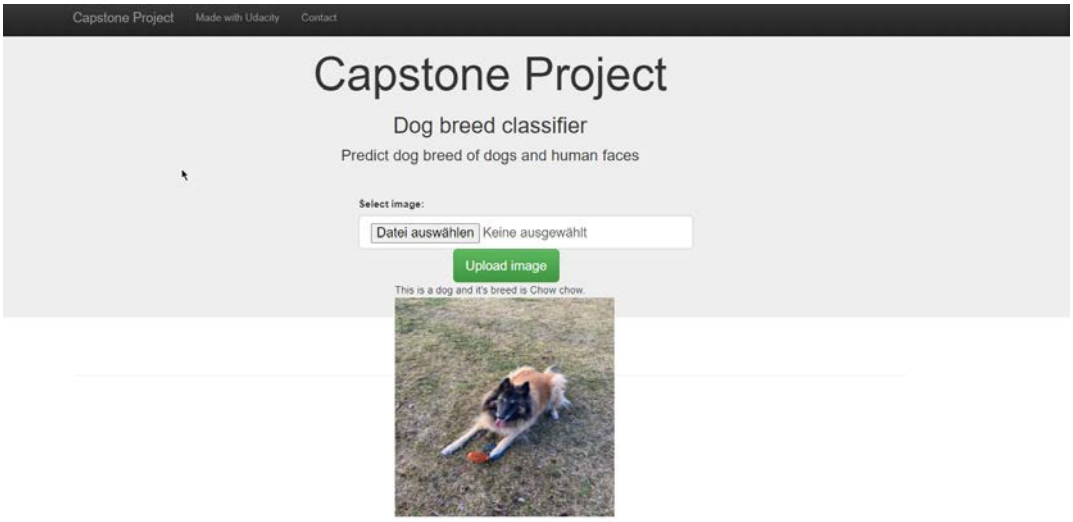
Let's combine all these functions and put them together in an IF clause, so that the weakness of the face detector does not come to fruition.

This is the final result:

```python
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def dog_breed_prediction(img_path):

# show image
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()

# human or dog detector
    human = face_detector(img_path)
    dog = dog_detector(img_path)

# if clause for dog breed detection
    if dog:
        breed = Resnet50_predict_breed(img_path)
        print ("This is a dog and it's breed is {}.".format(breed))
    elif human:
        breed = Resnet50_predict_breed(img_path)
        print("This is not a dog but it's look like a {}.".format(breed))
    else:
        print("This looks niether human or dog, must be something else.")
```

Web Application

Using Flask, I put all the data together and built a web application. Which makes it possible to upload an image and run it through our prediction algorithm.



Improvement

From my point of view it works very well, the algorithm only has its difficulties with similar breeds. These could be remedied with the following adjustments.

Architecture:

- Increase the model capacitiy by adding more layers
- change the batchsize or change the learing rate
- increase the number of epochs

Redesign the image:

- change the resolution
- rotation or flipping the image
- random image shifts