DSND Capstone Project Report – Dog Breed Classification

TABLE OF CONTENTS

Project Overview

This project is one of the most popular Udacity projects across machine learning and artificial intelligence nanodegree programs. It combines the more and more popular computer vision and deep learning technique, machine learning algorithm altogether. And it is fun to test the results in the final step.

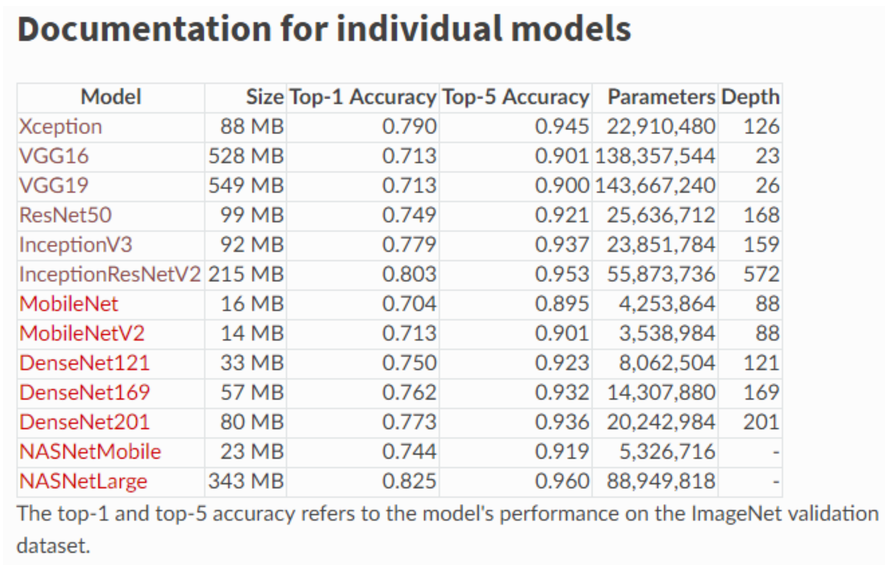
First let me briefly introduce some knowledge and techniques involved in the project.

ImageNet [1] is a large and popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. All the images of humans and dogs in the project are downloaded from ImageNet.

Haar feature-based cascade classifier [2], one of pre-trained face detectors in OpenCV, is used in this project for human face detection. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

It is known that that lower convolutional layers capture low-level image features, while higher convolutional layers capture more and more complex details, such as body parts, faces, and other compositional features. This critical information is contained in one of the final convolutional layers or early fully-connected layers in large convolutional neural networks trained on ImageNet. For a new task, people can simply use the features of a state-of-the-art CNN pre-trained on ImageNet and train a new model using the extracted features. Practically, we either keep the pre-trained parameters fixed or fine-tune them to ensure that we do not unlearn the previously acquired knowledge. This simple approach has been shown to achieve surprisingly good results on vision tasks. This is the essence of transfer learning [3].

Keras provides the following pre-trained CNN models for image classification, and these models are trained on ImageNet with weights, as shown in the table below [4].



Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

VGG16, VGG19, ResNet50, InceptionV3 and Xception ranked the top 5 in terms of accuracy, and thus are selected for transfer learning in this project. CNN models based on these top 5 models are constructed, trained, compared and optimized by the accuracy.

When compiling a CNN model in Keras, optimizers are used. Among all the optimizers available in Keras[5], RMSprop, Adam, Nadam are selected in the project, as they are proven to be effective and superb than other optimizers in image identification. They are all gradient-descent based adaptive learning rate method, since they can adapt the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. Adam can be viewed as a combination of RMSprop and momentum, and Nadam (Nesterov-accelerated Adaptive Moment Estimation) combines Adam and NAG (Nesterov accelerated gradient) [6]. It is also suggested that Adam might be the best overall choice when choosing the optimizers due to its superiority [6, 7].

Problem Statement

The goal is to classify images of dogs according to their breeds. At the end of this project, the code is supposed to accept any user-supplied images as input, detect objects such as dogs, human or neither, and then make predictions on their classes. If a dog is detected in the image, it will predict the dog's breed. If a human is detected, it will identify the dog breed that is the most resembling.

Procedures & Methodology

In order to achieve the project goal, seven steps are followed:

Step 0: Import Datasets: A dataset of dog images and a dataset of human images are both imported.

Step 1: Detect Humans: as mentioned in the project overview, OpenCV's Haar feature-based cascade classifiers are used to detect human faces in images.

Step 2: Detect Dogs:

- We use a pre-trained ResNet-50 model to detect dogs in images. We download the ResNet-50 model, along with weights that have been trained on ImageNet. Given an image, this pre-trained ResNet-50 model returns a prediction for the object that is contained in the image.
- Pre-process the data: When using TensorFlow as backend, Keras CNNs require a 4D tensor as input, with the shape of (nb_samples, rows, columns, channels). Additional image conversion and normalization is also required. Therefore data preprocessing is implemented before training the model for dog detection.

Step 3: Create a CNN to Classify Dog Breeds (from Scratch): a CNN model from scratch is created and trained, and the accuracy of classifying bog breed is calculated.

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning): Transfer learning is used. A pre-trained model called VGG16 is used in this step. The last convolutional output of VGG-16 is fed as input to our own model. We only add a global average pooling layer and a fully connected layer. The model is trained and accuracy is calculated.

Step 5: Create, Compare and Optimize a CNN to Classify Dog Breeds (Comparing 4 Transfer Learning Models):

- The other four pre-trained models (VGG-19, ResNet-50, InceptionV3, Xception) are used for transfer learning. They are all trained and compared using the prediction accuracy.
- The superior model out of the five (VGG-16, VGG-19, ResNet-50, InceptionV3, Xception) models are selected for further optimization by using different optimizers.

Step 6: Write My Algorithm to Classify Dog Breeds: the function should be able to determine if the images contains a human, dog or neither, and then

- 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.

Step 7: Test My Algorithm: Run the algorithm on pre-imported images to check the dog breed classifications.

Implementation & Optimization

In this session, the abovementioned seven steps are also followed. Only the essential code blocks for implementation will be illustrated here. For the complete implementation codes, please refer to the jupyter notebook on Github.

Step 0: Import Datasets

Step 1: Detect Humans:

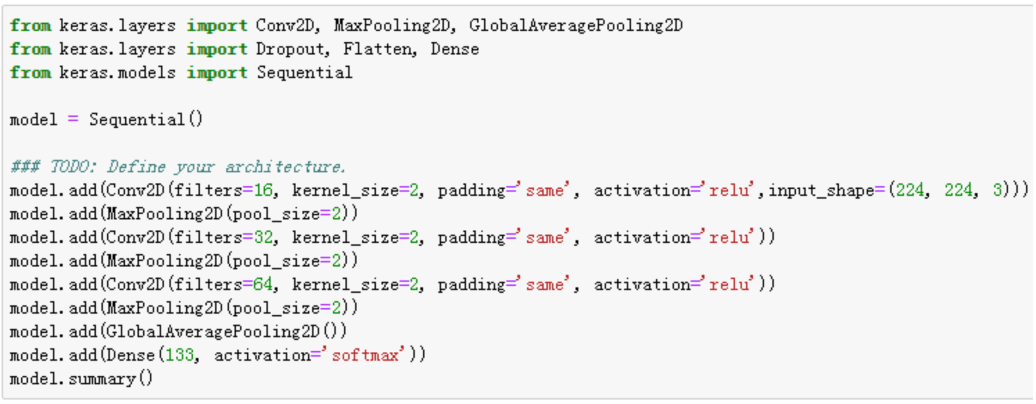
Step 2: Detect Dogs: First of all, we need to pre-process the data, and there are three steps before the images are ready for CNN models.

1. TensorFlow-backend Keras CNNs require a 4D tensor as input, with shape of (nb_samples,rows,columns,channels). The path_to_tensor function in the jupyter notebook takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224224×224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels, and the returned tensor have shape of (1, 224, 224, 3). The paths_to_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape of (nb_samples, 224, 224, 3).
2. The RGB image is then converted to BGR by reordering the channels. All pre-trained models have the additional normalization step that the mean pixel (expressed in RGB as [103.939,116.779,123.68][103.939,116.779,123.68] and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image. This is implemented in the imported function preprocess_input.
3. We rescale the images by dividing every pixel in every image by 255.

For full codes of data preprocessing, please check out the jupyter notebook.

Step 3: Create a CNN to Classify Dog Breeds (from Scratch): a CNN model from scratch is created and trained, and the accuracy of classifying bog breed is calculated.

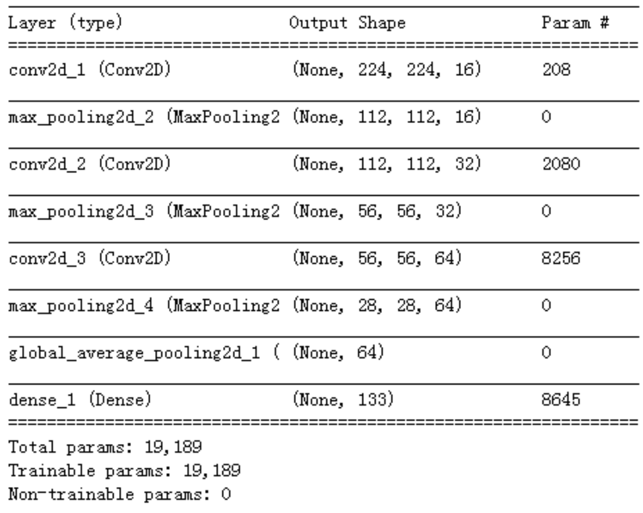
The codes and architecture for the CNN model are shown as below.



```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

model = Sequential()

### TODO: Define your architecture.
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(GlobalAveragePooling2D())
model.add(Dense(133, activation='softmax'))
model.summary()
```



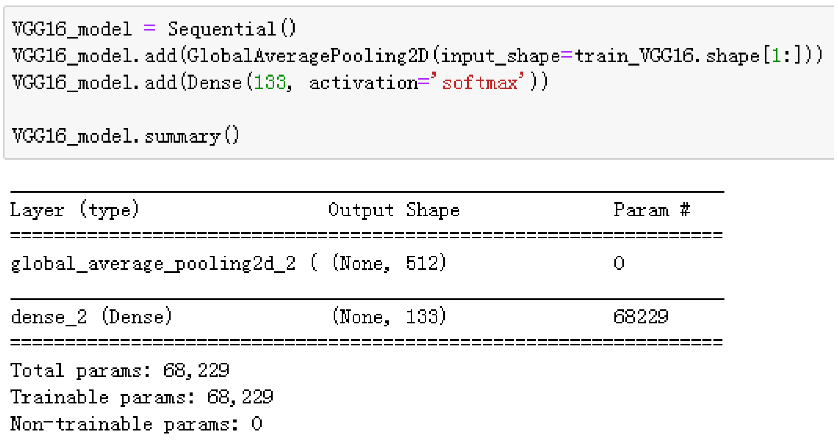
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               8645
=================================================================
Total params: 19,189
Trainable params: 19,189
Non-trainable params: 0
```

There are totally 8 layers.

- The 1st layer is a convolutional layer. The number of filters is the same with the number of small pieces of feature maps. Kernel size by default is 2*2, and stride by default is 1. To prevent the information from losing, I used fill with zero on 'padding'. The input shape is based on the dimension of input tensors.
- The 2nd layer is a maximum pooling layer, which is used to decrease the dimensionality.
- The 3rd layer is a convolutional layer.
- The 4th layer is a maximum pooling layer.
- The 5th layer is a covolutional layer.
- The 6th layer is a maximum pooling layer.
- The 7th layer is GAP (Global Average Pooling) layer. It takes the average of all the features on each feature map. This reduces the dimensions greatly by converting the feature maps into a feature vector.
- The 8th layer is a dense layer. The input nodes will be the total classes of all the dogs, which in our case is 133. By default a softmax activation function is used.

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning): Transfer learning with the pre-trained model VGG-16 is used. The output of VGG-16 (network and weights) is extracted and stored in bottleneck_features, which will be the input of our models. We only add a global average pooling (GAP) layer and a fully connected layer in our architecture, as shown below.

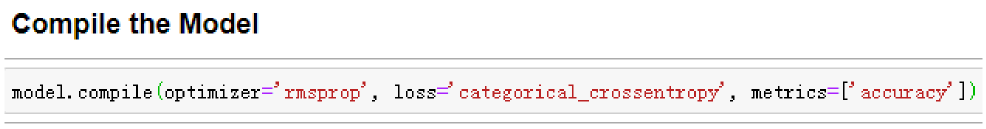
The GAP layer is used to reduce the dimensionality and parameters for computation. The dense layer has 133 nodes, which is the number of dog breeds we are going to classify.



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

VGG16_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
```



Compile the Model

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

When compiling the model, rmsprop is firstly used as the optimizer. The categorical_crossentropy applies to categorical multiclass classification, and thus applies well to our case. Accuracy is used as the evaluation metrics for CNN models.

Step 5: Create, Compare and Optimize a CNN to Classify Dog Breeds (Comparing 4 Transfer Learning Models):

- Four pre-trained models (VGG-19, ResNet-50, InceptionV3, Xception) are used for transfer learning. They are all trained and compared by the prediction accuracy.
- The best model out of the five (VGG-16, VGG-19, ResNet-50, InceptionV3, Xception) models are selected for further optimization by using different optimizers. As mentioned above, the categorical_crossentropy applies well to our case. Here we only change the optimizer to further optimize the best model. Adam and Nadam are both used as the alternative optimizers to further train the optimal model from step 4.

Step 6: Write My Algorithm to Classify Dog Breeds

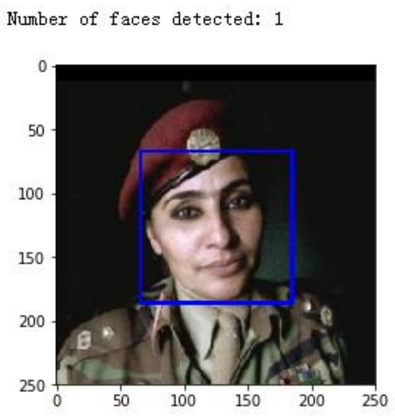
Step 7: Test My Algorithm: the function wrote in step 6 is run on pre-imported images.

Results & Analysis

In this session, the results and analysis will also be based on the seven steps.

Step 0: Import Datasets: A total 133 dog categories and 8351 dog images, together with a total 13233 human images are imported.

Step 1: Detect Humans: The human face detector based on OpenCV gives the following prediction results: In the first 100 human images, the percentage of human face detected is 100.00%. In the first 100 dog images, the percentage of human face detected is 11.00%. Below is the returned result from human face detector.



Step 2: Detect Dogs: The human face detector based on ResNet-50 gives the following prediction results: In the truncated human images, the percentage of dogs detected is 0.00%. In the truncated dog images, the percentage of dogs detected is 100.00%.

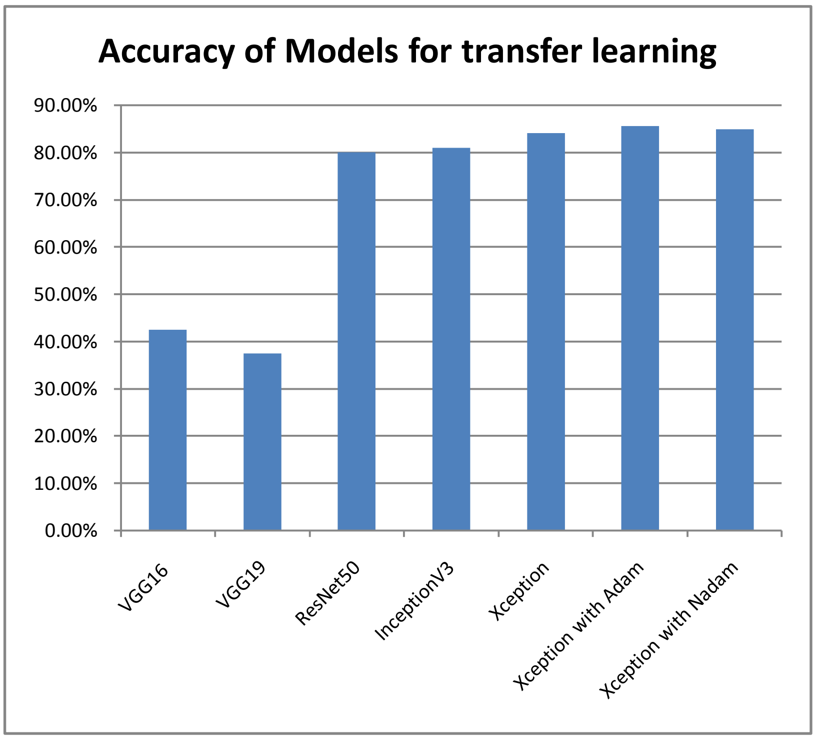
It seems that our dog detector gives better accuracy then the human detector, which means that ResNet-50 has a better accuracy than Haar feature-based cascade classifier. This classifier has limitations as it normally requires human images with a clear view of a face.

Step 3: Create a CNN to Classify Dog Breeds (from Scratch): The model gives from scratch an accuracy of 4.1866% within 10 epochs. This is acceptable, as a random guess only provides the correct answer around 1 in 133 times.

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning): By using VGG-16 as our pre-trained model, we obtained an accuracy of 42.4641%, which is a drastic increase from that of the scratch CNN model we built in step 3.

Step 5: Create, Compare and Optimize a CNN to Classify Dog Breeds (Comparing 4 Transfer Learning Models):

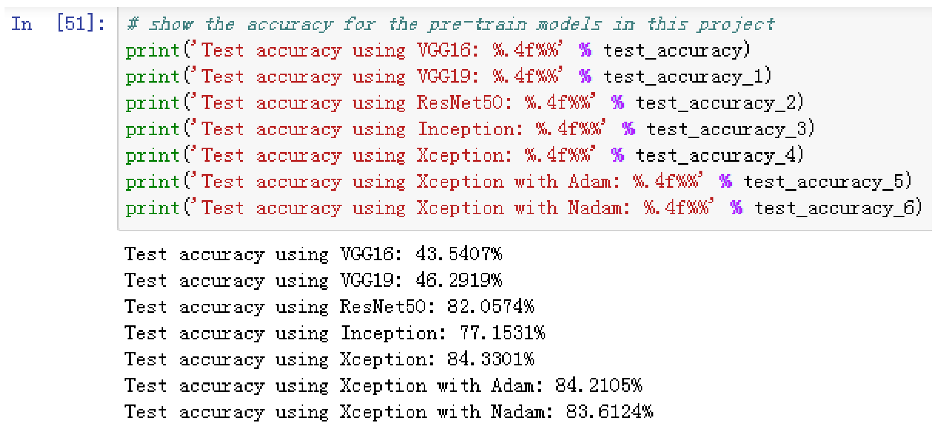
Models for transfer learning	**Accuracy**
VGG16	42.46%
VGG19	37.44%
ResNet50	80.14%
InceptionV3	80.98%
Xception	84.21%
Xception with Adam	85.65%
Xception with Nadam	84.92%



The table above shows the prediction accuracy of all the CNN models used with different optimizers. The accuracy of ResNet50, Inception, Xception are all above 80%. Xception with Adam outperforms the rest models with an accuracy of 85.65%. Apparently, VGG16 and VGG19 fall short for this kind of task.

Compared with a CNN model from scratch, transfer learning has an unbeatable strength in boosting the accuracy and computation speed.

It is worth to point out that it is hard to get reproducible results every time I run the code, although random seed was set [8, 9]. In most times, VGG19 has better performance than VGG16, and InceptionV3 is better than ResNet50. Sometimes, Xception with Nadam can have slightly better accuracy than or equals to that with Adam. Below was one of the results I got once. But overall, the results are consistent. ResNet50, InceptionV3 and Xception can give prediction accuracy over 80%, and Xception stands out.



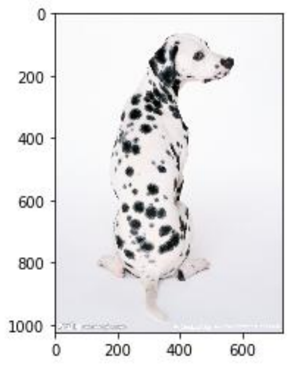
```
In [51]: # show the accuracy for the pre-train models in this project
         print('Test accuracy using VGG16: %.4f%%' % test_accuracy)
         print('Test accuracy using VGG19: %.4f%%' % test_accuracy_1)
         print('Test accuracy using ResNet50: %.4f%%' % test_accuracy_2)
         print('Test accuracy using Inception: %.4f%%' % test_accuracy_3)
         print('Test accuracy using Xception: %.4f%%' % test_accuracy_4)
         print('Test accuracy using Xception with Adam: %.4f%%' % test_accuracy_5)
         print('Test accuracy using Xception with Nadam: %.4f%%' % test_accuracy_6)
```

```
Test accuracy using VGG16: 43.5407%
Test accuracy using VGG19: 46.2919%
Test accuracy using ResNet50: 82.0574%
Test accuracy using Inception: 77.1531%
Test accuracy using Xception: 84.3301%
Test accuracy using Xception with Adam: 84.2105%
Test accuracy using Xception with Nadam: 83.6124%
```

Step 6: Write My Algorithm to Classify Dog Breeds

Step 7: Test My Algorithm: Please refer to the jupyter notebook for the test results. The model gives better prediction accuracy on human faces or neither, while it made mistakes in predicting dogs. However the overall accuracy is high and therefore acceptable. Below is one of the test results.

```
Hello, dogs!
You are a Dalmatian.
```



Conclusions & Discussions

1. Based on the prediction results, the model gives better prediction accuracy on human faces or neither, while it made mistakes in predicting dogs. However the overall accuracy is high and therefore acceptable. The results are better than my expectation.

2. Transfer learning has unbeatable advantages compared with creating a CNN model from scratch, since it provides a much better accuracy and drastically reduces the training time. As mentioned in the project overview, the critical information of images is contained in one of the final convolutional layers or early fully-connected layers in large convolutional neural networks trained on ImageNet. Normally for a large number of images, the network can be very deep and consumes weeks to train. Meanwhile, these pre-trained models were already trained on a large corpus of photographs and are proven to make predictions on a relatively large number of classes, as long as the bottleneck features from photographs are effectively extracted [11]. Transfer learning has proven to be effective in making predictions of images[12]. Therefore for a new task, we can simply use the features of a state-of-the-art CNN pre-trained on ImageNet and train a new model using the extracted features.

3. Among the five pre-train models (VGG16, VGG-19, ResNet-50, InceptionV3 and Xception), ResNet50, InceptionV3 and Xception can give a prediction accuracy over 80%, and Xception stands out. VGG16, VGG-19 fall short in this kind of task. The result is a little different from the ranks given by Keras shown above in the project overview session. Taking the size into consideration, Xception takes the crown again.

4. RMSprop, Adam, Nadam are all effective and superb optimizers in this task. Most times, Adam performs slightly better than Nadam and RMSprop.

5. There are several proposals to further improve my model:

- In transfer learning, other than the pre-trained models compared in this project, other models can also be used, such as InceptionResNetV2, MobileNet, DenseNet listed on Keras documentation [4].
- When constructing the last few layers for transfer learning, other layers can also be used, such as Dropout layers. An aggressive Dropout layer has been proven to prevent overfitting and increase accuracy.
- Data Agumentation can also be considered to further improve the model. By using data agumentation, more relevant features will be trained and strengthened, while the less relevant patterns will be weakened. This can enhance the overall performance, and overfitting will also be prevented. The robustness of the model would be boosted as well. This has been proved by experiments [10].
- Fine-tuning on the model parameters and hyperparameters, such as different learning rates, vanilla SGD without momentum, or Adam with warm restarts as mentioned here [5]. In each restart, the learning rate is initialized to some value and is scheduled to decrease. Importantly, the restart is warm as the optimization does not start from scratch but from the parameters to which the model converged during the last step.

References

[1] http://www.image-net.org/
[2] https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
[3] http://ruder.io/transfer-learning/
[4] https://keras.io/applications/#available-models
[5] https://keras.io/optimizers/
[6] http://ruder.io/optimizing-gradient-descent/
[7] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[8] https://machinelearningmastery.com/reproducible-results-neural-networks-keras/
[9] https://stackoverflow.com/questions/32419510/how-to-get-reproducible-results-in-keras
[10] https://towardsdatascience.com/data-augmentation-experimentation-3e274504f04b
[11] https://machinelearningmastery.com/transfer-learning-for-deep-learning/
[12] ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf