Know Your Band:

A Content-Based Image Recognition

Mobile Application

by

SUN Jingxuan

(14252392)

A thesis submitted in partial fulfilment of the requirements
For the degree of

Bachelor of Science (Honours)
in
Computer Science

at
Hong Kong Baptist University

April 2018

Declaration

I hereby declare that all the work done in this Final Year Project is of my independent efforts. I also certify that I have never submitted the idea and product of this Final Year Project for academic or employment credits.

_____
SUN Jingxuan

Date: _____

Hong Kong Baptist University
Computer Science Department

We hereby recommend that the Final Year Project submitted by SUN Jingxuan entitled "Know Your Band: A Content-Based Image Recognition Mobile Application" be accepted in partial fulfilment of the requirements for the degree of Bachelor of Science (Honours) in Computer Science.

_____	_____
Prof. YUEN, Pong Chi	Prof. XU, Jian Liang
Supervisor	Observer
Date: _____	Date:_____

ACKNOWLEDGEMENT

I would like to express my great gratitude towards my supervisor, Prof. Pongchi Yuen, who had given me invaluable advice to this project. Moreover, I would also like to thank Prof. Jianliang Xu, who is the observer of this project, for his constructive comments during the development of the project.

Table of Contents

Abstract

Heavy metal as a music genre has many distinctive characteristics in its visual representation. Metal bands' logos and album arts, which are self-segregated from those of other genres, involve highly textured patterns. In reality, such images often appear on music festival posters or in social networks such as in Facebook photos, which will attract users' attention and provide essential information about the context, but they are visually complex even for human beings to recognize. There emerges the need for a "metal image translator" due to discriminative difficulties. Thus, the objective of this project is to develop "Know Your Band", a content-based image retrieval mobile application for matching the problem domain images and offering users information about the band of interest. The client-side application is implemented on the platform of iOS. On server side, upon the abundance of raw images and data collected from the internet, object recognition, feature extraction and matching are backed with deep neural network as well as local handcrafted descriptors. An attempt of network and feature optimization is also carried out.

1. Introduction

1.1 Background

Heavy metal is a visually unique music genre. For example, many of the male artists wear long hair; for merchandise the colours black, white and red are utilized; elements such as skulls, Death and its scythe are observable in visual design. As for one of the characteristics related to this task, a typical logo always involves highly textured patterns and involve less pictorial elements. Thus emerges the need for a "metal logo translator" due to the difficulties of fans in recognizing different metal bands' logos.

From another perspective, I have always been wondering what music other fans are listening to, specifically the albums. As there are not many Chinese fans globally I have made friends with many foreign fans, and I can only know what they are listening through social network applications like Facebook, because they will post photos there. However, they will not necessarily mention the name of the album. Therefore, if I want to know exactly what the album is, I shall use “search by image”.

1.2 Objective

I would like to develop a mobile app called “Know Your Band”, which fundamentally is a content-based image retrieval system. I myself is a royal fan of heavy metal music. I propose this app according to my own needs which is also the urgent need of other metal heads. Its functionality is as follows: user inputs a real-life photo of interest in which there might be a band logo and/or an album CD. He wants to know which band is it or what the album is called and related information such as history of the band, the band’s Instagram account, etc. The proposed system would be able to return such information based on the input photo.

2. Proposed System Solution

The underlying architecture is separate between client side and server side. For client side, it is a lightweight iOS interface for uploading image through the network to the server, and receiving information from the sever. For the server side, there is a database and an object recognition engine.

Object detection. The detection engine Faster R-CNN [1]. A training procedure is to fine-tune the CNN on which Faster R-CNN is based, in this case Resnet50 pre-trained on ImageNet [2], with changing the original classification layer with a new layer with

2 classes only, to classify ground truth image of logos and album covers. Output is cropped RoIs.

Feature extraction. An extra CNN, in this case Resnet50, plus local invariant handcraft feature descriptor SIFT [3] will extract feature vector from RoIs. Feature fusion is carried out with VLAD transformation [4] on SIFT as well as CNN conv5 features.

Image Matching. Feature vectors will be stored and indexed with a ball tree. Euclidean distance will be literately calculated pairwise between the query feature vector and each ground truth. Top 10 pairs with minimum distance will be considered as the final candidate matches.

Information retrieval. Finally, related information of final matches, in advance scraped from the online databases [5] [6] and saved in local database, will be extracted from the database and returned to client.

In following sections, implementation details will be comprehensively introduced in a step-by-step fashion.

3. Data Collection

Training images for object recognition. Task is carried out by Region Proposal Network (RPN) [1] within FRCNN. These images should contain at least one album/logo object as well as objects which are not of interest, such as people, faces, background furniture etc. For example, the interested album should be marked as RoI in the situations like “a person’s holding an album”. Thus, except from collecting such images by miscellaneous keywords search, I complement the training set with thumbnails of YouTube record unboxing videos (containing albums) and heavy metal festival posters (containing logos). Training images of albums and logos are joint

together because object recognition task does not need to distinguish between the class of RoIs. Please refer to Figure 1 and Figure 2 as examples.

Training images for object classification. These images should majorly display an album/a logo, which are visually similar to the RoIs marked by RPN. If directly use the training set for object recognition, the background information might be a confusion to classification task. Also, it should be physical CDs instead of pure artwork. Such task is carried out by the underlying CNN with in the FRCNN during test time, however, in pre-processing the CNN is pre-trained separately as a single complete network, thus the separate training set is necessary. Training images for album/logo classes are separately configured. For albums, I scrape the images from second-hand metal album CDs' eBay pages; for logos, as most appearance of logos is in 2D form, that is they are printed on medium such as paper, the training images can be same as ground truths. So for higher utility of data and as Metal Archive logo images are sufficient in amount, I use part of logo images scraped from Metal Archive directly. Please refer to Figure 3 as example.

Ground truth as the training images for instance matching. Because in clustering SIFT features k-means algorithm is used, so this is an unsupervised training process, thus the ground truth set is also treated as a third training set. For albums, the ground truths are collected from the online music database Discogs, and for logos I still use Metal Archive logos. Please refer to Figure 4 and Figure 5 as examples.

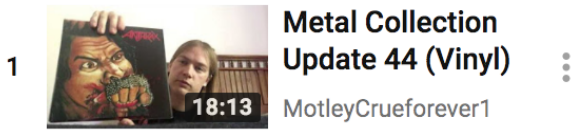

Figure 1. (up) YouTube Thumbnails Containing Albums and Other Objects [7]
Figure 5. (below) Metal Archive Logo [10] as Ground Truth

Figure 2. (left) Heavy Metal Festival Posters Containing Logos and Other Objects
Figure 3. (up) eBay Images Containing Majorly One Album [8]
Figure 4. (below) Discogs Album [9] as Ground Truth

Test images. I depart from the ground truth sets to collect miscellaneous images containing the same albums/logos. For instance matching and object recognition, I prepared separate test sets, namely matchtest and end2end. The end2end images are mostly collected from posters and images under the tag #NuclearBlast30th [11] from various social networking systems.

I collect the band information and image URLs with multiple scrapers. For writing the scrapers, I use Python 3.6 with major libraries of *request*, *urlopen* (both included in *urllib2* in Python 2), *re* and *bs4* (BeautifulSoup).

For downloading the images from scraped URLs, I use *wget* command, Google Chrome extensions of *Open Multiple URLs* [12] and *Bulk Image Downloader* [13], and an open source software called *JDownloader2* [14]. Posters are directly downloaded with the library of *icrawler* [15] without scraping URLs.

4. Initial Training of FRCNN

The training of FRCNN involves 2 steps, namely, pre-training underlying CNN and training a new FRCNN.

4.1 Environment

The whole training process is done on Linux with NVIDIA Tesla K80 GPU offered by the department. To be compatible with the system setting, codes are written in Python 2.7. Major libraries and frameworks used are *numpy, keras, pickle*. Backend of Keras is *TensorFlow*. Results are tested and shown with *cv2*.

4.2 Pre-training CNN

The essence of pre-training CNN is to fine-tune the top layers of the pre-trained ImageNet networks. According to [16], the top layers are replaced with the classifier

of only 2 classes. The network of choice is Resnet50, while VGG16 has also been tested. Detailed procedures are listed as follows:

i. Image pre-processing with *ImageDataGenerator* (carried out transformations of rotation, shift, rescale, zoom, flip, etc.).

ii. Load the whole model of Resnet50, that is including the top fully connected layers. Input image size is 224*224; stride is 7; convolutional blocks activation function is ReLU; pooling method is maxpooling; top layer classifier is softmax.

iii. Run the training images through the network with ImageNet pre-trained weights. Initializing weights are obtained from [17]. Save bottleneck features to .npy files.

iv. Construct the new top model of only 2 class with softmax activation. Bottleneck features are fed as input to the top model.

v. Run the training images through the top model and start training by calling *model.fit* function. This is the time and computation demanding step. Save the top layer weights to .h5 file.

vi. Fine-tune the last convolutional block for better performance. In Resnet50, I freeze the first 3 convolutional blocks (147 layers) and fine-tuned the rest 1 convolutional block together with the top classifier block (35 layers). Save the weights as final model for further use.

CNN was trained with 6000 images of each class, and for one class the images are separated into training and validation set at a rate of 1:3. Step 5 and 6 were run through 100 epochs and get the accuracy above 0.95 and loss below 0.01. There should not be the problem of overfitting because the training set is rather large. I believe the performance of the CNN is good enough. The output model weights will be used for training FRCNN.

4.3 Training A New FRCNN

According to [1], there are 2 approaches of training a FRCNN, namely Alternating Training and Approximate Joint Training. Alternating Training refers to separately training the RPN and the CNN, using the output RoIs of RPN as the input of CNN. Approximate Joint Training refers to the end-to-end training of the whole FRCNN. The evaluation metrics include number of overlapping bounding box, classification accuracy, loss of RPN, loss of CNN. Good performance is yielded through a balance of the four criteria, and the stop of training is a man-made decision. Considering the technical difficulty and the performance, Approximate Joint Training was chosen to reduce the training time.

During implementation time, I manually label the training set with the help of a software called *RectLabel* [18] on the Mac platform, and transform the bounding box information from .xml to the training data in .txt format as input to the FRCNN. Training set is of the size of 2500 images, which is a combination of logo and album, and in total 6092 objects. Please refer to Figure 6.

After 500 epochs in total, the network achieves a loss of 0.372613697442 as shown in Figure 7.

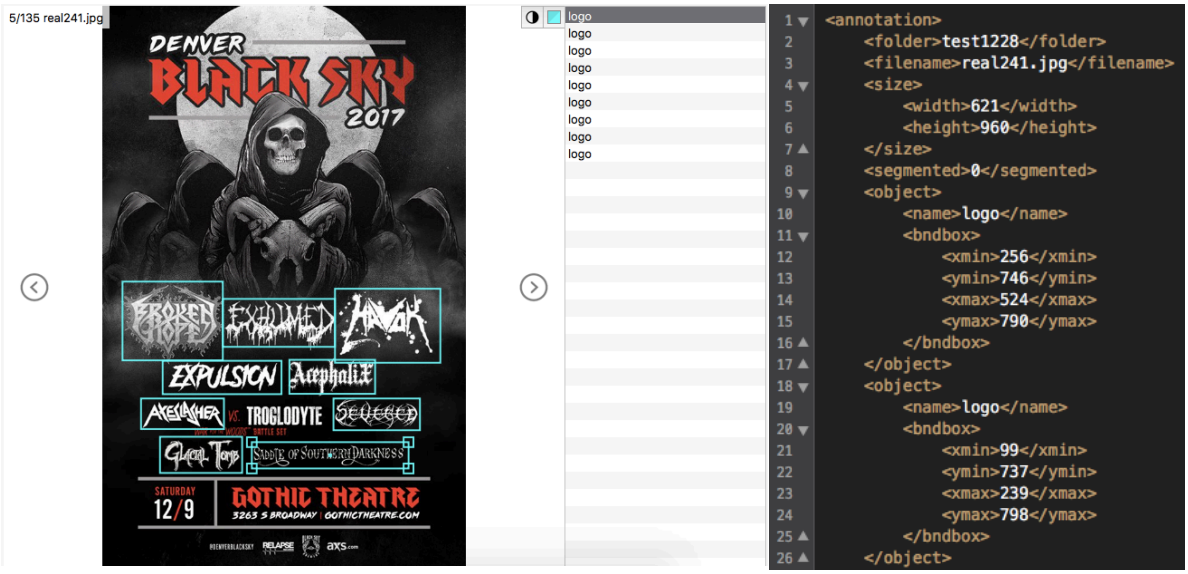

Fig. 6 Bounding Box Annotation with RectLabel and xml

```
Epoch 298/350
 999/1000 [============================>.] - ETA: 1s - rpn_cls: 0.2556 - rpn_regr: 0.0404 - detector_cls: 0.0440 - detector_regr: 0.0059Average number of over
lapping bounding boxes from RPN = 25.261 for 1000 previous iterations
1000/1000 [==============================] - 1522s - rpn_cls: 0.2557 - rpn_regr: 0.0404 - detector_cls: 0.0440 - detector_regr: 0.0059
Mean number of bounding boxes from RPN overlapping ground truth boxes: 25.239
Classifier accuracy for bounding boxes from RPN: 0.98315625
Loss RPN classifier: 0.281904333636
Loss RPN regression: 0.0405812064885
Loss Detector classifier: 0.0440586159273
Loss Detector regression: 0.00606954138941
Elapsed time: 1522.04713583
Total loss decreased from 0.384222835034 to 0.372613697442, saving weights
Best weights so far saved to ./model6.hdf5. best_loss = 0.372613697442
[[  5.40990470e-08   3.61807202e-03   9.12488922e-02   7.39599857e-03
    9.68750000e-01]
 [  1.04356118e-07   4.39091511e-02   5.59981167e-02   2.77232495e-03
    9.68750000e-01]
 [  2.52584994e-01   1.00336432e-01   3.02801607e-03   6.93191495e-03
    1.00000000e+00]
 ...,
 [  5.90860658e-08   2.74258386e-03   1.20253884e-03   4.21987521e-03
    1.00000000e+00]
 [  6.19410301e-08   1.87421683e-02   1.69737056e-01   3.37358974e-02
    9.68750000e-01]
 [  5.59893074e-08   6.67789802e-02   1.09739833e-01   2.01013666e-02
    9.68750000e-01]]
```

Fig. 7 FRCNN Final Performance

4.4 Human Evaluation and Problems

For initial evaluation, I shunted 40 images from each training set before training and tested the network. It is able to crop out most of the album regions correctly. However, many crops of logos only cover partially of the whole logo. As suggested by supervisor, I have tried to calculate several enlarged RoI region vertically and horizontally in proportion to the original RoI. However, as the number of RoIs increases exponentially, it will in the end drag the end-to-end query time. On the other hand, in later tests, even partial logos are able to be classified and matched from VLAD descriptors, so the enlarged RoI approach is aborted at last. Additionally, small logos cannot be recognized.

5. Pre-processing for Instance Matching

2 major steps involved are VLAD extraction and database formation.

5.1 Environment

For VLAD extraction, as it involves unsupervised k-means clustering, this step is done on Linux with NVIDIA Tesla K80 GPU offered by the department. To be compatible with the system setting, codes are written in Python 2.7. Major libraries used are *numpy, pickle, cv2, sklearn*.

For database formation, simple data formatting is carried out in Python 3.6 on Macbook Air. A software program called *csv-to-sqlite* [19] command is used to build the database from formatted data.

5.2 VLAD Transformation on SIFT Features

Having obtained a thorough understanding of VLAD descriptors from [4], I followed [20] to carry out the feature extraction and related procedures. Detailed procedures are listed as follows:

i. Read in ground truth image one at a time. Extract SIFT features with built-in functions of cv2. Dump SIFT features from all ground truths into a .pickle file.

ii. Use built-in k-means algorithm of sklearn to cluster the SIFT features. The centroids become “words” of the dumped “visual dictionary”.

iii. Read in ground truth image one at a time again to calculate the VLAD vector of this image according to the distances of each SIFT feature to the “words”. Now for each image, the keypoint-as-unit SIFT features are converted to image-as-unit single VLAD descriptor. Dump all VLAD descriptors into one .pickle file.

iv. Use built-in ball tree algorithm of sklearn to index the VLAD descriptors into a tree structure. For clarity, 2 trees are built separately for searching albums and logos.

5.3 Problems of Database Configuration and Solutions

Bands with the same name. For filtering, I go into the discography page of the bands’ home page on Metal Archive and scrape their whole discography. If the query album is within the band’s discography, then such band would be the correct match and related information is returned.

Exceptions. cv2.error are met because some of the crops have x/y values below zero, which causes the cropped image to be empty. Current solution is to jump pass the empty crops, but it might cause the correct RoIs to be jumped pass.

Naming inconsistency. Records with double artist or double album name have naming inconsistency between the 2 online databases I collect data from, namely Discogs and Metal Archive. For example, the following album name uses “:” as splitter in Metal Archive while it uses “()” in Discogs. Solutions to such problem can only depend on exhaustive empirical tests, which is hardly possible within the human resources of this project, thus is currently left unsolved. Please refer to Figure 8 and Figure 9.

Varied versions of logos of same band. Along years of splitting up and reforming, many bands may have two or three different logos. The specification of Metal Archive is to upload all logos of certain band within one logo image, so that in one logo ground truth image there might be multiple logos, which will cause confusion in matching process. Solutions to such problem can only depend on exhaustive empirical tests, which is hardly possible within the human resources of this project, thus is currently left unsolved. Please refer to Figure 10.

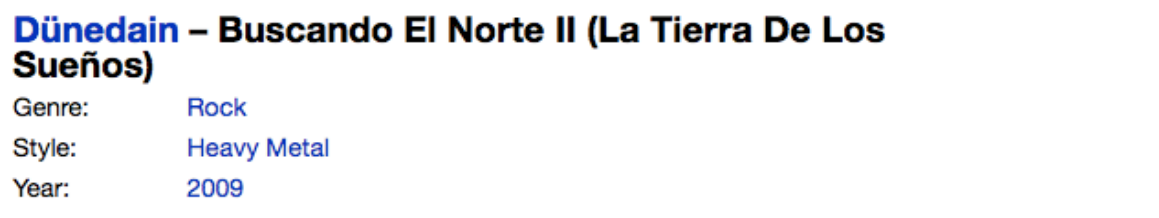

Figure 8. Naming of Metal Archive [21]
Figure 9. Naming of Discogs [22]
Figure 10. Multiple Logos in One Ground Truth

6. Initial Experiments of Feature Fusion

For the purpose of improving performance, combinations of different sets of features, both low-level and high-level, are tested in the query procedure.

6.1 Preparing RoI Features

Among all combinations, 4 used RoI features output from the self-trained FRCNN. So these 4 tests (ii-v) are carried out in an end-to-end fashion, and conv5 feature maps are used as RoI features.

For other combinations, the found RoIs are cropped out and written out as images. In the tests, query functions extract other types of features by reading in the crops and calculating along according methodologies.

6.2 Colour Histograms

As suggested by supervisor, colour information of the images might be supplementary to the content based features. It is an intuitive voting mechanism to raise the possibility of some of the matched images. Thus, built-in functions of cv2 are used for calculating colour histograms.

6.3 Detailed Experiments and Evaluation

For initial evaluation, successful rates are estimated on a few test images. For each crop, 10 most similar images are retrieved. Successful cases are those in which the expected match appears in the 10 images. Please refer to Table 1.

After comparison, original VLAD descriptors are used as extracted feature due to higher successful matching rate.

According to [23] [24], the combination of scene-level CNN feature and VLAD should yield a high successful matching rate. Also, PCA should be helpful in raising successful rate while reducing the dimensionality of the features. The rationale for an opposite result is that, within the self-collected image sets, each two instances are

quite different visually from each other, which is different from current instance matching benchmark datasets, most of which contains several visually similar images of one object. Therefore, from machine's "point of view", even it might have done proper retrieval, from human perspective, each result still "looks" quite different from the query image.

Additionally, PCA works well when there are redundant features. However for a set of images among which each is very different, there should not be much "redundant" features. This is my hypothesis why PCA hasn't worked.

Furthermore, the concatenation of CNN features and VLAD descriptor will cause considerable increase in total distance to certain image. This increment might cause the rank of similarity to drop behind some former not-so-similar images. This is my hypothesis why combination of CNN features and VLAD descriptors hasn't worked.

No.	Merged Features	Successful Rate %
i	VLAD	100
ii	VLAD + Object-level FRCNN features	22
iii	Object-level FRCNN features	22
iv	FRCNN column features	22
v	Object-level + FRCNN column features	22
vi	VLAD + Object-level features of Pre-trained CNN	56
vii	VLAD + Column features of Pre-trained CNN	67
viii	Object-level features of Pre-trained CNN	56
ix	Column features of Pre-trained CNN	67
x	Object-level + Column features of Pre-trained CNN	33
xi	VLAD + Color Histogram	0
xii	Color Histogram	0
xiii	VLAD with PCA (6400 components)	<30 (didn't keep record)
xiv	VLAD with PCA (6400 components) + Object-level features of Pre-trained CNN	<30 (didn't keep record)
xv	VLAD with PCA (12800 components)	<30 (didn't keep record)

Table 1. Initial Experiments with Feature Fusion

7. Initial End-to-end Test and Evaluation

Item	Remark	Number
Album ground truth image set		500
Logo ground truth image set		500
Test images		155
All crops	Crops = RoIs written to file	426
Correct crops	Crops containing album or logo	316
Matched crops	Correct band information retrieved	212
Recall of object recognition	Recall = correct RoIs/correct RoIs+RoIs haven't been recognized (multiple crops on one object is counted once)	159/192 = 83.68
Precision of object recognition	Precision = correct crops/(correct+wrong) crops	170/426 = 39.91
Overall accuracy		33.40

Table 2. Initial End-to-end Result

Complementary notes:

1. Deviation might be caused by incomplete database. Some albums/logos are not included in database but recognized in the query image, such as very small albums in the background.

2. mAP of matching is calculated at k=10.

3. From human perspective, performance is well in different lightning conditions and colours, but not so good if the album/logo is flipped (mostly horizontally).

4. From human perspective, performance is well in visually simple images, but not so good on complex images. Please refer to Figure 11 and Figure 12 as examples.

5. Partial logos can be recognized and matched correctly, when the visual characteristics of the logo is distinguishable. For example, in Figure 13 the logo of “Wintersun” is on the girl’s hoodie and it’s only part of the logo, but system recognized and matched it correctly at the second most similar choice. In contrast, “Avenged Sevenfold” logo in Figure 14 looks quite similar to plain text. And as it is a bit small, the network failed to recognize it even though it is an intact logo. Thus the matching result varies from image to image due to the characteristics of the logos

themselves. Due to the same reason, the network cannot distinguish between logo and landmark sometimes. Please refer to Figure 15 as example.

Figure 11. Visually Simple Query Image

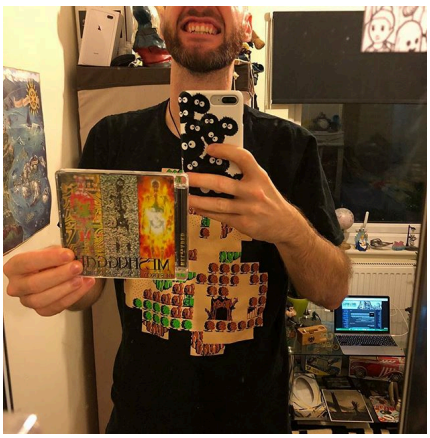

Figure 12. Visually Complex Query Image

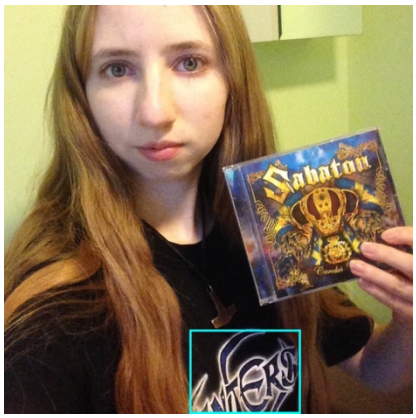

Figure 13. Partial Logo of Wintersun

Figure 14. Text Logo of Avenged Sevenfold

Figure 15. Ensiferum Crop of Trees in Background

8. Network Optimization

Major contribution of Sem 2 is on addressing the optimization issues. According to the steps in the pipeline, the focus of performance improvement moves to network optimization and more efficient feature fusion schemes.

8.1 Training Details

On analysing the training process of Sem1, problems and faults are found as follows.

Training from zero verses fine-tuning. I mistakenly considered the previous training as fine-tuning, because on loading weights, I used the pre-trained weights of Resnet50 and the convolutional layers of Resnet were loaded weights by name. To recap, I trained my own Resnet model by fine-tuning to fit my own band data. However, the FRCNN still contains RPN and classifier layers, and there are no weights for these layers in the pre-trained CNN model and "the weights" cannot be loaded. Thus, in previous training, the RPN and the classifier were in fact trained from zero as new network. Such training approach has high potential of being problematic considering the data are self-collected and the quality and quantity might not be good enough. To stress this problem, I used the weights of pre-trained FRCNN on ImageNet obtained from internet [25], loaded RPN and classifier weights from

this model, and still loaded convolutional layer weights from my own pre-trained CNN model.

Cross validation. Diving deeper into the codes, the way the original model deals with the separation of data is without cross validation, that is, all the "training data" are used in training literally [26]. Consequence of no cross validation during training is a highly possibly overfitting model. In case, I added cross validation to randomly separate the training set at a ratio of 5:1 into training set and validation set in each epoch.

Data augmentation. One problem emerged early in previous training is the insufficiency of training data. Such problem becomes even severer when cross validation is used. Compared with the simplest way of feeding data in previous training, I added data augmentation of image rotation, shifting and flipping during retraining. For derived problem, see complementary notes below Table 3.

Human evaluation. Actually, the statistical results of FRCNN from last semester is obtained from mere human evaluation, that is, given several test images (no more than 30), the detected RoIs are shown within the image with the help of OpenCV function imshow. RoIs containing target album or logo were considered as successful detections. The number of successful detection was counted and confusion matrix was manually calculated. Such non-

scientific calculation is merely for illustration purpose, and the statistics from previous training was nothing but a coarse estimation. Thus, in retraining, I added the scripts, measure_map.py, for comprehensive evaluation of FRCNN. Main function is to calculate the mean average precision after all the training epochs.

Generalization ability. In previous training, only final optimal weights at stop time were saved. To balance between underfitting and overfitting, I tried to save weights on each accuracy update. Accordingly, mAP measurement was inserted at the end of such updating epochs instead of the original one measurement at stop time. With these diagnostic statistics, I tried to find the model with peak generalization ability by plotting training accuracy and validation accuracy.

In summary of the above faults and solutions, 7 trials of retraining are carried out and the results are as follows:

Trials	AP album	AP logo	mAP	loss	Rank
finetune + cross val + final mAP	0.875773854135	0.685576560585	0.78067520736	1.34752395183	2
finetune + aug + final mAP	0.75902475123	0.696653130193	0.727838940712	2.66465598715	6
finetune + cross val + aug + final mAP	0.871920340588	0.667815494874	0.769867917731	1.42401295861	4
finetune + cross val + per optimal mAP + gen/plt (best)	0.893460460836	0.727167063276	0.810313762056	1.2770022666	1
finetune + cross val + per optimal mAP + gen/plt (2nd best)	0.814064204169	0.733884851467	0.773974527818	1.37275927728	3
finetune + cross val + aug + per optimal mAP + gen/plt (best)	0.846656380608	0.645819069931	0.74623772527	1.55319615773	5
finetune + cross val + aug + per optimal mAP + gen/plt (2nd best)	0.727294211281	0.688764544586	0.708029377933	2.89686328396	7

Table 3. Trials after Adjusting Network Configuration

Complementary notes:

1. Judgement of best model is based on mAP as well as loss. Ideally and fortunately, there's no case where high mAP has high loss. For full statistics on optimal weights update each epoch, please refer to plt_stats.py.

2. The loss is universally lower than the experiments in Sem 1. Possible reasons could be the introduction of cross validation and/or data augmentation.

3. A lower mAP can be observed when introducing data augmentation. Though such approach might alleviate the problem of training data insufficiency, noise has also been enlarged, for equally applying rotation, shifting and flipping on different input images would be thoughtless. Different input images have different content, which after augmentation the resulting images might not even be considered as photorealistic, for example the figure might be severely distorted. Thus, it might cause a worse performance of the network.

Conclusion can be drawn in this comparison step that fine-tuning (relative to training from zero) with cross validation and without data augmentation would give the optimal solution of the neural network. Furthermore, the best two models are chosen for trials in feature fusion procedure, both implement the mentioned setup essentially.

8.2 Hyperparameter Optimization

Another optimizing approach been tried out is hyperparameter optimization.

Hyperparameters refer to the parameters in a model which cannot be “learned” through model training, or so to say their values are out of the scope of training and are human-defined based on experience. Hyperparameters are usually fixed before the actual training process begins [27].

Hyperparameter optimization is essential, because these parameters are quite a lot in amount and have equal influence on the performance of a network as the trainable parameters. Also, while a network has reached its bottleneck in performance and the loss is "nearly converged" and won't easily decrease anymore, hyperparameter optimization might make the change and become the breakthrough.

However, as there are no defined rules of how to choose the values of hyperparameters, the near optimal combination can only be found out through multiple trials, which might be gruesome and time consuming. In actual implementation, a Python library called Hyperopt [28] is utilized to carry out the automatic hyperparameter optimization so that I can free my hands a bit. It is "for serial and parallel optimization over awkward search spaces, which may include real-valued, discrete, and conditional dimensions" [29]. For example, it can use the Tree-structured Parzen Estimator (TPE) algorithm, which explore intelligently the search space while narrowing down to the estimated best parameters [28].

8.2.1 Automatic Searching Steps [28]

There are several steps needed to pave the way for automatic searching.

First, define a hyperspace. The hyperspace is a dictionary contains all the hyperparameters to be defined a value of. The expression of each hyperparameter is to determine the range of

possible values and the probability distribution. For example, kernel size of convolutional layers can be defined in uniform distribution with the range of 3 to 7, while learning rate would be better defined as a loguniform distribution with range 0 to 1.

Second, define an objective function, Fmin, for minimization. In network training, the one and simple idea is to use the loss function as the objective, because we always want to minimize the loss. In actual implementation, you can just copy the evaluation functions after each epoch to define the objective.

Third, define the trail object attributes to be saved. A trial object is a built-in type of database object, which is similar to the history object for a Keras model. In actual implementation, the diagnostic statistics can be in one way obtained from history then saved in trials.

After changing the hardcoded hyperparameter values into the expression of hyperspace variables, the training process can be started the same as without hyperopt.

8.2.2 Choice of Hyperparameters

For my network, I find the following hyperparameter space (in total contains 9 hyperparameters) to search in: RPN learning rate, RPN decay, classifier learning rate, classifier decay, whole model learning rate, whole model decay, Resnet convolutional layer

kernel size, anchor box scales, and number of RoIs for each round of calculation (similar to the idea of batch size).

Learning rates. Learning rates and decays are classic hyperparameters for tuning. Learning rate refers to the step size with respect to gradient descent in backpropagation aiming to find the minimal loss. The lower the learning rate, the smaller step it goes each time down the slope. To lower the learning rate can in some degree ensure there is no missed local minimum, but in return it takes more time in training generally [30]. For this particular task, the primary purpose is to make the final matching as accurate as possible, so comparably smaller learning rate would be suitable.

Decays. In training deep networks, it is usually helpful to anneal the learning rate over time. Good intuition to have in mind is that with a high learning rate, the system contains too much kinetic energy and the parameter vector bounces around chaotically, unable to settle down into deeper, but narrower parts of the loss function [31]. In Keras model, the 1/t decay is used, defined as the mathematical form $\alpha=\alpha 0/(1+kt)$ where a is the learning rate, a0, k are hyperparameters and t is the iteration number [32]. In original FRCNN setup, the decay values are default set as 0. I try to introduce decay to be paired up with different learning rate to achieve a reasonable performance.

Kernel size. An image kernel is a convolutional matrix whose function is to be moved along the image as a sliding window to see if the image contains the filter pattern at each searching location. It is functionally similar to a filter and thus comes its other name, filter. The output of a filter is the activation value, that is if certain location is highly likely to contain the filter pattern, the region will have a single and high value to represent itself, otherwise a lower one. So after going through the filters, the output dimension generally would be smaller than the input dimension (here comes the object level feature and column features to be used in feature fusion step), and for each "pixel" in the back layers it can represent a large area of the image from former layers, whose size is called the reception field. There are multiple filters in each convolutional layer. Kernel size refers to the width and height of a filter. A larger kernel size would shrink the input image quicker with fewer layers, the reception field would be larger and the output feature is a more concentrated representation which retains more local information of the image. Considering the nature of training images of this particular task, visually complex images is thought to be paired up with better "information preserver", thus I tend to test with an increase in kernel size.

Anchor boxes scales. Anchor boxes are templet generated boxes with predefined sizes used in RPN. Correction on top of anchor boxes are used to match with the ground truth bounding

box dimensions. For default FRCNN setup [1], 3 different sizes and 3 different ratios are defined as the templet in model configuration files. Considering the fact that the logo images are mostly flat rectangles, and that some logos are small in size compared with the whole input image, smaller sizes and more extreme ratios are defined during hyperparameter optimization.

The anchor box scales and number of RoIs are defined within the configuration file of the model, so there is no need to re-define these two within the hyperspace. For the rest 7 parameters, all learning rate and decay are defined in loguniform with range 10^{-6} to 10^{-3}, kernel size of convolutional layers are defined in uniform distribution with the enumerate value from 3 to 7.

Hyperparameter	Original value	Distribution	Range
RPN learning rate	10^{-5}	Log uniform	[10^{-6}, 10^{-3}]
RPN decay	0	Log uniform	[10^{-6}, 0]
Classifier learning rate	10^{-5}	Log uniform	[10^{-6}, 10^{-1}]
Classifier decay	0	Log uniform	[10^{-6}, 0]
Model learning rate	10^{-1}	Log uniform	[10^{-6}, 10^{-1}]
Model decay	0	Log uniform	[10^{-6}, 0]
Conv kernel size	3	Quniform	[3, 7]
Anchor box scales	[128,256,512]	Choice (enumerate)	[32,64,128,256,512]
Number of RoIs	32	Choice (enumerate)	[4,8,16,32,64]

Quniform = round(uniform(low, high) / q) * q [28].

Choice = returns one of the options, which should be a list or tuple.

Table 4. Choice of Hyperparameters

8.2.3 Implementation Details and Results

Other utility functions are saving the hyperparameter combination of each trail. Json files are output with randomly generated model id number and final loss as file names. It is for the ease of choosing the best models. Finally, the best two models chosen from the comparison step are used in hyperopt experiments. The loss value can be directly observed from Json file name.

Figure 16. Example of JSON output

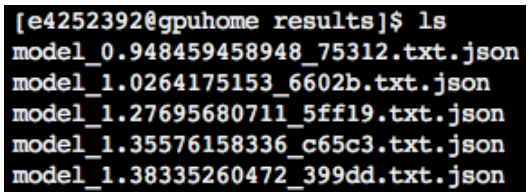

Figure 17. Top 5 JSON output

Because there should be a retraining step after finding the best combination of chosen hyperparameters, each round of hyperspace random search was set to run for 20 epochs only, just for the purpose to illustrate the learning potential of the network under certain setting. Such approach aims to eliminate unnecessary time consumption, but it might be problematic.

First, each model new setting is trained from zero because the architecture of the layers might change (due to search in kernel size), and there is certainly no time to first pre-train the setting with well-defined data sets such as ImageNet image sets, so no pre-trained weights can be obtained to serve as the base of fine-tuning.

Second, also due to time limitation, only a few rounds of hyperspace random search were carried out. To alleviate this issue, number of epochs was set to be 40 at the start, which is quite near the best generalizing stage of the network according to former observation and though I was not certain if the network would reach similar stage, which is highly unlikely when there is significant change in architecture and network is trained from zero, but changed to 20 afterwards. In the end 22 search results was output and saved and no more time could be afforded to continue the trials.

Finally, a far more essential problem also emerged in retraining step due to my lack of consideration. Similar to the first problem, the thorough retraining on best hyperparameter setup should also implement fine-tuning instead of training from zero, that is the network in retraining should be in same condition as previous FRCNN training. Although the task changed from many networks in the first problem to one network here, to obtain pre-training weights on ImageNet is still considered to be infeasible. Actually, I haven’t noticed the first

problem until I planned to start retraining. Even though I forced retrain with the setup, the resulting network would not be compatible to be compared with the other 7 in the comparison step. Therefore, finally the retraining step was aborted after all the effort.

The positive point is that, it seems for this particular task with consideration about the data, a larger kernel size than what is used in the original setup would be more suitable on highly textured image patterns. The rationale behind could be that reception field should be larger to keep more local information.

In summary, though it seems to be a waste of two weeks' time because hyperopt contributed nothing to final network, it still deserves to be considered as a fruitful learning experience which is supposed to be useful in actual production and research. Also, the hyperopt library is a great tool to do automatic network optimization when encounter a bottleneck.

8.3 Evaluation of Network

With the 7 networks obtained and the best as an illustration, evaluation and visualization are carried out with same test images as the first try of training a FRCNN in Sem 1. Again, here I list the statistical results and some observation from RoI detection results.

Best setup	AP album	AP logo	mAP	loss	Recall	Precision
finetune + cross val + per optimal mAP + gen/plt (best)	0.893460460836	0.727167063276	0.810313762056	1.2770022666	0.9375000	0.8075117

Complementary notes:

1. Query image set size = 155, with one or more albums or logos and background in each image.

2. Precision: detected correct / all detected = 344/426.

3. Recall: detected correct / (detected+not detected) = 270/288.

Improvements that can be observed are: higher detection accuracy, recall and confidence; more precise edge detection; higher recall on tiny objects (small logos). However, the general precision becomes lower, in which case non-relevant text on posters or in the background or rectangle-shaped items such as one frame of bookshelf may be considered as logo or album.

From my own optimistic perspective, such phenomenon in return proves a better performance of the network essentially, because even for human without problem domain knowledge, these patterns could also be easily classified as one of the classes. Thus, the discriminative ability is also a data related and problem domain issue, which might need some hand-crafted descriptors exclusive for heavy metal band logos and album arts. Please refer to Figure 18 and 19 as examples.

The conclusion best accuracy of the network is 0.81.

Figure 18. (left) Network misclassifies cupboard as album

Figure 19. (below) Network misclassifies logo-resembling text as logos

9. Optimization on Feature Fusion

After the selection of best 2 models from above optimization steps, feature fusion is carried out on CNN features and SIFT features.

9.1 Problems in Fusion Process

There are several problems with this pipeline step in Sem 1 experiments as well. To explain them, here I first list the steps of previous merging scheme:

Step 1, branch 1	SIFT	a. Describe SIFT features on all training images b. Kmeans clustering on all SIFT descriptors from all training images to build visual dictionary c. Calculate VLAD vector for each image based on visual dictionary
Step 1, branch 2	CNN	d. Extract conv5 feature as object level feature e. Extract conv4 feature as column feature
Step 2	Concatenation of CNN feature and VLAD descriptor	
Step 3	Index ball tree for searching	

Table 5. Steps of merging features

Input image size. Main problem is with respect to CNN feature dimensionality, that is Step 1 and 2. The default setup in original FRCNN is to resize the input image to enlarge the shortest edge to 600 pixels and keep the original width-to-height ratio. However, in Sem 1 when extracting CNN ground truths, for simplicity, I set the input image size as 224*224 pixels which is the default setup in Resnet50 but not FRCNN. Intuitively, the square size would cause geometric distortion to some degree. Further, successive problem of

incompatible dimensionality caused by different input image sizes will emerge, especially for logos because each has a different length of characters (it is not an issue with album reference images because from the single online database, Discogs.com, where the images were scraped, all the cover image are square and in similar size at around 300*300 pixels). This problem has been overseen because in the experiments of merged feature in Sem 1, the final achievement was obtained with merely VLAD on SIFT and there was no reference on CNN features, because the merged feature including CNN feature were considered to be inferior in performance and were not chosen, thus there was no dimensionality incompatibility in the final pipeline.

Aiming to do it thorough and the right way, this time input images to CNN and to FRCNN query are equally resized to 600 pixels on the shortest edge. However, this will result in different feature vector dimensions on different images. I come up with an alleviation method to unify the vectors. As the shortest edge is indeterminate between width and height, simple dimension reduction methods such as PCA would not work. Also, the convolutional feature maps are intra-connected with location information, it would not be rational to implement PCA on feature maps. So, considering the reception field and the relative location on the image, I cut the dimension values which represent the information of the right bottom corner

of the image from the feature vector, if the dimension is higher than my setup; then pad the vector with zeros, if the dimension is lower than my setup. In implementation, the setup size for album and logo images are different, namely (1,2,2,2048) and (1,2,7,2048) for object level feature, because the shape of album images are near squares and the shape of logo images are flat rectangles. Also, the setup value is determined by the mean of number of values along each axis of the matrix. Still, such approach might cause loss of information due to the cutting-off, vector sparsity and slower calculation speed due to the padding, so it is kind of an expedient, a method out of no methods.

9.2 Fusion mechanisms

9.2.1 Simple concatenation

Implementation details are as follows. First, with the best two models, trials on more comprehensive test images are carried out on several most potential feature combinations from Sem 1 experiments, with the updated feature dimensionality. Following the methodology of Sem 1, in this step the features are simply concatenated.

Trials	AP album	AP logo	mAP	loss	Rank
finetune + cross val + per optimal mAP + gen/plt (best)	0.893460460836	0.727167063276	0.810313762056	1.2770022666	1
finetune + cross val + final mAP	0.875773854135	0.685576560585	0.78067520736	1.34752395183	2

Table 6. Best 2 models used in feature fusion processes

	Album		Logo	
	Dimension	Accuracy	Dimension	Accuracy
SIFT(VLAD)	65536->65536	0.8877551	65536->65536	0.49215686
CNN object level feature (original weights)	2048->8192	0.41326531	2048->28672	0.33529412
CNN object level feature (model 1)	2048->8192	Fail	2048->28672	Fail
CNN object level feature (model 2)	2048->8192	Fail	2048->28672	Fail
SIFT(VLAD) + CNN object level feature (original weights)	67584->73728	0.41326531	67584->94208	0.33529412
SIFT(VLAD) + CNN object level feature (model 1)	67584->73728	Fail	67584->94208	No trial
SIFT(VLAD) + CNN object level feature (model 2)	67584->73728	Fail	67584->94208	No trial

Table 7. More comprehensive experiments on feature fusion

Complementary notes:

1. Query image set size = 300, with 200 albums and 100 logos.

2. No PCA used due to mentioned reasons.

3. Accuracy of CNN object level feature and SIFT(VLAD) + CNN object level feature are the same. In more details, all the retrieved images for each query image are the same and in same order. Thus, after the trials on album images, trials with the same feature combination on logo images were not carried out. Pointing to same set of images then failing in matching might be cause by vector sparsity mentioned above.

4. A problem with the pipeline considering calculation speed is, if CNN and SIFT merged feature is used, the whole pipeline would become FRCNN (object detection) + CNN (feature extraction) + SIFT (feature extraction), thus an extra pre-trained CNN is needed, which would be slow and redundant. Therefore, if other fusion mechanisms are not good enough to boost the accuracy, balancing the accuracy and speed, original SIFT feature might be chosen to be used in final pipeline.

In conclusion, though the chosen feature is not changed, higher overall accuracy can be observed compared with the results from Sem 1 (please refer to Table 1). SIFT feature on album matching could be considered as a successful case. On the other hand, for logo matching, the 20 matched pairs are different with SIFT trial and CNN object feature trial, so it could be considered that merging SIFT and CNN using other mechanism might have chance to boost the accuracy, thus comes the following experiments.

9.2.2 Using FC features

From previous experience, I extract two kinds of CNN features, namely object level features and column features, based on my understanding on [23]. On considering alternative merging approaches and rereading the literature, I found that I have been mistakenly treating the conv5 layer features as the so-called “object level features”. In the paper, object level should be the equivalence to the one-axis feature vector obtained from fully connected layer whose dimension is equal to the number of classes. As a FRCNN doesn’t have a fully connected layer, I mistreated conv5, the last layer in FRCNN base layers, as the FC layer. Thus in fact, the “object level feature” in my scenario is also column feature, which is essentially similar to conv4 feature but with a higher abstraction level. That is to say, in previous trails I have not used FC features, which is a global representation of the query image, in the concatenation process. Therefore, for the purpose of comprehensive testing, I introduced FC features into the fusion.

After fixing some complications with the model weights, the top layers of original Resnet50 are added to the CNN feature extraction pipeline. (The top layers were omitted in Sem 1 because essentially I didn’t use top layers, which turns out to be wrong.) Experiments are carried out with pre-trained CNN weights obtained from internet, because if using fine-tuned

weights on ground truth image, the final classifier only has 2 classes and the output FC feature would not give enough information about the query image. Instead, using the original configuration, the FC layer would output activation of 1000 classes, which should be representative. The difference in number of classes should not be a problem because the purpose of this step is only to obtain the representation of the image instead of classification. However, the results are not ideal.

	Accuracy
FC	0.18823529
SIFT(VLAD) + FC	0.12941176
SIFT(VLAD) with PCA + FC	0.22745098

Table 8. Experiments with FC features

9.2.3 VLAD Transformation on CONV5 Features

As explained in Section 8.2.2, to do it right, the "object level features" or the conv5 features should be accordingly pre-processed as column features. According to [24], CNN column features are similar to SIFT features in the sense that they represent local information of an image. It also introduces the pooling method of VLAD to concentrate different numbers of column descriptors in to one unified dimension representation. Essentially, this is just the idea of grouping SIFT descriptors, so it is rational to carry out VLAD pooling on CNN column features.

At implementation time, album matching is ignored because using SIFT feature only can reach an acceptable accuracy. The same combination trials are carried out on logo images with CNN “object level features” being transformed by VLAD, followed by L2 normalization and PCA transformation steps. However, there observes no significant accuracy boost. It only changes from 0.33529412 to 0.44313725 which is not ideal, so still alternative fusion mechanisms are to be figured out.

In fact, analysis has been made on this sub-ideal situation after trying out all the mechanisms mention above and below. Possible reason is that, most of the test images are very difficult. More specifically, considering the use case scenario, logos are mostly detected from music festival posters, T-shirts, or within an album, so the RoI image should be strictly near 2D, only with some noisy information in the background. In turn, for testing, I found images related to logos, such as keychains with the shape of the logos, promotional logos with artificial effects, logos with only outlines and jacket patches. Intuitively, I wished to use these images to test the discriminative ability of the system, but it seems at current stage that it is unrealistic and over-difficult.

Therefore, in a more realistic way, I obtained a new set of test images from the FRCNN test images, by manually cutting out the RoIs-to-be to serve as the input to the matching step

pipeline. It turns out that my hypothesis is correct and the accuracy becomes 0. 64705882 with realistic testing conditions.

Below is a summary of the trials on CNN VLAD features.

	Accuracy
CONV5(VLAD) on hard images	0.33529412
SIFT(VLAD) + CONV5(VLAD) on hard images	0.44313725
SIFT(VLAD) + CONV5(VLAD) on FRCNN RoIs	0.64705882

Query image set size = 102 for first two trails and 157 for the third trial.

Table 9. Experiments with VLAD transformation on CNN conv5 features

9.2.4 Cross decomposition

As suggested by supervisor, after searching for feature fusion algorithm codes, I came up with Canonical Correlation Analysis. It is a family of algorithms for measuring the linear correlation between multivariate distributions as described in sklearn library documentation [33]. In analysing multiple-X multiple-Y correlation, components which are most correlated in input 2D arrays are preserved and redundant components are discarded. Output vectors illustrate stronger linear relationship between two datasets. The algorithms are mostly used in the field of bioinformatics.

Canonical Correlation Analysis is suitable for this particular problem because the two features to be merged are extracted from the same query image, and they should have strong

correlation, so that the CCA features should be a more discriminative representation of the query image by eliminating the noisy information.

Upon the basic understanding of CCA, the implementation is simple by calling the library on the raw feature vectors. However, the result is a failure.

9.3 Conclusion

Finally, CNN conv5 features with VLAD transformation is chosen to be implemented in the server-side engine balancing the time consumption and accuracy. The best average accuracy on album and logo class is approximately 0.76.

10. Final End-to-end Test

The last step before embedding the pipeline into the server-side engine is to carry out the end-to-end test combining the FRCNN for object detection, extra CNN for feature extraction, and the searching index structure. With a mixture of 144 test images containing multiple albums and logos, the pipeline reaches a reasonable accuracy of 0.65 (93/144), considering FRCNN and CNN each have an accuracy around 0.80.

11. Client-server Configuration and Application Development

The final step of this project is wrapping the whole engine into a web server and connect the client to it. The micro server is set up within the Python framework of Flask [34], which is based on Werkzeug, Jinja 2 and good intentions.

11.1 Application Logic

The client side mobile application is developed on the platform of iOS, using Swift in Xcode9. The architectural pattern of model-view-controller [35] is implemented to group the band information into an object, which illustrates an object-oriented approach.

The main storyboard contains 4 scenes, namely default scene on application loading, image uploading scene, result candidate list scene and detailed information scene. Each of the views corresponds to a controller file containing the logic.

ViewController. The global controller, ViewController loads the default scene with two alternatives of choosing an image, taking a photo with built-in camera and choosing an image from photo library. After choosing the target image, clicking upload would send the image to server side for detection and matching. While client is waiting for the results, it's time for the server engine to work. Albums and logos are detected, regions of interest are cropped, reference images are matched and band information is retrieved… all the same as described

in the pipeline. Finally, the retrieved information is formed into json objects and cached as a json file. The URL of the json file is sent back to client application. The global controller gets the URL and passes it to the list view controller, and thus finishes its work.

ListViewController. The list view controller, ListViewController, has the function of displaying the candidate bands' information in an organized way within a table view. So essentially, it is an object of ListViewController class, which is a self-defined subclass of the built-in class TableViewController. On obtaining the json URL, the controller accesses and reads the content of the json file, and caches each entry with variables, namely band name, album name, band logo URL and album cover URL. Album name and URL are optional when the returned information is about a logo. Band logo URLs and album cover URLs are the original URLs which I scraped information from during data preparation step. Information entries of one band will be displayed in one table cell. Number of table cells is decided upon counting total number of json objects. For each cell, the list view controller gets the information entries and creates an InfoCell object.

InfoCell. The custom cell class, InfoCell, is a self-defined subclass of TableViewCell, for the purpose of setting a custom layout of each cell. I design that each custom cell contains the reference image returned by the server and other string type information, so there should be a

custom image view in the cell, which is not contained within any default type of table cells. Each cell will display the image by downloading data from the returned image URLs.

DetailViewController. To check out the detailed information and web page of the correct band from the candidate list, user can click on one info cell. The action will trigger the application to load the detailed view, thus call the detail view controller, DetailViewController. It displays more entries of information.

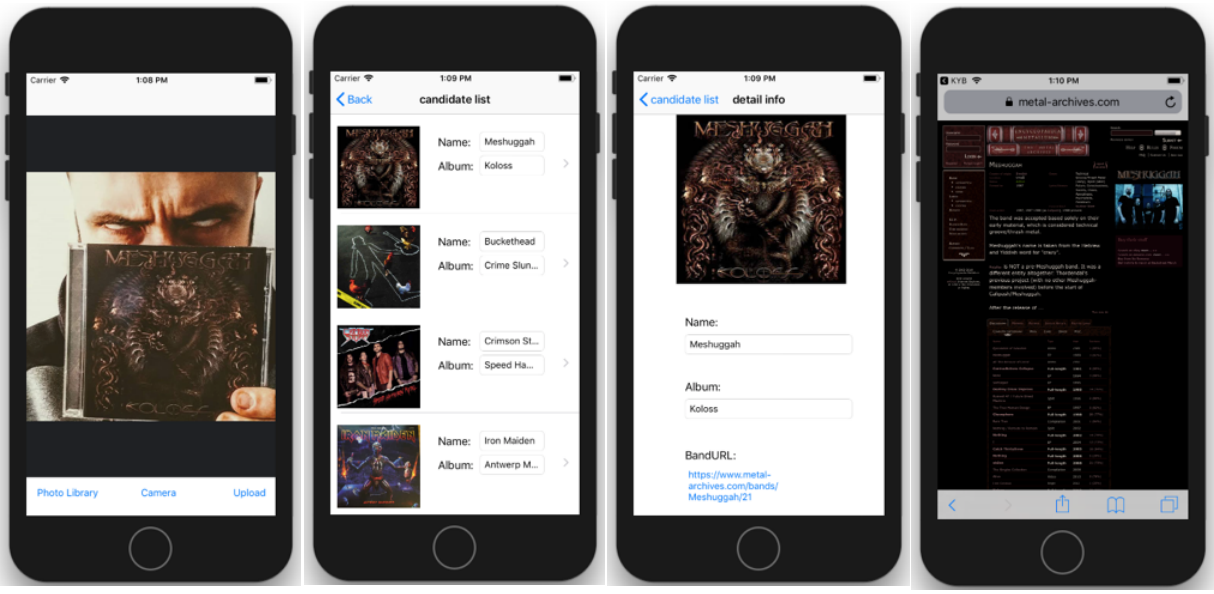

Figure 20. Mobile application screenshots

11.2 Future Improvement

As the server is set up on my personal computer and not yet deployed to cloud service or department server, there might be several defects which deter the performance of the whole system.

First, as mentioned, the URLs returned to client are original URLs from different online databases. Downloading images from different sites orders multiple HTTP connections, which will considerably slow down the process and is not safe. It also makes the system bound to the possible change of those servers, for example syntax change of image URLs. It is not optimal to download the images this way, but the thorough development of a full-fledged server is beyond the scope of this project.

Second, as it is a local server, testing on actual device is not feasible. Only simulator is used in testing.

Third, backend calculation is carried out by CPU which takes considerable time to return the results (approximately 20s per query image with the fastest combination of features). Considering the essence of mobile applications, this is a significant issue to be addressed in the future.

Finally, as observed during testing, there needs to be a caching scheme for the list information, because loading the images takes quite a long time. In future work, Swift libraries such as SDWebImages can be used to alleviate the phenomenon.

Conclusion

In this project, I have developed a content-based image recognition system, and had both experience on client-side iOS development and server-side network design and training. Although the final matching accuracy is not quite ideal, during the trial and error process I have grabbed a comprehensive understanding of the high-level network architectures, the evaluation metrics, the common approaches of network optimization, as well as other related image representations and more advanced feature fusion techniques. Wrapping the engine into a rather simple mobile application and learning about several online cloud platforms let me see the potential of machine learning and artificial intelligence being implemented in the mobile territory. I believe that portable machine learning would become a promising field and more interesting and entertaining applications could be sighted very soon.

Reference:

[1] Faster R-CNN: Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.

[2] Resnet50: He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[3] SIFT: Lowe, David G. "Object recognition from local scale-invariant features." *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee, 1999.

[4] VLAD: Aggregating local descriptors into a compact image representation, Hervé Jégou, Matthijs Douze, Cordelia Schmid, Patrick Pérez, 13-18 June 201

[5] *Discogs - Database and Marketplace for Music on Vinyl, CD* ...www.bing.com/cr?IG=58CAA01A2D9A4AD0A66B90A24EEE2777&CID=00B6EE964118691D2A79E55940B76880&rd=1&h=DVDq-4bjq82QIwx5ug3m22uqjwE8KqNdNW9b3_UZlPA&v=1&r=https://www.discogs.com/&p=DevEx,5063.1.

[6] “Encyclopaedia Metallum: The Metal Archives.” *Home - Encyclopaedia Metallum: The*

Metal Archives, www.metal-archives.com/.

[7] *YouTube*, YouTube, www.youtube.com/playlist?list=WL.

[8] “Results for Pre Owned Metal Vinyl.” *Pre Owned Metal Vinyl in Music | EBay*, www.ebay.com/sch/Music/11233/i.html?_from=R40&_nkw=pre owned metal vinyl&_pgn=2&_skc=200&rt=nc.

[9] *Explore US Heavy Metal on Discogs*, www.discogs.com/search/?sort=title,asc&style_exact=Heavy Metal&type=master&country_exact=US&page=1.

[10] “Blind Guardian.” *Blind Guardian - Encyclopaedia Metallum: The Metal Archives*, www.metal-archives.com/bands/Blind_Guardian/3.

[11] *#NuclearBlast30th | Facebook*, www.facebook.com/hashtag/nuclearblast30th?source=feed_text&story_id=1609959135729581.

[12] “Open Multiple URLs.” *Google*, Google, chrome.google.com/webstore/detail/open-multiple-urls/oifijhaokejakekmnjmphonojcfkpbbh?hl=en.

[13] *Google Search*, Google, www.google.com.hk/search?q=Bulk Image Downloader&oq=Bulk Image

Downloader&aqs=chrome..69i57j0l5.411j0j4&sourceid=chrome&ie=UTF-8.

[14] “Overview.” *JDownloader.org - Official Homepage*, 20 Mar. 2017, jdownloader.org/.

[15] “Hellock/Icrawler.” *GitHub*, github.com/hellock/icrawler.

[16] “Building powerful image classification models using very little data, The Keras Blogs, The Keras Blog.” *The Keras Blog ATOM*, blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

[17], “Fchollet/Deep-Learning-Models. ImageNet pre-trained weights for Resnet50.” *GitHub,* github.com/fchollet/deep-learning-models/releases.

[18] “Labeling Images for Bounding Box Object Detection and Segmentation.” *RectLabel,* rectlabel.com/.

[19] “Zblesk/Csv-to-Sqlite.” *GitHub,* github.com/zblesk/csv-to-sqlite.

[20] “Jorjasso/VLAD.” *GitHub*, github.com/jorjasso/VLAD.

[21] “Buscando El Norte II: La Tierra De Los Sueños.” *Dünedain - Buscando El Norte II: La Tierra De Los Sueños - Encyclopaedia Metallum: The Metal Archives*, www.metal-archives.com/albums/Dünedain/Buscando_el_Norte_II:_La_tierra_de_los_sueños/258965.

[22] “Dünedain - Buscando El Norte II (La Tierra De Los Sueños).” *Discogs*, 1 Jan. 1970, www.discogs.com/Dünedain-Buscando-El-Norte-II-La-Tierra-De-Los-

Sueños/master/1224625.

[23] Zheng, Liang, Yi Yang, and Qi Tian. "SIFT meets CNN: A decade survey of instance retrieval." *IEEE transactions on pattern analysis and machine intelligence* (2017).

[24] Yan, Ke, et al. "CNN vs. SIFT for image retrieval: alternative or complementary?." *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016.

[25] Pretrained FRCNN weights, “model_frcnn.hdf5.” *Google Drive*, Google, drive.google.com/file/d/0B2XPZBSClSzhdENkRmM1alVsN3c/view.

[26] “Yhenon/Keras-Frcnn.” *GitHub*, 21 Jan. 2018, github.com/yhenon/keras-frcnn.

[27] “Hyperopt Tutorial for Optimizing Neural Networks' Hyperparameters.” *Vooban*, 25 Aug. 2017, vooban.com/en/tips-articles-geek-stuff/hyperopt-tutorial-for-optimizing-neural-networks-hyperparameters/.

[28] “Hyperopt/Hyperopt.” *GitHub*, github.com/hyperopt/hyperopt/wiki/FMin.

[29] “Guillaume-Chevalier/Hyperopt-Keras-CNN-CIFAR-100.” *GitHub*, github.com/guillaume-chevalier/Hyperopt-Keras-CNN-CIFAR-100.

[30] Zulkifli, Hafidz. “Understanding Learning Rates and How It Improves Performance in Deep Learning.” *Towards Data Science*, Towards Data Science, 21 Jan. 2018,

towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

[31] Introduction to learning rate decay, *CS231n Convolutional Neural Networks for Visual Recognition*, cs231n.github.io/neural-networks-3/.

[32] *Optimizers - Keras Documentation*, keras.io/optimizers/#parameters-common-to-all-keras-optimizers.

[33] "Cross Decomposition" *1.8. Cross Decomposition - Scikit-Learn 0.19.1 Documentation*, scikit-learn.org/stable/modules/cross_decomposition.html#cross-decomposition.

[34] "Welcome to Flask" *Welcome to Flask - Flask Documentation (0.12)*, flask.pocoo.org/docs/0.12/.

[35] "Model-View-Controller (MVC) in IOS: A Modern Approach." *Ray Wenderlich*, www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.