Machine Learning Engineer Nanodegree

Capstone Project

Xin Meng

February 8th, 2018

I. Definition

Project Overview

Computer vision[1] is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Understanding in this context means the transformation of visual images into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

There are many successful applications of computer vision on image classification. For instance, to identify cancers[2] from healthcare images to reduce diagnose failure. Another example is to identify the person with an instant camera images on the customs

[1] Computer Vision <https://en.wikipedia.org/wiki/Computer_vision>

2 Identifying skin cancer with computer vision

<https://www.ibm.com/blogs/research/2016/11/identifying-skin-cancer-computer-vision/>(2016)

checkpoint and estimate immediately whether the man is the same person on the passport.

In this project, I will build a classification model to classify images. The system is supposed to learn associations between words used to describe an image and the visual features found in it, and gets better at predicting descriptions on its own. I have done the project with 2 Image datasets. One is CIFAR-10[3] and other is Caltech-101[4]. In the final project I selected Caltech-101 to build the model. Here's the reason why I abandon CIFAR10.

- The image size of CIFAR10 is (32, 32, 3) which is too small to be accepted by all the advanced Keras application models such as ResNet50, VGG16, etc. So we cannot use transfer learning on CIFAR10. That would greatly reduce the performance of the model.

- In addition, the very small image size could cause problem when using SIFT to extract keypoint and computer descriptors. There are many keypoint with None descriptor.

- Finally, the image provided by users is much larger that the images we trained on. So the model may not do well on user images.

After the trials with CIFAR10, I switched to a much larger image size dataset：Caltech101. The size of each image in Caltech101 is roughly 300 x 200 pixels.

The project is all about building a model to classify images. And as for an app, we need to design a user interface. I inform user to upload images from the 102 classes in Caltech101. The project can only predict unlabeled images of the predefined 102 classes.

Problem Statement

The problem is to train a multi-class classifier with Caltech101 dataset to classify images from 102 classes. In addition, the model is supposed to predict an unlabeled images which upload by users and the images should be in one of the 102 classes the model is trained to identify.

The task of the project is to develop a social image description platform on desktop, the task involved are the following:

3 CIFAR10 <http://www.cs.utoronto.ca/~kriz/cifar.html>

4 Caltech101 <http://www.vision.caltech.edu/Image_Datasets/Caltech101/>

1. Download and processes the Caltech101 dataset

2. Train a classifier from Caltech101 dataset by different approaches.

3. Evaluate the performance of the model by benchmark and find out the best approach.

4. Design a user interface for uploading images and predict the description word.

The final application is expected to be useful for users to upload images belong to the 102 classes and the app will predict the label to describe the image.

The candidate approaches for training a classification model are:

- CNN from Scratch
- CNN from Transfer Learning
- Bag of Words model combining Supervised Learning and Unsupervised Learning

I have tried these 3 approaches and select the best approach, and evaluated the performance on the test set by the measure metric. The result is reproducible.

Metrics

In the Data Exploration section of the project, we see there are 9145 images from102 classes in Caltech101 dataset. The average percentage of each class is 1%. Most categories have under 1% images while a few categories have more than 4% images, very few even exceeds 8%. Since the classes are imbalanced, we use accuracy, f1 score as the model metric. I also include the predict time cost for it is related to user experience.

The metrics to evaluate.

1. Evaluate the accuracy on test set, which can be compare to benchmark.

   Accuracy is a common metric for classifiers. This is a multi-classes classifier the accuracy=true classifications/datasize

2. Evaluate the f1score on test set, which can be compare to benchmark.

   Since I did not find the accuracy and f1score benchmark on the Caltech101. I used the accuracy and f1score of CNN model from scratch as the benchmark.

3. Evaluation time cost for prediction on test set.

We prefer a good accuracy, f1score and an acceptable time cost for practical use.

I abandon the metric of time cost for training model which is in the proposal. Because practically we only need to train the model once. The train time cost is not counted on the user experience. So I keep the time cost for prediction as a metric since it is relative to user experience.

II. Analysis

Data Exploration

Caltech101 has pictures of objects belonging to 102 categories. About 40 to 800 images per category. Most categories have about 50 images. The maximum category has 800 images and the minimum category has 31 images. The median is 59. The numbers of images in each category are variable. The maximum category is 'airplanes', the minimum category is 'inline_skate'. The size of each image is roughly 300 x 200 pixels. The image data contain 'target_names', 'data', 'target', and 'filenames'. We sample 10 image data to demonstrate here:

```
('The first 10 images have targets', array([ 6,  6, 12,  4,  6,  0,
 4, 18, 83,  2]))
('The first 10 images have filenames', array(['101_ObjectCategorie
s/airplanes/image_0669.jpg',
      '101_ObjectCategories/airplanes/image_0531.jpg',
      '101_ObjectCategories/binocular/image_0014.jpg',
      '101_ObjectCategories/Motorbikes/image_0204.jpg',
      '101_ObjectCategories/airplanes/image_0111.jpg',
      '101_ObjectCategories/BACKGROUND_Google/image_0254.jpg',
      '101_ObjectCategories/Motorbikes/image_0367.jpg',
      '101_ObjectCategories/camera/image_0001.jpg',
      '101_ObjectCategories/sea_horse/image_0003.jpg',
      '101_ObjectCategories/Faces_easy/image_0318.jpg'],
     dtype='|S53'))
('The first 10 images have target_names', ['BACKGROUND_Google', 'Fa
ces', 'Faces_easy', 'Leopards', 'Motorbikes', 'accordion', 'airpla
nes', 'anchor', 'ant', 'barrel'])
```

The full categories are:

```
The full categories are', ['BACKGROUND_Google', 'Faces', 'Faces_ea
sy', 'Leopards', 'Motorbikes', 'accordion', 'airplanes', 'anchor',
 'ant', 'barrel', 'bass', 'beaver', 'binocular', 'bonsai', 'brain',
 'brontosaurus', 'buddha', 'butterfly', 'camera', 'cannon', 'car_s
ide', 'ceiling_fan', 'cellphone', 'chair', 'chandelier', 'cougar_b
```

```
ody', 'cougar_face', 'crab', 'crayfish', 'crocodile', 'crocodile_h
ead', 'cup', 'dalmatian', 'dollar_bill', 'dolphin', 'dragonfly', '
electric_guitar', 'elephant', 'emu', 'euphonium', 'ewer', 'ferry',
 'flamingo', 'flamingo_head', 'garfield', 'gerenuk', 'gramophone',
 'grand_piano', 'hawksbill', 'headphone', 'hedgehog', 'helicopter
', 'ibis', 'inline_skate', 'joshua_tree', 'kangaroo', 'ketch', 'la
mp', 'laptop', 'llama', 'lobster', 'lotus', 'mandolin', 'mayfly', '
menorah', 'metronome', 'minaret', 'nautilus', 'octopus', 'okapi',
'pagoda', 'panda', 'pigeon', 'pizza', 'platypus', 'pyramid', 'revo
lver', 'rhino', 'rooster', 'saxophone', 'schooner', 'scissors', 's
corpion', 'sea_horse', 'snoopy', 'soccer_ball', 'stapler', 'starfi
sh', 'stegosaurus', 'stop_sign', 'strawberry', 'sunflower', 'tick
', 'trilobite', 'umbrella', 'watch', 'water_lilly', 'wheelchair',
'wild_cat', 'windsor_chair', 'wrench', 'yin_yang'
```

The images which targets are ‘accordion’ looks like:

Each image contains 3 channels for Red, Green, Blue.

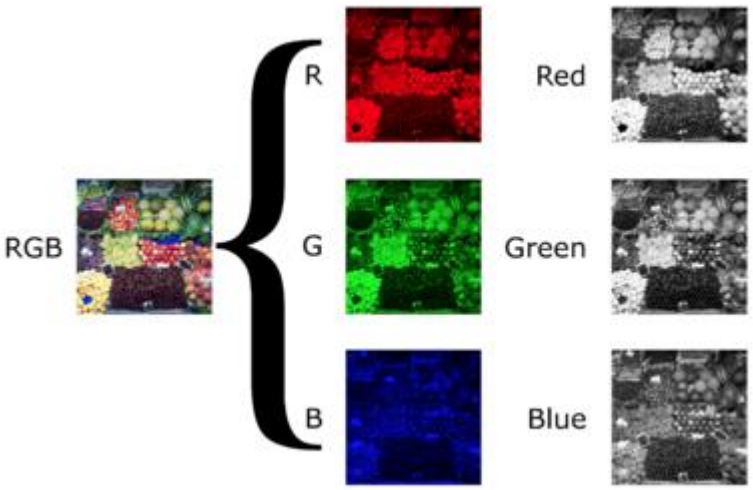


Fig.1 RGB channels of a image

We got the statistic of image count in targets (labels):

```
count    102.000000
mean      89.656863
std      123.678960
min       31.000000
25%       45.000000
50%       59.000000
75%       80.750000
max      800.000000
Name: images_count, dtype: float64
```

Fig.2 Statistic of Images Count in Targets

The maximum category has 800 images and the minimum category has 31 images. The median is 59. The numbers of images in each category are variable.

Exploratory Visualization

We see the bar chart of images count in each target (label) as Fig.2. Most categories have about 50 images while a few categories have more than 400 images, some categories have even 800 images.

<matplotlib.axes._subplots.AxesSubplot at 0x7f010832a910>

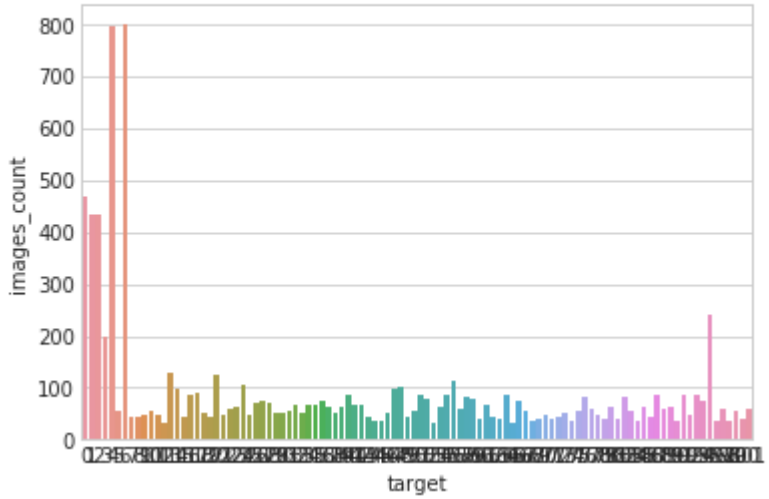


Fig.3 Images Count of Target

Algorithms and Techniques

I used CNN (Convolutional Neural Network[5]) for the main algorithms and I also have tried Bag of Words[6] model which combining unsupervised algorithm and supervised algorithm to classify images.

In machine learning, a **convolutional neural network[7] (CNN, or ConvNet)** is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. A CNN consists of an input and an output layer, between the input and output layer, there are convolutional layers, pooling layers, fully connected layers and normalization layers. A demonstration of ConvNet is here:

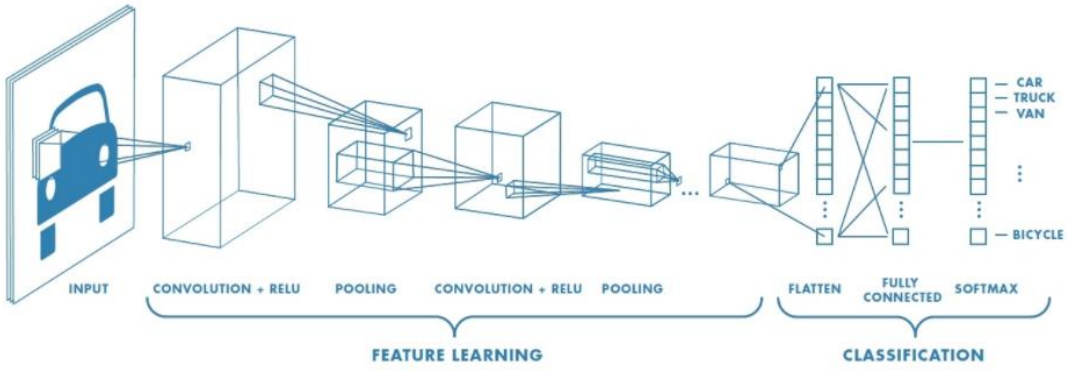


Fig.4 Demonstration of ConvNet

[5]Convolutional neural network <https://en.wikipedia.org/wiki/Convolutional_neural_network>
[6]Bag-of-words_model < https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision >
[7]Convolutional neural network < http://cs231n.github.io/convolutional-networks/>

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer**. We will stack these layers to form a full ConvNet architecture.

We illustrate the idea of each layers by a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

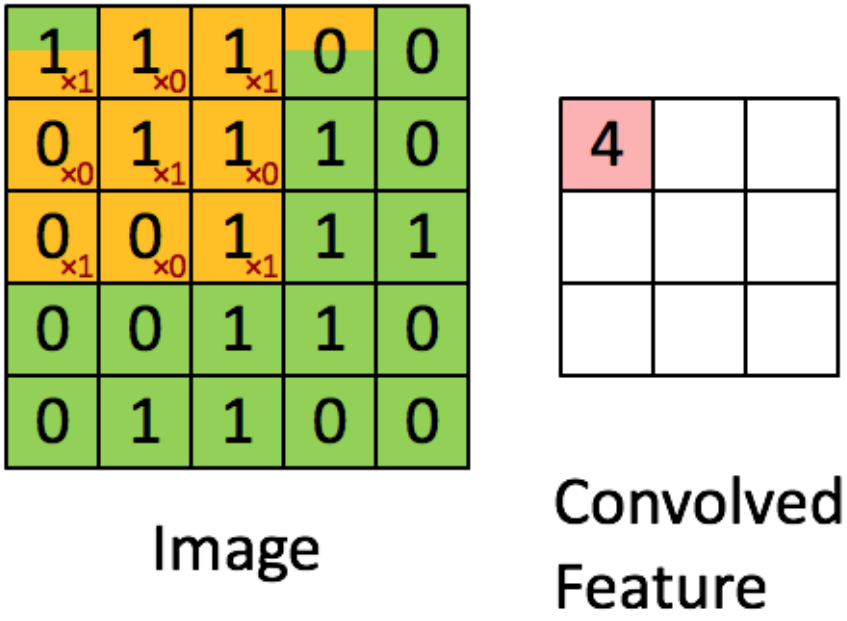


Fig.5 Demonstration of Conv layer

- RELU layer will apply an elementwise activation function, such as the max(0,x)max(0,x) thresholding at zero. This leaves the size of the volume unchanged.
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

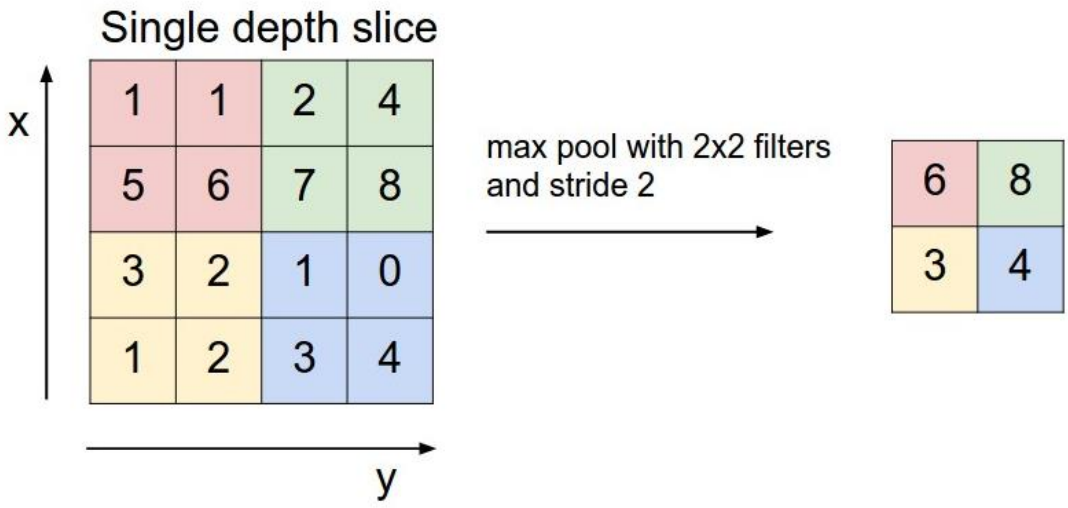


Fig.6 Demonstration of Pooling layer

- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. Each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

CNN classifier have these following parameters can be turned to optimize:

Training parameters:

- Number of epochs
- Batch size(how many images to look at once during a single training step)
- Optimizer (Optimization[8] is the process of finding the set of parameters W that minimize the loss function.)

Neural network architecture:

- Number of layers

[8]Optimizaion <http://cs231n.github.io/optimization-1/>

- Layer types(convolutional, fully connected, or pooling layer)
- Layer parameters

  Conv2D layer: filters, kernel_size, stride, activation

  Pooling layer: pool_size

  Fully connected layer: units, activation

I also tried to classification with **Bad of Words model.**

The **bag-of-words model**[9] is a simplifying representation used in natural language processing and information retrieval (IR). Also known as vector space model. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model has also been used for computer vision.

In **computer vision**, the **bag-of-words model** (BoW model) [10]can be applied to image classification, by treating image features as words. In document classification, a bag of words is a sparse vector of occurrence counts of words; that is, a sparse histogram over the vocabulary. In computer vision, a *bag of visual words* is a vector of occurrence counts of a vocabulary of local image features.

Benchmark

Since the description of Caltech101 does not provide any benchmark. I used the performance of CNN model from scratch as the benchmark. It includes:

- The accuracy and f1score on test set.
- The time cost of prediction on test set.

In the CNN from scratch model, I used a 4 layers of Conv layers and each layer followed by a pooling layer and a dense layer is add to the top of the mode. The input image tensor shape is (224, 224, 3) and the dense layer unit is 102.

The CNN from scratch model could be taken as benchmark because it is relatively simple of 4 conv layers and it is only trained on Caltech101 dataset. The more advanced and more complicated model such as CNN from transfer learning is supposed to have much high performance, because it is trained on a very large ImageNet dataset and have more complicated architecture of layers.

[9] Bag of Words model <https://en.wikipedia.org/wiki/Bag-of-words_model>
[10] Bag of Words model in computer vision <https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision>

III. Methodology

Data Preprocessing

In the data exploration section, we see the variety of images numbers from each category, the maximum is 800 while the minimum is 31. To use accuracy as a metric, we need the distribution of labels is balanced. One option comes into my mind is to select 30 images from each label. So we only have 3060 images for training and test, only one third of the total number 9145.

Actually I have tried training with the subset of images and found that would badly impact on the performance of the model. So I switched back to use the whole dataset to build the model.

The preprocessing done in the 'Pre-process Dataset' in the notebook consists of the following task:

1. Transform image data to tensor array with shape (9145,224,224,3)

2. Transform target label to one hot encoding array.

In the Train-Test-Split, I used train_test_split() to split the dataset for 2 times. Because we need train set, validation set and test set, so we split the whole dataset for 2 times. One is to get test set (915 images), another is to split train set(7407 images) and validation set(823 images). We used these train set, validation set and test set for CNN algorithm model.

Be noticed that the image data are hex. I have tried to divide the data by 255, but got a worse performance of accuracy on validation set is about 30% when using transfer learning.

```
train_Resnet50=extract_Resnet50(train_tensors.astype('float32')/2
55
)
valid_Resnet50=extract_Resnet50(valid_tensors.astype('float32')/2
55
)
test_Resnet50=extract_Resnet50(test_tensors.astype('float32')/255
)
```

```
The length of an image data 6709
 ('The image data looks like', '\xff\xd8\xff\xe0\x00\x10JFIF\x00\x
01\x01\x00\x00\x01\x00\x01\x00\x00\xff\xdb\x00C\x00\x08\x06\x06\x
07\x06\x05\x08\x07\x07\x07\t\t\x08\n\x0c\x14\r\x0c\x0b\x0b\x0c\x1
```

```
9\x12\x13\x0f\x14\x1d\x1a\x1f\x1e\x1d\x1a\x1c\x1c $.\' ",#\x1c\x1c
(7),01444\x1f\'9=82<.342\xff\xdb\x00C\x01\t\t\t\x0c\x0b\x0c')
The total number of images is 9145
('The shape of tensor stack is', (9145, 224, 224, 3))
```

While in the trail of Bag of Words model. We need a separate split. This time we only need train set and test set. But we still need two times split. Because the collection set of keypoint of images is huge. The memory error occurs when cluster with huge keypoint descriptors. To avoid memory error, I suggest the total keypoint descriptors are controlled up to 50000. So we need the first split to get a smaller dataset and then split the smaller set to get the train set(365 images) and test set(92 images).

Implementation

The model build implementation process can be split into 3 approaches:

1. The trail of CNN from scratch
2. The trail of CNN transfer learning
3. The trail of Bag of Words model

In the **trail of CNN from scratch**, training and testing process were done in the Jupyter notebook *'Step2 Create a CNN to Classify Images (from Scratch)'*, and can be further divided into the following steps:

1) Define the CNN architecture and training parameters (epoch and batchsize)

   I used a 4 layers of CNN and each layer followed by a pooling layer and a dense layer is add to the top of the mode. The input image tensor shape is (224, 224, 3) and the dense layer unit is 102.

2) Define the loss function (categorical_crossentropy), optimizer (rmsprop), metric (accuracy)
3) Train the network, logging the validation/training loss and the validation accuracy
4) If the accuracy and f1score is not high, return to step 1, try tuning parameters of CNN
5) Save weights of best model
6) Testing with test set

We got a test accuracy of 40% , f1score of 0.18 and we have not use data augmentation yet. The predict time on test set is 1.5 seconds

In the **trail of CNN transfer learning**, training and testing process were done in the Jupyter notebook *'Step3: Trials for Transfer Learning to Classify Images'*, and can be further divided into the following steps:

During the way1: Train ResNet50 Starting from None Weight

1) Load ResNet50 model from keras.applications starting from weights is None and including the fully-connected layer at the top of the network and setting classes=102
2) Define the loss function (categorical_crossentropy), optimizer (rmsprop), metric (accuracy)
3) Train the network, logging the validation/training loss and the validation accuracy

The speed of model fit is very slow, so I only trained 2 epochs and just have some senses of the result. We see after the 2 epochs training, the validation accuracy is about 10%, which is quite lower than CNN from scratch getting 20% validation accuracy on the first epoch. So training the ResNet50 from None weight is a bad idea. We abandon the approach and do not proceed the evaluation on test set.

During the way2: Transfer Learning with Imagenet Weights

1) Load ResNet50 model from keras.applicaions with weights trained by ImageNet (weight=imagenet) and without the fully-connected layer at the top of the network.
2) Extract ResNet50 model (with weight=ImageNet) output of train set, valid set and test set. And we train and test with the 3 extract outputs in the transfer learning CNN model.
3) Add a global pooling layer and a dense layer with 102 and activation = softmax.
4) Define the loss function (categorical_crossentropy), optimizer (rmsprop), metric (accuracy)
4) Train the network, logging the validation/training loss and the validation accuracy

We see the test accuracy is 92%, f1score is 1 and the predict time on test set is 1 seconds. It beats the accuracy performance of CNN model from scratch which is accuracy =40% , f1score=0.18 and have a close predict time on test set. Be noticed that we used the same trainset, validation set and test set in the CNN from scratch model and CNN transfer learning model. So the prediction time cost is comparable.

In the **trail of Bag of Words model**, training and testing process were done in the Jupyter notebook *'Step4: Trials for Bag of Words to Classify Images'*, and can be further divided into the following steps:

1) Generate SIFT features from images by 'gen_sift_features()' function. The function takes image file list as input and return a list of SIFT keypoint descriptors with the same indices as image file list input. We generate trainset(365images) and test set(92images) images keypoint descriptors. Be noticed that every keypoint descriptor is a vector of 128 dimensions. Images may have different number of keypoint.
2) Use 'descriptorcorrect()' function to detect and remove any descriptors which is null.
3) Collect and stack all the keypoint descriptors into 1 list. We got :

```
The total keypoint descriptors in X_train_desc_correction are 20
0816
The total keypoint descriptors in X_test_desc_correction are 413
38
```

4) Cluster the keypoint descriptors, try different K and select K with the higher silhouette_score. The higher silhouette_score means the larger inter cluster distance and smaller intra cluster distance. We prefer the higher silhouette_score clusters.

   Here notice that, we have 200816 keypoint of 365 images in train set and 41338 keypoint of 92 images in test set. I have memory error occur when clustering with the trainset. So just to demonstrate the concept, here I use test set for clustering because its keypoint space is much smaller than trainset.

   I should mention that the clusters number is not the labels. We use cluster numbers to computer histogram of each image and use the histogram to predict labels. So the cluster number can be less than the labels.

5) Predict the clusters of keypoint of each image (on the smaller testset).

6) Computer the cluster histogram of each image (on the smaller testset).

7) Use supervised learning to predict the label with the image histogram of clusters. (on the smaller test set)

We evaluate the mean accuracy score on the data we train the model, it is only be 18%. So the performance of unknown data is supposed be lower. The performance of BoW is much lower than our benchmark of CNN from scratch. So BoW is not a good idea on Image classification.

Refinement

Since transfer learning model takes advantage of weights trained on “ImageNet” , it is no need to tune parameters on it . We tune parameters on the model of CNN from scratch, because we have more parameters to tune on.

We use `keras.wrappers.scikit_learn` to wrap the CNN from scratch model and tune parameters with `Gridsearchcv` in `scikit-learn`. We set `cv=5`, so we have more image data for training. We tune the `filters` and `optimizer`. Be noticed that we can't set `callbacks=[checkpointer]`, so we don't use validation set. We use the `accuracy` which is the metric of estimator to evaluate. And we show `best_parameters` and `best_score` of `gridsearchcv`.

```
('The best parameters are:', {'epochs': 5, 'first_filters': 16, 'op
timizer': 'rmsprop', 'poolsize': 2, 'batch_size': 20})
('The best accuracy is:', 0.25489402489654445)
```

We see the best parameters(`first_filters`,`optimizer`) is the same with the previous CNN from scratch model. So we keep the previous CNN model parameters.

IV. Results

Model Evaluation and Validation

The best model is CNN with transfer learning. We got the test accuracy is 92% and f1score=1, and the predict time on test set is 1 seconds. It beats the accuracy performance of CNN model from scratch which is accuracy=40%, f1score=0.18 and have a close predict time cost on test set. Be noticed that we used the same trainset, validation set and test set in the CNN from scratch model and CNN transfer learning model. So the prediction time cost is comparable.

For a complete description of the final model and the training process is following:

1) Load ResNet50 model from keras.applicaions with weights trained by ImageNet (weight=imagenet) and without the fully-connected layer at the top of the network.

2) Extract ResNet50 model (with weight=ImageNet) output of train set, valid set and test set. And we train and test with the 3 extract outputs in the transfer learning CNN model.

3) Add a global pooling layer and a dense layer with 102 and activation = softmax.

4) Define the loss function (categorical_crossentropy), optimizer (rmsprop), metric (accuracy)

5) The epoch is 10 and batchsize=20

To verify the robustness of the final mode, 11 test imageswas conducted using an image from internet. Fig.7 shows the testing classification sample output.

Fig.7 test classification sample output

We see that the model correctly classified the typical object image such as: ‘dolphin’, ‘bonsai’, ‘old camera’. It classified the ‘phone’ to ‘laptop’ and ‘ant’ to ‘scorpion’, but I think it is closed to right answer and reasonable. It wrongly classified : ‘text’, ‘airplane’, ‘cartoon_ant’, ‘bass’, ‘new camera’, ‘human’. They were all classified as ‘Background_google’. I looked in to the images of Background_google, it is a miscellaneous folder. So the model predicted all the image unknown to Background_google. The model does badly in the images has more than one object such as 'bass.jpg'. And also does badly in the new style of object such as new camera.

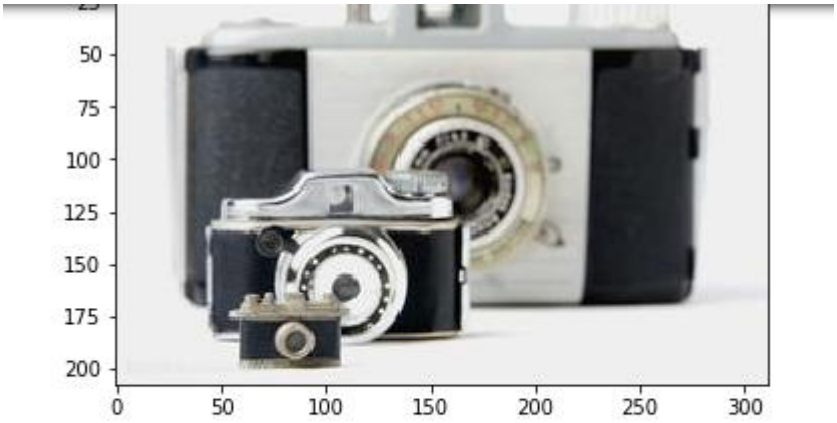
Justification

We got the test accuracy is 92%, f1score is 1, and the predict time on test set is 1 seconds. It beats the accuracy performance of CNN model from scratch which accuracy is 40%, f1score is 0.18 and have a close predict time on test set. Be noticed that we used the same trainset, validation set and test set in the CNN from scratch model and CNN transfer learning model. So the prediction time cost is comparable.

V. Conclusion

Free-Form Visualization

The model does well in a typical style of object , such as a traditional camera in Fig.8. But does badly in the new style of object such as new camera in Fig.9. So for the sake of improved classification we need to add more new style object to the trainset.




```
('The label of the upload image is:', 'camera')
('The prediction label is: ', 'camera')
('The max probability is: ', 0.99592066)
The predict time on image is 0.091316 seconds
```

Fig.8 Traditional Camera


```
('The label of the upload image is:', 'new camera')
('The prediction label is: ', 'BACKGROUND_Google')
('The max probability is: ', 0.96034819)
The predict time on image is 0.101740 seconds
```

Fig.9 New style Camera

The model also does badly on the images contain multi object, such as Fig.10 ,it classifies it into Background_google which is a miscellaneous folder.

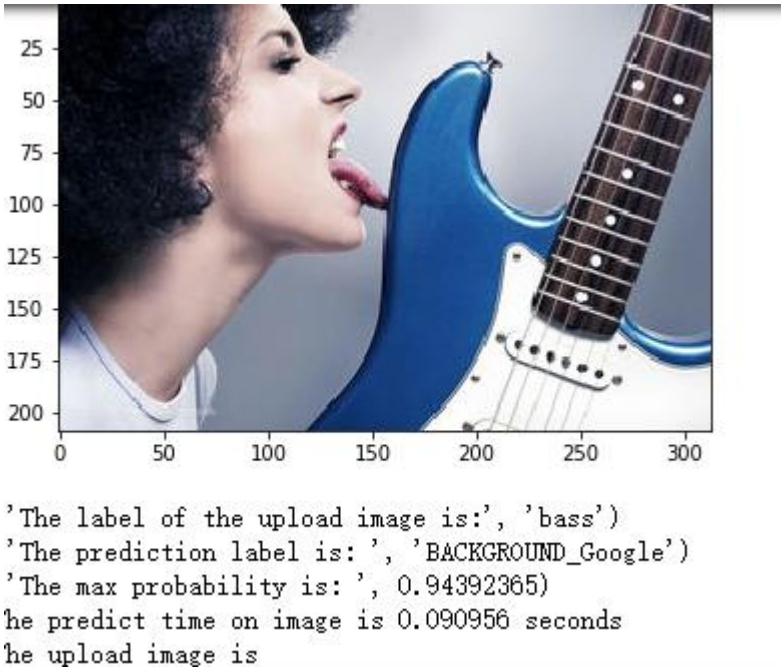


Fig.10 Bass and human

Reflection

The process used for project can be summarized using the following steps:

1) An initial problem and relevant, public datasets were found.
2) The data was downloaded and pre-processed (to tensors).
3) A benchmark was created for the classifier.
4) The classifier was trained using the data (3 approaches trail until the best and satisfactory model is found)
5) Tested with an unknown image and predicted correctly.
6) A user interface design is demonstrated.

I have proceeded the project with 2 datasets. My first trial is to build a model on CIFAR10 dataset. Because the images size of CIFAR10 are too small (32, 32, 3). The images cannot be used to on CNN transfer learning. That would greatly limited the highest performance

the model can approach. I only got the best accuracy from CNN scratch model of 44%. Although it beats the benchmark of 11% from the description of CIFAR10. It is still very low to be accepted on an app. So I decided to switch to Caltech101 dataset which have images of roughly 300 x 200 pixels and I got the best accuracy of 92% and predict time of several milliseconds which is not only beyond the CNN scratch model but also quite acceptable by an app.

Another difficulty I want to mention is the trial of Bag of Words model. The process of BoW model is complicated which including using SIFT to generate keypoint descriptors, unsupervised learning to cluster 'words', and supervised learning to predict labels with the histogram of words in the images. One reason I switched to Caltech101 is the small image size of CIFAR10 have problems when generate keypoint descriptors and there are many null descriptors. So I have to use 'descriptorcorrect' function to remove these null descriptors. I'd like to try BoW with larger image size to evaluate the model performance. I don't think BoW is a good approach because the space of keypoint descriptors is huge. We can't use large image dataset to train the model. We can't expect training from a small dataset would lead to a good performance.

Improvement

As for the improvement I think other transfer learning model can be tried, such as Inception, VGG16, VGG19. VGG16 and VGG19 is larger than Resnet50 , so they are more complicate and maybe have more capability in high performance.

I have tried with VGG16 and Xception transfer learning to see whether the performance can be improved.

VGG16 transfer learning,only has accuracy=0.5 and f1score=0.6 , not beat ResNet50 transfer learning.

Xception transfer learning,only has accuracy=0.54 and f1score=0.8 , not beat ResNet50 transfer learning.