UNIVERSITY OF CALIFORNIA

Los Angeles

Pet Finding:

Computer Vision Approaches for

Pet Face Recognition and Verification

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics

by

Yifeng Lan

2022

ABSTRACT OF THE THESIS

Pet Finding: Computer Vision Approaches for

Pet Face Recognition and Verification

by

Yifeng Lan

Master of Applied Statistics

University of California, Los Angeles, 2022

Professor Yingnian Wu, Chair

This thesis aims to implement two computer vision approaches for pet owners to find their lost pets in a more effective manner. We first introduce the product design of this thesis in real life conditions. Then we implemented YOLOv3 for face extraction, BN-Inception embedding with proxy anchor loss model for face recognition, and combinations of different embedding methods and classification models for face verification. For face recognition, when our approach returns top 5 most likely matched pet images to the query image, the face recognition model could find 45.45% of all the matching gallery images. For face verification, by using EfficientNet as image embedding model and SVM with gaussian kernel as the classification model, our approach achieves test AU-ROC scores of 88.89%. Finally, we will conclude this case study and discuss further improvements.

The codes and datasets used and processed in this thesis could be accessed on Github repository: AliceLLLLLan/PetFinding[5].

The dissertation of Yifeng Lan is approved.

Hongquan Xu

Tao Gao

Yingnian Wu, Committee Chair

University of California, Los Angeles

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGES

I am grateful to Yunbai Zhang, Tianjian Cheng, Wen-Lung Hsu, Chen Liu, and Runhan Xu, whom I have had the pleasure to work with during this thesis. I would especially like to thank Dr. Yingnian Wu, the chairman of my committee, Dr. Hongquan Xu, and Dr. Tao Gao. Each of the committee members has provided me with extensive personal and professional guidance and taught me a great deal about both scientific research and life in general. I would also like to express my sincere thanks and gratitude to my parents for supporting me all the time.

CHAPTER 1

Introduction

1.1 Background

Deep learning and machine learning approaches have yielded impressive achievements in human face recognition and verification. There has been far less research into applying machine learning and deep learning to individual animal recognition.

A pet owner's worst fear is losing a beloved companion. There are approximately 78 million dogs and 85.8 million cats living as companion animals in households in the United States alone. Approximately 6.3 million companion animals enter U.S. animal shelters nationwide every year. Of those, approximately 3.1 million are dogs and 3.2 million are cats[1]. Many of these pets are microchipped by their owners to identify the pets if the pets run away, which includes registration in a database.

There are 4 existing methods to find lost pets:

1. Contact local animal shelters and animal control agencies: File a lost pet report with every shelter within a 60-mile radius of the owner's home and visit the nearest shelters daily, if possible.
2. Search the neighborhood: Walk or drive through the owner's neighborhood several times each day.
3. Advertising: Post notices at grocery stores, traffic intersections, pet supply stores, and other locations. Include the pet's sex, age, weight, breed, color, and any special markings.
4. Try the internet: The Internet could also broaden the search for the owner.

All the methods above could be effective when a pet is lost for a short time and hasn't run far away. What if the pet has been lost for more than a day or ran outside of its neighborhood? Then the probability of finding the lost pet by the above methods would be hopeless.

However, with the development of computer vision technology, we could build a pet-finding computer vision application that aims to utilize image data and apply deep learning models to build up a third-party platform for pet identification and recognition and help people find their lost pets.

1.2 Product Design

To design our pet finding application, we first need to understand who our target users are and what exactly our target users want, which is called a design lifecycle. This process contributes to prioritizing user needs while prototyping ideas for the application.

The design lifecycle has four steps that form a feedback loop: need-finding, design alternative, prototyping, and evaluation. This loop helps us learn about our users, and narrow down on design alternatives that serve users best for the task.

1.2.1 Needs Finding

After designing and carrying out 5 user interviews[5], and recording them for later analysis, we conclude that our problem space is focusing on finding lost pets, our target users are pet owners, and the mission of our application is to offer a more efficient way to find missing pets by building computer vision algorithms to verify and identify the missing pets.

Therefore, compared with the traditional pet finding methods that require users to ask neighbors one by one and search over the social media and shelters every day, our applications

will automate these processes by transferring from passively waiting to actively search for possible pairs over the databases.

1.2.2 Prototyping

We name our product as Finding Pet Plus. Figure 1.2.2 displays the logic of our product and we will explain each step one by one.

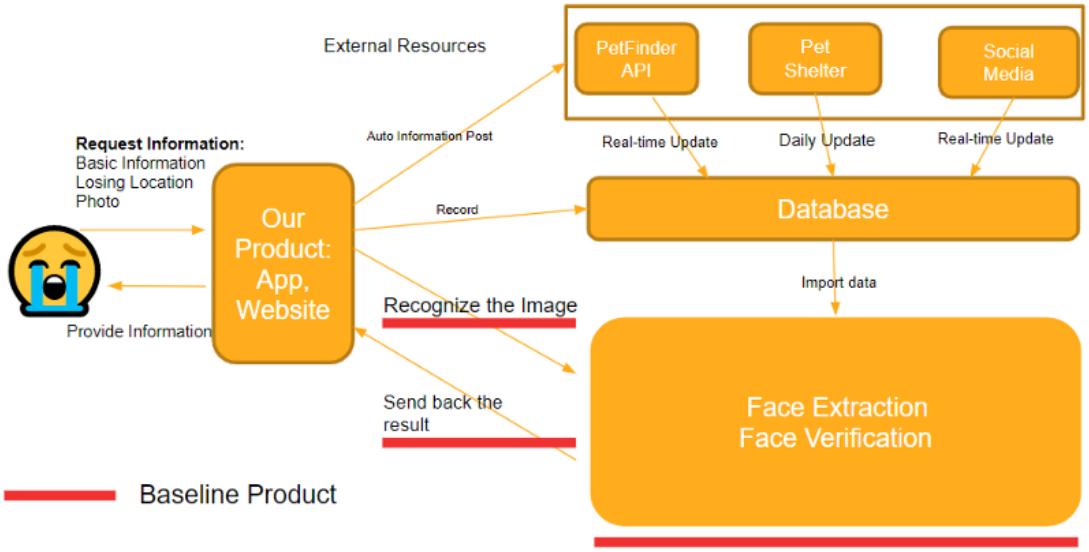


Figure 1.2.2: Product Design Flowchart

First, the application would ask users to input basic information about their missing pets through the app interface, like the breeds, ages, sexes, colors of their missing pets, and locations where they lost their pets. In addition, we would ask users to upload several pictures of their missing pets. In addition, the database of this application will be integrated by the pet face images collected from nationwide shelters databases.

Then the application will do 3 things:

1. Automatically posts this missing pet onto social media.
2. Automatically inputs this missing pet information (pet's demographic feature and pet's images) into our database.

3. Runs pet face recognition algorithm to find a matching pet in the database.

Secondly, to run our algorithm, we plan to integrate our databases from 4 different sources: pet finder API, pet shelter databases, social media posts, and user inputs on the application.

Finally, the application will compare the uploaded image of the lost pet from the user as the target image to our image databases for potential matches through the following two steps:

1. Extracts the pet face from the target image to reduce background noise

2. Compares each pair of the target image and database images to see whether they belong to the same pet, and which pet id it is

If the application finds a match from its database, it represents that the matched image in the database comes from the user's lost pet. Therefore, the user could find the lost pet based on the source of the matched image.

1.3 Problem Statement

This thesis will focus on the third part of the application, which is using computer vision algorithms to achieve pet face extraction and pet face recognition to find a matching pet in the database.

To simulate the database of this application, which ideally will be integrated by the pet face images collected from nationwide shelters databases, we choose to use an open source PetFinder.my Adoption Prediction database[2]. In addition, in this thesis, we constrain our pet finding cases to cats.

All these models are based on convolutional neural networks. We firstly applied YOLOv3 for object detection to extract cat faces from images, then we implemented two

methods to find target pet ID from the database to the query pet image: face recognition and verification. The face verification model is composed by feeding transferred EfficientNet embedded images into the SVM classification model.

The thesis firstly transferred YoloV3 for face extraction to reduce image background noise. Then we implemented two methods to find target pet ID from the database to the query pet image: face recognition and verification.The face verification model is composed by feeding transferred EfficientNet embedded images into the SVM classification model. The face recognition model calculates similarities between image embeddings with Batch-normalization Inception with proxy anchor loss. The face verification model is slow because each image has to be fed into the model to calculate new similarity every time when the query pohto changes. This challenge calls for the face recognition model that transfers images into embeddings in high-dimension space where similar photos have short distances.

As a result of the face recognition and verification, we got 88.89% test AU-ROC score for the face verification model and 45.45% recall@5 for face recognition model. Since the size of the image dataset is large and we would like to train the models by ourselves, we need to use cloud computing and GPU. Therefore, we will use Google Colab as our primary computer environment[5]. The codes and datasets used and processed in this thesis could be accessed on Github repository: AliceLLLLLan/PetFinding[5].

The organization of this thesis is as follows: The second chapter describes the dataset used for training and testing. The third, fourth, and fifth chapters introduce the algorithms and implementation. The sixth chapter makes a conclusion.

CHAPTER2

Dataset

2.1 PetFinder

To simulate the database of this application, which ideally will be integrated by the pet face images collected from nationwide shelters databases, we choose to use an open source PetFinder.my Adoption Prediction database[2].

This database includes two parts: pet images and pet demographic information. The database is over 2.3G and has records of 18,967 pets, 72,776 pieces of images, and contains photos of both cats and dogs. All the images in this database are identified with petID and photo id. The pet demographic information contains petID, breed, color, age, gender, health condition, etc.

2.1.1 Pet Images

In this thesis, we will only use images in the *train_images* folder as our training data and images in the *test_images* folder as our testing data.

In addition, since in this thesis, we design our application to specialize in cat face recognition and verification, we will filter image annotation and only use images with the type of cats instead of dogs.

For pets that have photos, they are named in the format of *PetID-ImageNumber.jpg*. Image 1 is the profile (default) photo set for each pet. For privacy purposes, faces, phone numbers and emails have been masked from images.

For a given pet, the database involves different postures and angles for this pet. Here are some samples of our image dataset.

Figure 2.1.1: Examples of pet images with different postures and angles for one pet id

2.1.2 Image Noise

From Figure 2.1.1, we could see that since most of the pet images were taken during the real life conditions which could introduce various kinds of noise that could potentially influence face recognition and verification algorithms on distinguishing pet faces.

Figure 2.1.2: Images of Noise

For example, in some images, like Figure 2.1.2, there are two or three pets in one image, or there will be obstacles, like cages, in front of cats' faces. In addition, lightness and blurriness could also influence the model performance. Therefore, instead of directly feeding training

images into our face recognition and verification model, we first need to preprocess images with face detection and extraction.

CHAPTER 3

Face Extraction

3.1 YOLOv3

The goal of the face extraction is to reduce image noise and then improve the performance of face verification and recognition by extracting the cat faces from the image dataset.

In order to achieve cat face extraction, we will use a state-of-the-art, real-time object detection system called You only look once (YOLO), which was firstly proposed by J. Redmon, S. Divvala, R. Girshick, and A. Farhad[3] in 2015.

YOLO is a new approach to object detection. Compared with prior work on object detection, which repurposes classifiers to perform detection, YOLO frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities[3]. The boundary boxes are generated by clustering the dimensions of the ground truth boxes from the original dataset to find the most common shapes and sizes[4].

YOLO is a convolutional neural network, which can process input images in a form of structured arrays and then identify patterns between them, like Figure 3.1.1. Therefore, YOLO has the advantage of being much faster than other networks while still maintaining accuracy[4].

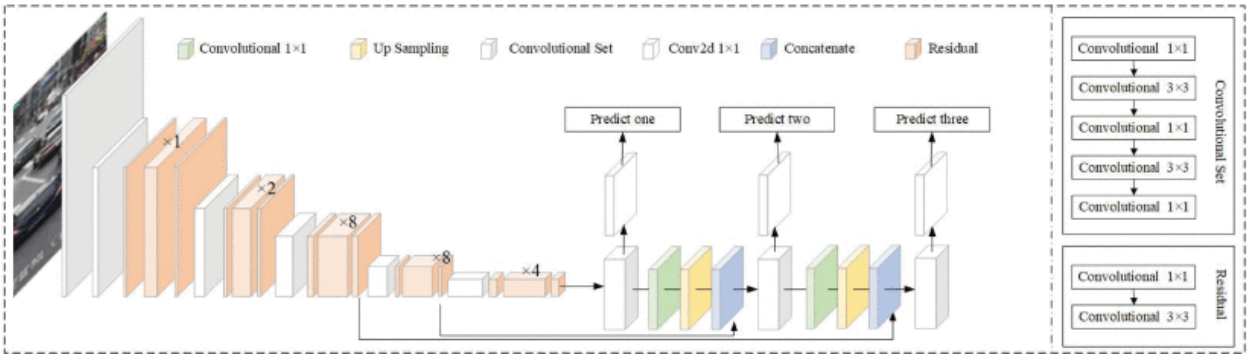


Figure 3.1.1: How YOLOv3 works

In addition, since YOLO looks at the whole image at test time, it uses the global context in the image for its prediction, and "scores" regions based on their similarities to predefined classes. The higher the score is, the more positive detections of whatever class they most closely identify with, representing the confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box[4].

YOLOv3 is an improved version of YOLO and YOLOv2, which were all created by J Redmon and A Farhadi, and are implemented using the Keras or OpenCV deep learning libraries. Its performance on speed, presion, and architecture all exceed its previous versions. YOLOv3 uses Darknet-53as its backbone feature extractor, which is efficient and effective. YOLOv3 is fast and accurate also in terms of mean average precision (mAP) and intersection over union (IOU) values[4]. Therefore, we decided to apply YOLOv3 for cat face extraction as image preprocessing to reduce background noise.

3.2 Implementation

The goal of the face extraction is to improve the performance of face verification by extracting the cat face from images to reduce background noise. By using YOLOv3, we will remove the image backgrounds and put cat faces in the middle of the images.

The coding process of the face extraction is demonstrated on Google Colab GPU platform[5], and the code modified from AntonMu's TrainYourOwnYOLO github repo[6] to build a custom object detector YOLOv3 from Scratch.

The pipeline to build and test our YOLO object detection algorithm has the following steps: image annotation, model training on annotated images, and model inference to detect objects in new images[6].

3.2.1 Image Annotation

In order for our detector to learn to detect objects in images, such as cat faces in pictures for our case, YOLO needs to be fed with labeled training data. Therefore, we need to manually label cat faces for 100 selected images from our training images without replacement as ground truth images[7].

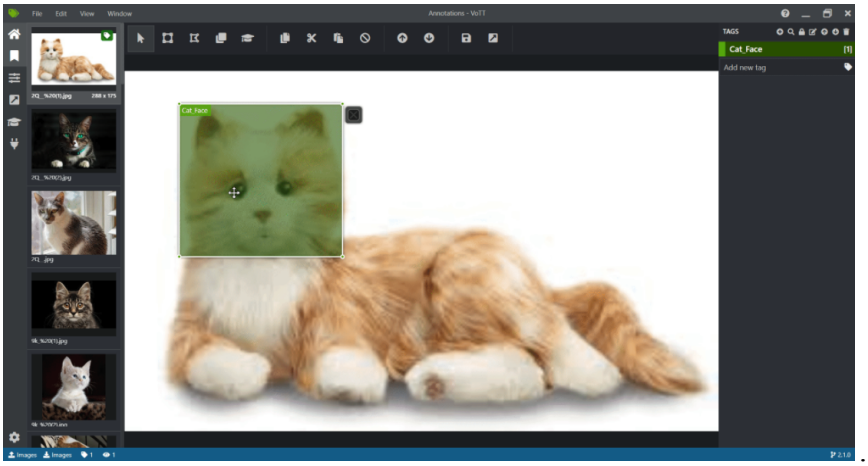


Figure 3.2.1-1: VoTT Object Labeling for Ground Truth Data

image	xmin	ymin	xmax	ymax	label
036af6df3-11.jpg	37.63712917	0	280.971879	218.920536	Cat_Face
036af6df3-12.jpg	147.9218171	95.6381281	256.947234	188.403521	Cat_Face
03a2876c0-1.jpg	87.94723252	26.395973	257.157541	202.403873	Cat_Face
03a2876c0-4.jpg	72.04420028	62.9848538	218.341152	223.49321	Cat_Face
064d06815-3.jpg	44.70973187	60.8702559	146.906761	178.871866	Cat_Face
07af2a6b8-5.jpg	58.60284058	2.61142061	189.670201	129.230501	Cat_Face
10ba7efac-24.jpg	275.2966625	27.1643328	524.758962	256.587727	Cat_Face
1377d2c0e-4.jpg	43.02849437	146.312674	223.679292	338.523677	Cat_Face
163919bd1-1.jpg	180.1513819	4.79630919	366.432567	180.281598	Cat_Face
1de2c7e27-1.jpg	104.3646341	20.2979794	288.072556	213.859761	Cat_Face
1de2c7e27-4.jpg	448.5568603	51.0340483	604.184178	201.448402	Cat_Face
2a631f760-1.jpg	145.6118665	38.7321604	252.75224	136.532651	Cat_Face

Figure 3.2.1-2: Example of VoTT Ground Truth CSV Output

To label images, we used Microsoft's Visual Object Tagging Tool (VoTT). On VoTT, we drew bounding boxes around cat face objects, like Figure 3.2.1-1. Once we have labeled all selected images and export them, we will have a folder including a CSV file called *Annotations-export.csv*, like Figure 3.2.1-2, which contains file names and bounding box

coordinates of the cat face objects. This is how we generate our ground truth data, which will be used for later image inference and evaluation.

3.2.2 Train YOLOv3 Model

To train our YOLOv3 model, we first got started by downloading the pre-trained dark-net weights and converting them to the keras format. The weights are pre-trained on the ImageNet 1000 dataset[8] and thus work well for object detection tasks that are very similar to the types of images.

In order to implement transfer learning, we froze the first 249 layers of the total 252 layers. We trained the last 3 layers with single class detection in YOLOv3 with 51 epochs on 90 samples and validated on 10 samples with batch size 32. After that, we unfreeze all layers, and trained another 51 epochs on 90 samples and validated on 10 samples with batch size 4 in order to fine tune the model. This method of progressively freezing layers is a way to speed up the training efficiency. Freezing a layer prevents its weights from being modified, and this technique is often used in transfer learning, where the pretrained model is frozen. Once we freeze the weights of a layer, the backward pass to that layer can be completely avoided, resulting in a significant speed boost[9].

The result shows that the training process stopped early at epoch 86 with learning rate of 0.0, training loss of 14.93555, and validation loss of 15.774, shown in Figure 3.2.2.

```
wandb: Waiting for W&B process to finish, PID 779... (success).
wandb:
wandb: Run history:
wandb:      epoch
wandb:       loss
wandb:         lr
wandb:   val_loss
wandb:
wandb: Run summary:
wandb:      best_epoch 76
wandb:   best_val_loss 15.48536
wandb:           epoch 86
wandb:            loss 14.93555
wandb:              lr 0.0
wandb:        val_loss 15.774
```

Figure 3.2.2: YOLOv3 Training result

3.2.3 YOLOv3 Inference

After we trained our transfer learnt YOLOv3 detector on our cat face training image dataset, we would like to apply this freshly trained YOLOv3 object detector to extract cat faces from our cat face image database. The outputs include the original images with bounding boxes and confidence scores as well as a file called *Detection_Results.csv* containing the image file paths and the bounding box coordinates, shown in Figure 3.2.3-1 and Figure 3.2.3-2.

	image	image_path	xmin	ymin	xmax	ymax	label	confidence	x_size	y_size
0	cf8e714b5-20.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	275	0	385	171	0	0.658163	640	480
1	c9cc0915c-2.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	49	72	220	222	0	0.954133	400	300
2	8c2aae63b-1.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	82	34	222	194	0	0.990522	267	400
3	fa773690e-3.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	134	76	288	245	0	0.917266	360	480
4	d9362f7f1-2.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	144	6	295	161	0	0.959801	398	400
...	...	...	...	...	...	...	...	...	...	...
36390	25b3328fd-1.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	234	335	483	470	0	0.304951	640	640
36391	25b3328fd-1.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	243	210	474	520	0	0.827460	640	640
36392	0907a14f3-4.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	121	117	199	213	0	0.757232	299	400
36393	8afeab374-4.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	97	23	246	168	0	0.937938	300	400
36394	62f87187d-1.jpg	/content/sample_data/pet_find/TrainYourOwnYOLO...	0	0	217	127	0	0.841127	225	127

36395 rows × 10 columns

Figure 3.2.3-1: Output *Detection_Results.csv* file example

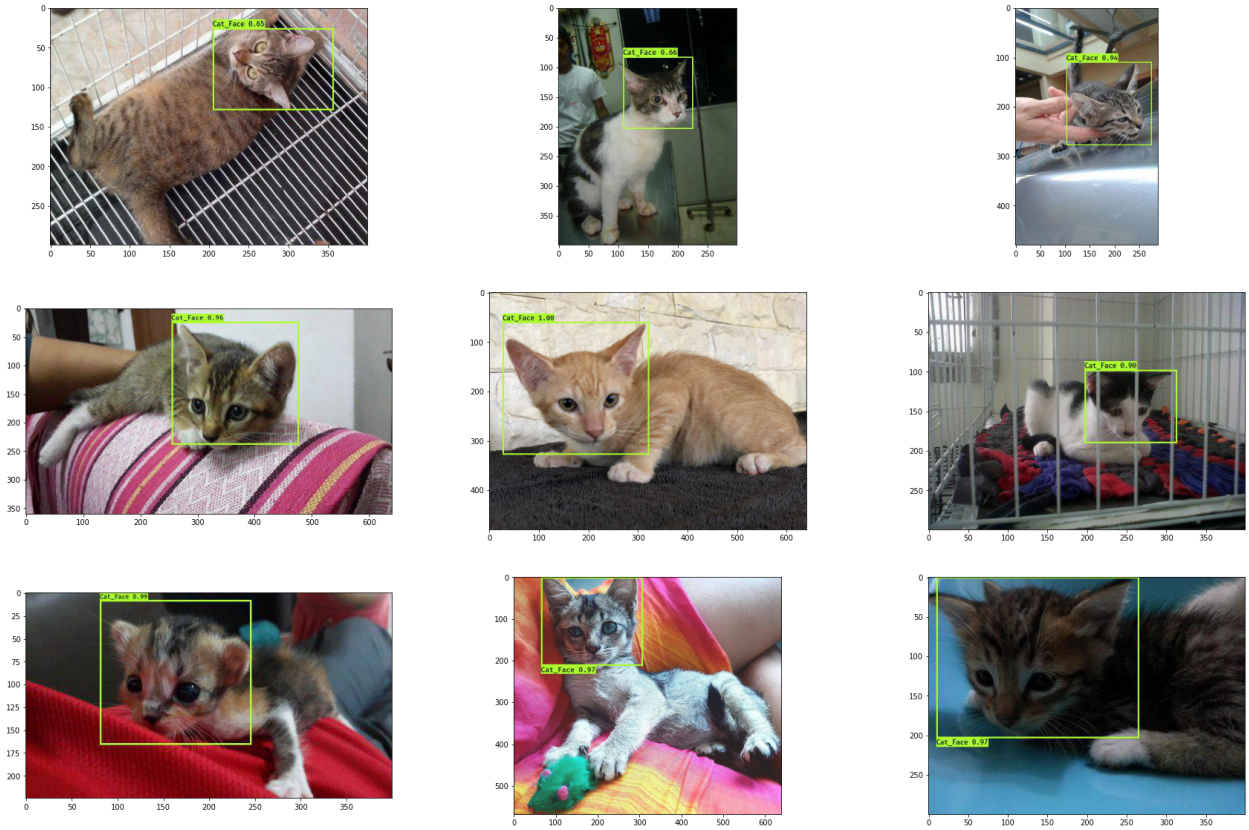


Figure 3.2.3-2: Output images with bounded boxes and confidence score

After this, we cropped the bounded boxes from the original images. In addition, to ensure the quality of the images used for the face verification model, we filter the cropped images with the following criteria:

1. We drop all the pictures that contain more than one face.
2. We filter out the pictures with a confidence level below 0.6.
3. We make sure each pet ID in the dataset has at least three shots.

After filtering on criteria, we saved the cropped images of cat faces for later face recognition and verification, shown in Figure 3.2.3-3

Figure 3.2.3-3: Cropped cat face from original image

3.3 Evaluation

We evaluated our trained YOLOv3 object detector by metrics of the object detection problem: Intersection over Union (IoU). The code to implement this evaluation is based on rafaelpadilla's Object_Detection_Metrics repo[10].

To evaluate, the test data or ground truth instance is in the form of (xmin, ymin, xmax, ymax) bounding box coordinates selected manually by VOTT from Section 3.2.1 Image Annotation. And the predicted instance is in the form of (xmin, ymin, xmax, ymax) bounding box coordinates from outputs of trained YOLOv3 detector generated during Section 3.2.3 YOLOv3 Inference.

3.3.1 Intersection Over Union (IOU)

Intersection Over Union (IOU) is a measure based on Jaccard Index that evaluates the overlap between two bounding boxes. It requires a ground truth bounding box and a predicted bounding box. IOU is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them. Figure 3.3.1-1 below illustrates the IOU between a ground truth bounding box (in green) and a detected bounding box (in red):

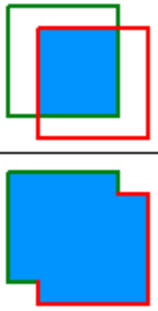
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$ 

Figure 3.3.1-1: IOU Intersections

By applying the IOU we can tell if a detection is valid (True Positive) or not (False Positive)[10]. And some basic concepts about evaluation metrics are as follows[10]:

1. True Positive (TP): A correct detection, representing detection with IOU ≥ threshold.
2. False Positive (FP): A wrong detection, representing detection with IOU < threshold.
3. False Negative (FN): A ground truth exists but the detector failed to detect it.
4. True Negative (TN): Does not apply to this case.
5. Threshold: depending on the metric, it is usually set to 50%, 75% or 95%.
6. Precision: the ability of a model to identify only the relevant objects. It is the percentage of true positive predictions (Figure 3.3.1-2)[10].
7. Recall: the ability of a model to find all the relevant cases (all ground truth bounding boxes). It is the percentage of true positives detected among all relevant ground truths (Figure 3.3.1-3)[10].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}}$$

Figure 3.3.1-2: Precision Formula

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}}$$

Figure 3.3.1-3: Recall Formula

3.3.2 Average Precision

The Precision x Recall curve is a way to evaluate the performance of an object detector as the confidence is changed by plotting a curve for each object class. An object detector of a particular class is considered good if its precision stays high as recall increases, which means that if you vary the confidence threshold, the precision and recall will still be high[10].

Average precision (AP) compare the performance of object detectors is to calculate the area under the curve (AUC) of the Precision x Recall curve. In practice, AP is the precision averaged across all recall values between 0 and 1[10].

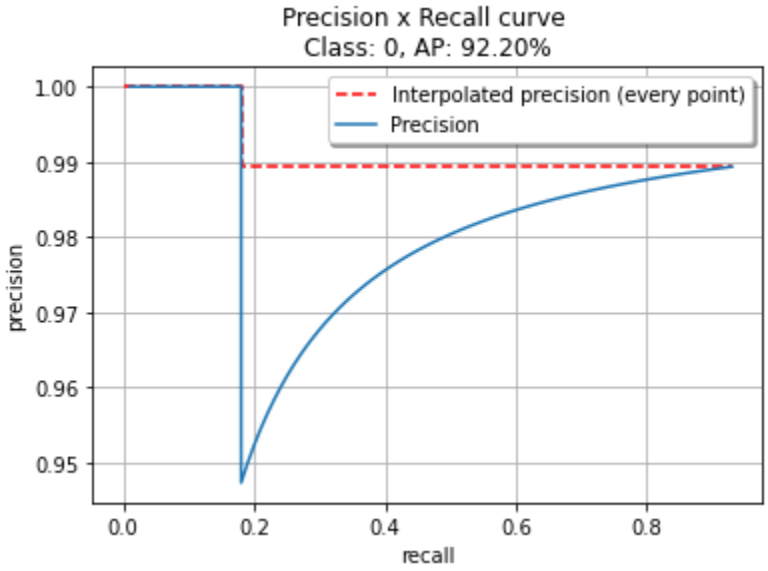


Figure 3.3.2: Average Precision Values Per Class

IoU affects how we define TP and FP. If the average precision is calculated with IoU at 0.5, then it's average precision@0.5. In this project, we use average precision@0.5. We could also visually have the interpolated precision points by looking at the recalls starting from the highest (1.0) to 0 (looking at Figure 3.3.2 from right to left) and, as we decrease the recall, we collect the precision values that are the highest as shown in the image below, and achieved average precision@0.5 being around 92.20%.

CHAPTER 4

Face Recognition

4.1 Proxy Anchor Loss

The way an algorithm learns how to distinguish different cat faces is a way to learn good embeddings for various classes. Faces from the same pet should be close to each other when projected onto the higher dimension space and form a well-separated cluster. The key to a successful face recognition model is the right loss function. The goal of loss functions is to minimize the distance of two embeddings with the same label and to maximize embeddings with different labels in the embedding space.

Existing metric learning losses can be categorized into two classes: pair-based and proxy-based losses. The former class can leverage fine-grained semantic relations between data points, but slows convergence in general due to its high training complexity. In contrast, the latter class enables fast and reliable convergence, but cannot consider the rich data to-data relations[11].

Proxy anchor loss for deep metric learning was firstly proposed by S. Kim, D. Kim, M. Cho, and S. Kwak[11] on CVPR 2020. The paper presents a new proxy-based loss that takes advantages of both pair- and proxy-based methods and overcomes their limitations. By using proxies, proxy anchor loss boosts the speed of convergence and is robust against noisy labels and outliers. At the same time, it allows embedding vectors of data to interact with each other through its gradients to exploit data-to-data relations[11].

In this paper, 5 different loss functions were trained and compared accuracy in Recall@1 versus training time on the Cars196[12] dataset: triplet loss, proxy NCA loss, contrastive loss,

multi-similarity loss, and proxy anchor loss. All methods were trained with a batch size of 150 on a single Titan Xp GPU. Proxy anchor loss enables it to achieve the highest accuracy, and converge faster than the baselines in terms of both the number of epochs and the actual training time[11], shown in Figure 4.1.

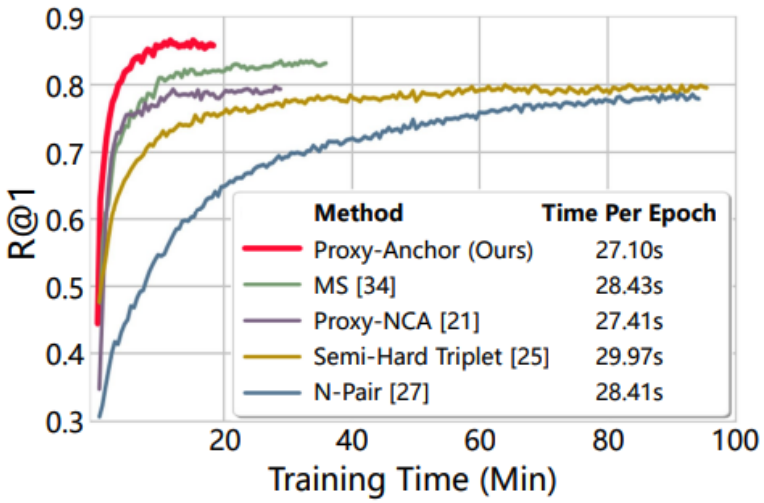


Figure 4.1: Accuracy of Recall@1 versus training time on the Cars-196[11]

4.2 Batch Normal Inception

The Inception network was an important milestone in the development of convolutional neural network classifiers. It was introduced as GoogLeNet in (Szegedy et al. 2015a), here named Inception-v1. The Inception network is heavily engineered so that it has great improvement on speed and accuracy. Later the Inception architecture was refined in various ways, first by the introduction of batch normalization (Ioffe and Szegedy 2015) (BN-Inception/Inception-v2). Later by additional factorization ideas in the third iteration (Szegedy et al. 2015b) which will be referred to as Inception-v3[13].

Batch Normalization (BN) is used for normalizing the value distribution before going into the next layer. With BN, higher accuracy and faster training speed can be achieved. In

addition, BN can reduce the dependence of gradients on the scale of the parameters or their initial values. As a result, higher learning rate can be used, and the need for dropout can be reduced[13].

4.3 Implementation

The code of face recognition is implemented on Google Colab GPU[5]. There are a total of 14067 cat images that splitted into train set and test set. The train set has 9846 cat images, and the test set has 4221 cat images. For each unique pet id in the test set, we randomly drew one image of the pet id and saved it into the query set, and saved the rest of the images for that pet id to the gallery set. Therefore, for our model, we have 3 types of input images: anchor images which is "the control group" image, positive images (with same pet id) which share the same label with the anchor image, and negative images (with different pet id) which have different labels with the anchor image.

We embed images into higher dimension space by a pre-trained BN Inception network with embedding size of 256, which turns each image in a form of image feature vector with length of 256.

Given a query image of a target cat, we calculate the similarity between the query image and the gallery images pairwisely, we calculate the dot product of these pairs of image feature vectors.

Since there are three observations. First, it is linear in both image feature vectors. Second, the dot product of orthogonal vectors is zero. Third, the dot product of a vector with itself equals the square of its magnitude. Therefore, suppose we have two image vectors, x and y. To see the geometric interpretation of their dot product, if we assume that both x and y have a magnitude of

one, then the dot product equals the scaling factor of y, which also equals the cosine of the angle between x and y[14], shown in Figure 4.3-1.

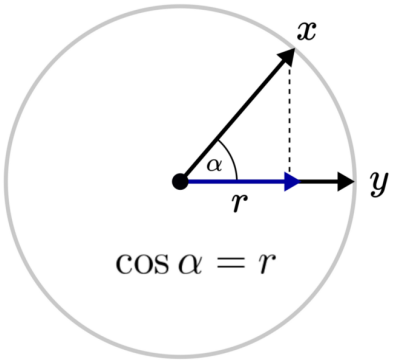


Figure 4.3-1: Geometric Interpretation of Dot Product

After calculating the pairwise similarities between the query image and the gallery images, we sorted the similarity score and returned the top n gallery images as our searching result for this query image from our gallery database, where n could be customized to 5/50/100, shown in Figure 4.3-2.

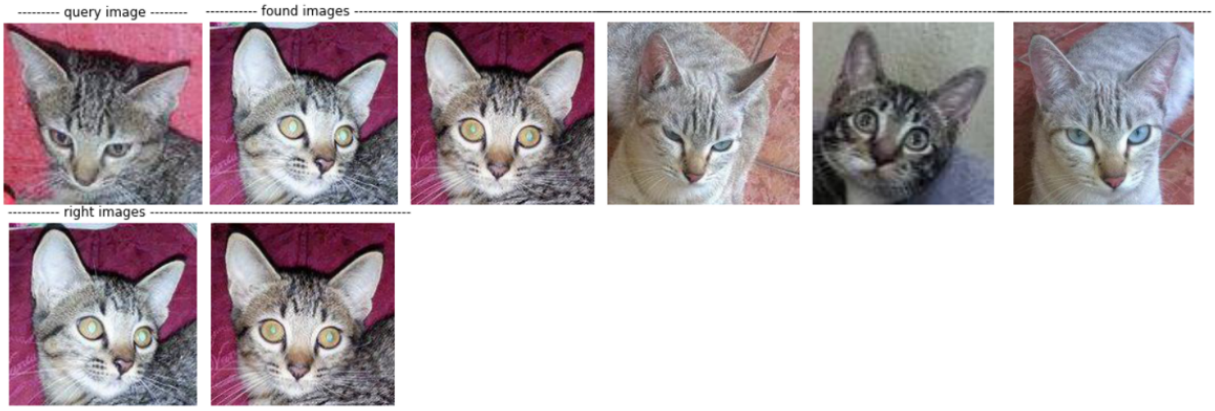


Figure 4.3-2:Example of Searching Result of Face Recognition

From this example output of our face recognition model, we could see that the face recognition model successfully recognizes the right images and ranks them on the top of the outputs.

4.4 Evaluation

To reflect real user experience while evaluating the face recognition model, we choose Recall@k as our metrics: the average positive rate for top k returned gallery images. As long as there is one photo that matches the user's pet photo, we mark it as a positive case, shown as Figure 4.4.

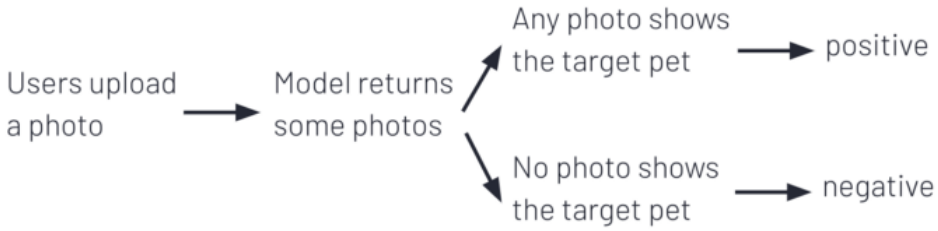


Figure 4.4: Flowchart of Recall@k evaluation

We calculate Recall@k and Precision@k for k equals to 5, 50, and 100. Recall@k represents among all the gallery images that have the same pet id as the query image, what is the percentage of them that actually have been found by the model and returned as the top k selected found images. Precision@k represents that among all the k selected found images, what is the percentage of them that actually have the same pet id as the query image (true positive).

	Recall@5	**Recall@50**	**Recall@100**
Test set	0.4545	1.0	1.0
	Precision@5	**Precision@50**	**Precision@100**
	0.13939	0.0406	0.02181

Table 4.4: Evaluation result of face recognition

From Table 4.4, we could see that even when at k equals to 5, the face recognition is already powerful enough to successfully find 45.45% of all the matching gallery images. And when k increases to 50, the model could already catch all the matching gallery images. For precision, we could see that the scores reduce as the k increases. It is because there are at most 5

matching gallery images for each pet id, therefore, as k increases above 50, the numerator of the Precision@k remains the same while the denominator of the Precision@k keeps increasing, leading the value of Precision@k to decrease.

CHAPTER 5

Face Verification

5.1 Image Embedding

The design of our face verification model is to feed embedded image vectors into a machine learning classification model as a binary classification approach.

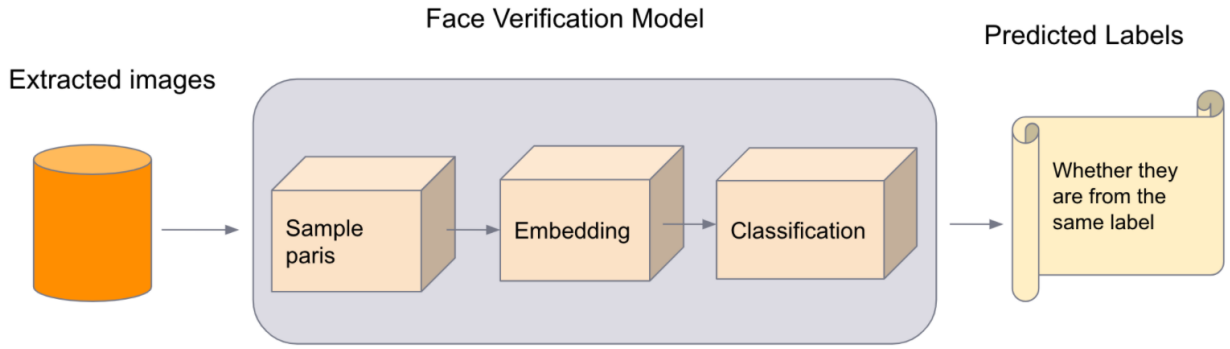


Figure 5.1-1: Workflow of Binary Classification Approach

In the workflow of this binary classification approach, we firstly embedded the extracted face images, then sampled image pairs, and fed them into classification models to see whether the pair of images comes from the same pet.

For image embedding, it is a kind of feature engineering technique to extract and learn important features of images. We use convolutional neural network-based transfer learning models. We cut out final activation and convolutional layers, and then use the output as image features. We applied transfer learning on EfficientNet, VGG16, and Resnet, then fed the embedded matrix into a classification model, e.g. logistic regression model, XGBoost, Random Forest and SVM, to compare their performance on this task. Then to determine whether these two random images come from the same pet.

5.1.1 VGG

VGG16 is a simple and widely used Convolutional Neural Network (CNN) Architecture, and its architecture was developed and introduced by K. Simonyan and A. Zisserman in 2014[18]. "VGG" is the abbreviation for Visual Geometry Group, and "16" implies that this architecture has 16 layers[19].

VGG16 made improvements over AlexNet architecture by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3 × 3 kernel-sized filters one after another. VGG16 is used in many deep learning image classification techniques and is popular due to its ease of implementation. VGG16 is extensively used in learning applications due to the advantage that it has[19].

5.1.2 Resnet

There is a common trend in the research community that our network architecture needs to go deeper. AlexNet, the state-of-the-art CNN architecture, is going deeper with only 5 convolutional layers, while the VGG network and Inception_v1 had 19 and 22 layers respectively. However, because of the notorious vanishing gradient problem. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly[20].

The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers to deal with the vanishing gradient issue, as shown in Figure 5.1.2[20].

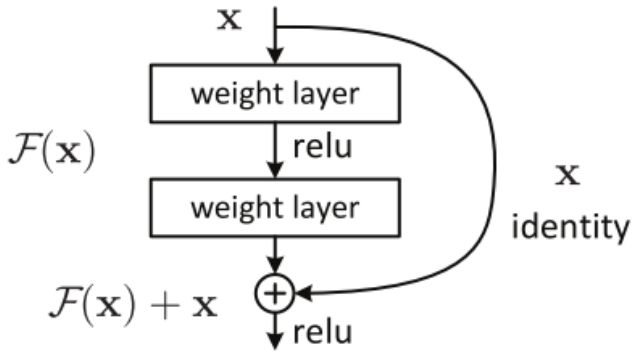


Figure 5.1.2: Identity Shortcut Connection of ResNet

Typical ResNet models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between. Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through[21].

5.1.3 EfficientNet

The conventional practice for model scaling is to arbitrarily increase the CNN depth or width, or to use larger input image resolution for training and evaluation. While these methods do improve accuracy, they usually require tedious manual tuning, and still often yield suboptimal performance. Therefore, it is necessary to find a more principled method to scale up a CNN to obtain better accuracy and efficiency.[16]

In M. Tan and Q.V. Le's ICML 2019 paper, they proposed a novel model scaling method that uses a simple yet highly effective compound coefficient to scale up CNNs in a more structured manner[17]. Unlike conventional approaches that arbitrarily scale network dimensions, such as width, depth and resolution, their method uniformly scales each dimension with a fixed set of scaling coefficients. Powered by this novel scaling method and recent progress on

AutoML, they have developed a family of models, called EfficientNets, which superpass state-of-the-art accuracy with up to 10x better efficiency[17].

5.2 Classification Model

5.2.1 Logistic Regression

Logistic regression, despite its name, is an efficient binary and linear classification model rather than a regression model. It is robust to achieve good performance with linearly separable classes. Logistic regression is a process of modeling the probability of a discrete outcome useing a loss function referred to as “maximum likelihood estimation (MLE)” which is a conditional probability. If the probability is greater than 0.5 or a customized threshold, the predictions will be classified as class 0. Otherwise, class 1 will be assigned. It is important to note that as illustrated in Figure 5.3.1[22], logistic function ranges between 0 and 1 while logit function can be any real number from minus infinity to positive infinity[22].

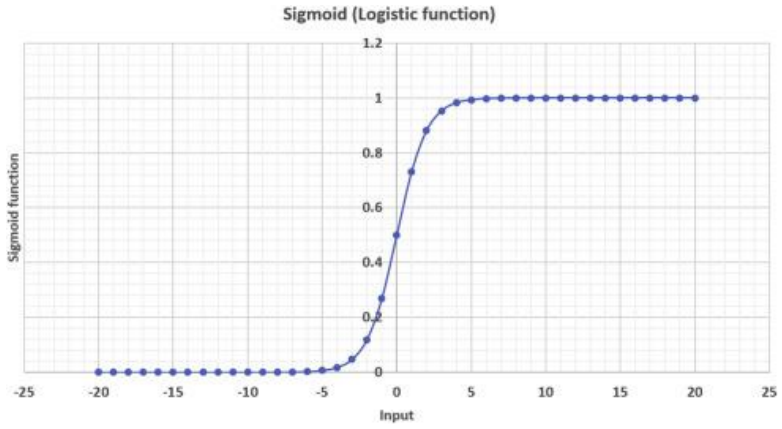


Figure 5.2.1: Logistic regression applied to a range of −20 to 20

5.2.2 SVM

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space where N refers to the number of features that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen, and our objective is to find a plane that has the maximum margin.

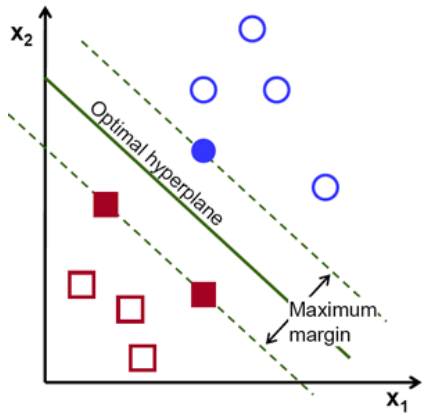


Figure 5.3.2: SVM finds a plane that has the maximum margin[23]

The advantages of support vector machines are that first it is efficient in higher dimensional spaces. Second, it is effective even under the curse of dimensionality where the number of dimensions is greater than the number of samples.

5.2.3 Random Forest

Random forests or random decision forests is an ensemble learning method for classification, regression that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Compared with boosting machines, as a bagging machine, random forests ensemble independent trees parallely. For each tree estimator, the sample and features are sampled with replacement from the training set to reduce variance, leading the model to be less likely to overfit and robust to noise.

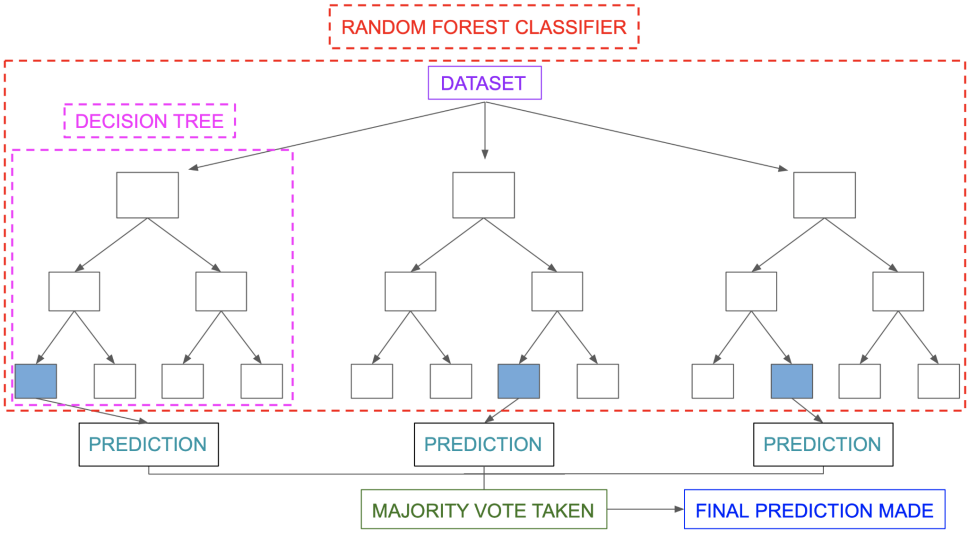


Figure 5.3.3: Architecture of Random Forest[24]

A random forest produces good predictions that can be understood easily since it doesn't contain black box process. It can handle large datasets efficiently while providing a higher level of accuracy in predicting outcomes over the decision tree algorithm.

5.2.4 XGBoost

XGBoost stands for Extreme Gradient Boosting. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves data science problems in a fast and accurate way.

The advantage of XGBoost is that first it is a battle-tested algorithm that wins many data science and machine learning challenges, and widely used in production by multiple companies. In addition, it has great performance with a well-optimized backend system for the best performance with limited resources.

5.3 Implementation

By using 14067 cat face images extracted from Chapter 3 by YOLOv3, we first need to create image sample pairs to train and test.

After the train-test splitting the cat face images by pet id, we have a train set of 2128 pet id and test set of 913 pet id. Each pet id is composite with multiple images of that cat since the photos may be taken from different angles. Since if we splitted by pet images instead of pet ids, some pets in the training model would have seen the pets in the test set, thereby resulting in an information leakage problem. Then within the train set and test set, we created sample pairs respectively.

The principles of sampling image pairs are: for both the train set and the test set, we have two variables:

- k: Number of samples we want to take in total.
- ratio (float): class_weight (0,1] or the ratio between images pairs from the same pet id and images pairs from the different pet id. For example, ratio = 1/3 = 0.33 if we want the ratio between same id pairs and different id pairs to be 1 and 3. To control the ratio of image pairs that are from the same pet and from different pets so that we could avoid imbalanced data problems, we found 1:1 is best for the classification model.

For images from the same pet, we paired up all the pet id combinations, then we shuffled all pairs and selected top k/2 pairs.

For images from the different pets, we randomly selected two indexes. And given these two indexes, we got the random pair. If this pair comes from the same pet id or this pair is a duplicate, then we will resample a pair. We will keep sampling this process until we collect k/2 pairs.

By following this logic, we sampled pairs from the train set with ratio of 1:1 and k of 9747 and sampled pairs from the test set with ratio of 1:1 and k of 4177. Then we continue to embed these image pairs.

Image embedding is a kind of feature engineering technique. We use previously introduced network-based transfer learning models in Section 5.3 to embed, cut out final activation layers and save the output as image features. The advantages of using transfer learning models include: they capture information of neighborhood pixels so that it can learn the features better, they are previously trained on large datasets which return better accuracy and performance, and they no need to retrain by freezing parameters which is friendly to limited computing power.

For each pair of images, after image embedding and concatenating them, we appended all the image embedding pairs and fed them into principal component analysis (PCA) to capture 90% of the variance so that we could further eliminate the influence of noise. PCA is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional subspace. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation[15].

After the noise of the images was reduced by PCA, we fed preprocessed image pairs into classification models to train and predict the test data.

5.4 Evaluation

After running the combinations of classification methods (Logistic Regression, XGBoost, Random Forest, SVM) and embedding methods above (VGG16, ResNet, EfficientNet), we could see that:

Among all the classification models shown in Table 5.5-1 and Table 5.5-2, Random Forest with the number of trees in the forest equals to 200 is too powerful to this train set. Since for Random Forest combined with all three embedding methods, the AU-ROC scores of the train set are all equal to 1.0000 while there are large gaps compared to the results of test sets (0.8063, 0.7979, 0.8464). It is a typical sign of the model being overfitting on the train set. Therefore, the Random Forest model needs further parameter tuning to reduce overfitting.

The results of XGBoost and SVM as classification are both outstanding. There are also small gaps between the results of training and test datasets but the gap comes from our train-test set splitting rule. Overall speaking, the best result on both training and test data is using EfficientNet as image embedding model and SVM with gaussian kernel as the classification model, where training and testing AU-ROC scores equals 0.9845 and 0.8889 respectively.

	Logistic	SVM	Random Forest	XGBoost
VGG16	0.6331	0.9818	1.0000	0.8863
ResNet	0.6821	0.9907	1.0000	0.9066
EfficientNet	0.6613	0.9845	1.0000	0.9087

Table 5.5-1: AU-ROC Scores on Train Data

	Logistic	SVM	Random Forest	XGBoost
VGG16	0.5090	0.8597	0.8063	0.8152
ResNet	0.5015	0.8741	0.7979	0.8427
EfficientNet	0.4927	**0.8889**	0.8464	0.8598

Table 5.5-2: AU-ROC Scores on Test Data

After taking a deeper look at classification results, we could summarize several factors that might affect the result confidence: different facial expressions, variation of background lights, front face not included, and blurred images.

Factors	Solutions
One image containing multiple pets	Removing all images with more than one face during the face extraction step.
Various face expression	Increasing the size of pairs for models to learn different facial expressions.
Low quality images	Removing unconfident images during the face extraction step.
Blurred images	Image augmentation

Table 5.5-3: Result Analysis

CHAPTER 6

Conclusion and Future Work

In this thesis, in order to computer vision approach for pet owners to find their lost pets in a more effective manner, we implemented YOLOv3 for face extraction, BN-Inception embedding with proxy anchor loss model for face recognition, and combinations of different embedding methods and classification models for face verification. For face recognition, when our approach returns top 5 most likely matched pet images to the query image, the face recognition model could find 45.45% of all the matching gallery images. For face verification, by using EfficientNet as image embedding model and SVM with gaussian kernel as the classification model, our approach achieves test AU-ROC scores of 0.8889.

Face Extraction	Reduce noise from images to boost the performance of later approaches	
	Face Verification	**Face Recognition**
Motivation	One to one face comparison	One to many face comparison
Model	EfficientNet and SVM	BN-Inception with proxy anchor loss
Advantages	Easy to interpret and implement	Converge fast; Good performance on low quality images
Disadvantages	Take time to train; Sensitive to large dataset and corner cases	Hard to interpret; Need stronger computational power
Use Case	Smaller size of images and users	Larger size of images and users; Increasing amount of low quality images

Table 6.1: Summary of approaches

In the future work, we would like to further tuning the classification models in face verification approach to reduce overfitting. In addition, in order to possibly launch this product, we shall implement a prototype website UI to further productize it. What's more, in order to actually build up a database of lost pet images, we plan to start from a local region and collaborate with local shelters.

REFERENCE

[1] Pet Statistics. Retrieved 14 March 2022, from https://www.aspca.org/helping-people-pets/shelter-intake-and-surrender/pet-statistics

[2] PetFinder.my Adoption Prediction | Kaggle. (2019). Retrieved 14 March 2022, from https://www.kaggle.com/c/petfinder-adoption-prediction/data

[3] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[4] Meel, V. YOLOv3: Real-Time Object Detection Algorithm (What's New?). Retrieved 14 March 2022, from https://viso.ai/deep-learning/yolov3-overview/

[5] AliceLLLLLan/PetFinding. (2021). GitHub. Retrieved from https://github.com/AliceLLLLLan/PetFinding

[6] AntonMu/TrainYourOwnYOLO. (2021). GitHub. Retrieved from https://github.com/AntonMu/TrainYourOwnYOLO

[7] Muehlemann, A. (2019, October 4). How to train your own YOLOV3 detector from scratch. Retrieved March 14, 2022, from https://blog.insightdatascience.com/how-to-train-your-own-yolov3-detector-from-scratch-224d10e55de2

[8] Imagenet Large Scale Visual Recognition Challenge 2015 (ILSVRC2015). (n.d.). Retrieved March 14, 2022, from https://image-net.org/challenges/LSVRC/2015/index

[9] Gupta, H. (2017, August 3). Train your deep learning faster: FreezeOut. Retrieved March 14, 2022, from https://www.kdnuggets.com/2017/08/train-deep-learning-faster-freezeout.html

[10] rafaelpadilla/Object-Detection-Metrics. (2021). GitHub. Retrieved from https://github.com/rafaelpadilla/Object-Detection-Metrics

[11] Kim, S., Kim, D., Cho, M., & Kwak, S. (2020). Proxy anchor loss for Deep Metric Learning. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr42600.2020.00330

[12] Krause, J., Stark, M., Deng, J., & Fei-Fei, L. (2013). 3D object representations for fine-grained categorization, pages 554–561. 2013 IEEE International Conference on Computer Vision Workshops. doi:10.1109/iccvw.2013.77

[13] Tsang, S. (2018, September 9). Review: Batch normalization (Inception-V2 / BN-inception) -the 2nd to surpass human-level... Retrieved March 14, 2022, from https://sh-tsang.medium.com/review-batch-normalization-inception-v2-bn-inception-the-2nd-to-surpass-human-level-18e2d0f56651

[14] Danka, T. (2021, April 21). How the dot product measures similarity. Retrieved March 14, 2022, from https://towardsdatascience.com/how-the-dot-product-measures-similarity-b3e16e22beda

[15] Sharma, A. (2020). Principal Component Analysis (PCA) in Python Tutorial. Retrieved 1 January 2020, from https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python

[16] Tan, M., & Le, Q. (2019). EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. Retrieved 29 May 2019, from https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[17] Tan, M., & Le, Q.V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946.

[18] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

[19] Perumanoor, T. (2021). What is VGG16?—Introduction to VGG16. Retrieved 23 September 2021, from https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615

[20] Feng, V. (2017). An Overview of ResNet and its Variants. Retrieved 15 July 2017, from https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[21] Wikipedia contributors. (2022, February 21). Residual neural network. In Wikipedia, The Free Encyclopedia. Retrieved 09:47, March 14, 2022, from https://en.wikipedia.org/w/index.php?title=Residual_neural_network&oldid=1073238799

[22] Belyadi, H., & Haghighat, A. (2021). Machine learning guide for oil and gas using Python. Gulf Professional Publishing. doi: 10.1016/c2019-0-03617-5

[23] Gandhi, R. (2018). Support Vector Machine — Introduction to Machine Learning Algorithms. Retrieved 7 January 2018, from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[24] Mbaabu, O. (2020). Introduction to Random Forest in Machine Learning. Retrieved 14 December 2020, from https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/