The Crowd Counting Application System Powered By Cascaded Multi-Task Convolutional Neural Network

Peng Zhao - 899632
pzhao2@student.unimelb.edu.au
Xinrui Lyu - 869719
xlyu2@student.unimelb.edu.au
Shimin Wei - 883861
shiminw1@student.unimelb.edu.au

Supervisor: Prof. Richard Sinnott
COMP90055 Computing Project (Software Development Project)

***Abstract*— There are many challenges in the problem of how to count people based on a camera picture of a crowd, such as occlusion, illumination, varying poses and blurring caused by distance. With the recent advance in object detection technology, especially deep learning based techniques for human face detection and alignment, we can achieve reasonably good performance on this task despite these challenges.**

In this report we describe the implementation of a crowd counting application system. This system consists of an inference engine based on Multitasks Cascade Convolutional Neural Network (MTCNN), which has a cascaded structure enabling it to detect and gradually refine face locations. We tailored this model accommodating to our needs, elaborated the training and evaluation process in details, implemented practical performance improvement and made this system accessible by building iOS Android applications. We also developed a online statistic platform for data gathering and analysis. The backend logic was achieved by Python Flask framework to provide RESTful web service for the applications and the data storage service are provided by Firebase.

I. INTRODUCTION

Deep learning is one of the most popular technologies and research areas in machine learning. Intelligent technology is implemented in computing systems by establishing an artificial neural network with a hierarchical structure. Convolutional Neural Network(CNN) is a typical artificial neural network, which is widely used in many fields, such as image recognition, natural language process, and AlphaGo. CNN has many advantages, such as the high efficiency of feature extraction, the simplicity of data format and the small number of parameters. These advantages prompted us to apply CNN related technology to the research.

This paper is mainly to introduce how to achieve crowd counting using a Multi-tasks Cascaded Convolutional Neural Network. Compared to other models, this model has a good performance with shorter inference time, shallower network structure and smaller size. In order to demonstrate the value and feasibility of the technology and provide a friendly operating environment for users, the application has been achieved on both IOS and Android Platform. Besides, a web platform is developed to visualize the real-time data in the firebase server.

II. BACKGROUND AND RELATED WORK

With the improvement of peoples living conditions, the population continues to show a growth trend. At the same time, the techniques associated to the detection of people have been developed. Crowd counting is an important technique in the field of computer vision. The significance of crowd counting is to use video surveillance, or other camera equipment, to perform counting to the crowd on real-time scenes without disturbing the targets. Crowd Counting can be applied in many fields, users can achieve the safety effects, and analyze economic benefits from this. Pedestrian volume estimation and Event attendance rate are two typical scenarios. On December 31th 2014, a deadly stampede caused by a large number of visitors celebrating the new year occurred in Shanghai, making a great deal of personnel casualty and property losing. If pedestrian volume estimation is implemented in that area, the security can be monitored through estimating the relationship between the current number of people and the threshold of security. In that case, safety officer can be alerted to prevent more people from entering the overloading area. Event attendance rate shows the the popularity of the event. For a sports league, If the number of spectators can be obtained more accurately, the organizer can make more flexible and dynamic adjustments to the ticket plan. Also, in a shopping mall, customer preferences can be acquired by counting the number of people in a queue, or counting the frequency of viewing a single item. In this case, the goods can be reasonably allocated, and the staff can be reasonably assigned as the situation shown by crowd counting.

The history of crowd counting research can be classified into two stages. In the early days, crowd countings are mostly based on detection and regression. In 2006, the paper Reducing the Dimensionality of Data with Neural Networks [1] published by Hinton brought this field into Deep Learning. He creatively used multi-layer neural networks to transform

high-dimensional inputs into low-dimensional outputs and used gradient descent for the tuning of the weights. In 2012, the amazing performance of AlexNet [2] dramatically increases the performance of the CNN network. Since then, the CNN-based methods began to widely used in crowd counting and other topics about computer vision.

Many researches predict number based on the density map. Multi-column Convolutional Neural Network(MCNN) [3] was proposed for accurate estimation of popular density distribution and counting. It is a 3-column structure working through mapping from the photos to the density maps. The size of the faces will be different due to different angles of view, so the different size of kernels are used for the convolution neural networks to adapt to the different size of faces. Also, it implements a convolution layer with a 1*1 filter instead of the fully connected layer. In this case, this model works well for images with uncertain resolution and shooting angles. Besides, researchers in this project developed a huge dataset, in which around 330,000 in nearly 1200 images are labeled accurately.

Contextual Pyramid CNN(CP-CNN) [4] was proposed for estimating the density of the crowd using global and local features. The structure consists of four parts, which are GCE(Global Context Estimator), LCE(Local Context Estimator), DME(Density Map Estimator) and F-CNN(Fusion-CNN). In GCE, the density classes have been set in advance. Then, for the whole image, features can be extracted and implemented into classification. Finally, the images with the same height and with are got as the global context. In LCE, in order to get better density maps, similar to GCE, the extraction and classification are implemented for features to get the local context. In DME, the image is transformed into high-dimensional feature maps. Eventually, the high-dimensional feature maps and both global and local contexts are combined in the F-CNN structure, producing the output image.

CSRNet [5] is proposed for implementing accurate crowd counting in Highly Congested Scenes. The network is classified into two parts. In the front-end, VGG16 is used to extract 2D features because of its strong learning ability, flexible structure and accessibility to the back-end. In the back-end, dilated convolutional layers are deployed to extract deeper information. Gaussian kernel is used to process the head annotation, generating ground truth for error detection. The CSRnet can map the image to the corresponding density map. The size of the input image can be variable because there is no fully connected layer in the structure.

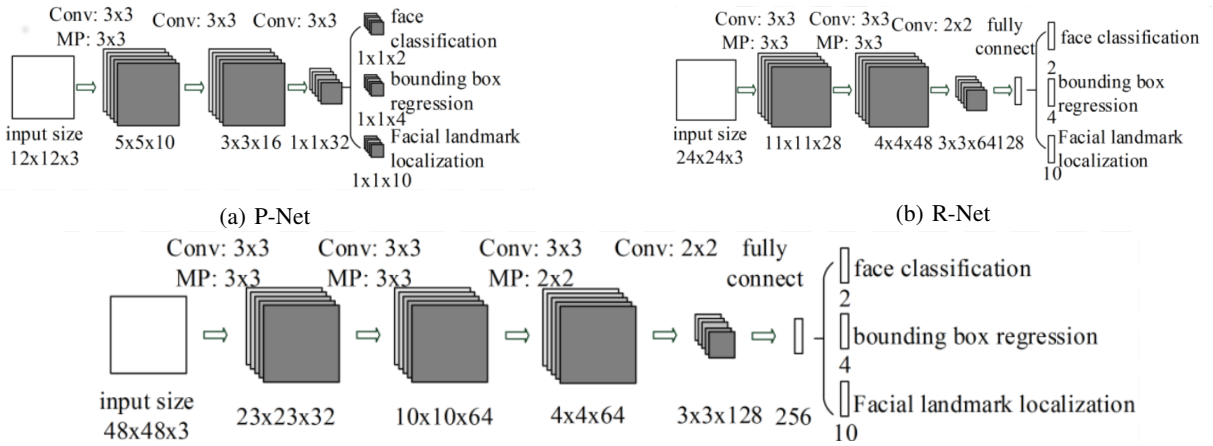
Besides these based on the density maps, there are many other popular research methods, Region-based convolutional neural network(R-CNN) and its variations are some of them. The representative three outcomes of them are R-CNN [6], Fast R-CNN [7], and Faster R-CNN [8]. There are mainly three steps in R-CNN. First, using selective search to identify a specific number of bounding boxes that is likely to be the face, then choosing the Alexnet structure as the primary parameter and tune them during the training process. Finally, calculating scores all over the image, choosing the most suitable candidate and locate the site of a candidate using a trained an SVM classifier. However, there are three disadvantages of R-CNN, complexed steps, high cost, and Iow efficiency[5]. Compared to R-CNN, a new structure fast R-CNN is proposed. In this structure, the image is first processed to generate the feature map using max-pooling layers. Then feature vectors are extracted from the candidates through RoI pooling layers. Softmax probability and offset of bounding boxes can be acquired for each output. In Fast R-CNN, the overlap and redundancy were eliminated, and features can be extracted much faster. In faster R-CNN, the Region Proposal Network(RPN), which is used for choosing bounding box is arranged after the CNN network. In this case, the region proposal can be implemented in the GPU, which dramatically decrease the time cost in the process.

III. DATASET

The dataset we used for our training and evaluation works were WIDER FACE and CelebA datasets [9]. We applied former one on our training works for face locations and later one for landmarks locations. The evaluation work were carried out based on WIDER FACE validation set. The detail of each datasets are elaborated below respectively.

WIDER FACE is a face detection benchmark dataset which consists of 32,203 images and 393703 labeled faces with a variety of different scale, occlusion and pose. It is divided in to 61 event classes and from each class 40%,10% and 50 images are selected for training, validation and test sets separately. Each face bounding box has 6 attributes those are blur, expression, illumination, occlusion, pose and invalid, each attribute uses numeric value to describe its degree or True False statements. In the annotation files, the ground-truth sets of each image are structured in the identical format that starts with the file name and the number of bounding boxes it contains ,followed by a list of bounding boxes information. Each line of bounding box records contains the coordinates of the upper left points on the box, its width as well as the height and value of six attributes it has.

CELEB A, ([10]) as well as Large-scale CelebFaces Attributes dataset, contains more than 200k celebrity images with 40 attribute annotation for each one. It also contains a large amount of various poses and clutter. Furthermore, it totally contains 10,177 identities, 202,599 face images and 5 landmarks for each face. We only used *list_bbox_celeba.csv* and *list_landmarks_align_celeba.csv* out of five annotation files to do the landmark training. The former file contains bounding box information for each image containing upper left point coordinate of bounding box, corresponding width and height. The later one contains the landmark information for each image, which consists of the coordinates of both eyes, nose and mouth for each face.



(a) P-Net

(b) R-Net

(c) O-Net

Fig. 1: Network structures

IV. Approach

The tool we use for this task is Multitasks Cascaded Convolutional Neural Network(MTCNN), proposed in [11]. we will describe its main idea below, and provide a detailed workflow.

Within the MTCNN framework, three tasks are executed synchronously:

- Face classification.
- Face bounding box position prediction.
- Facial landmark position prediction.

It has three sub-networks: **P-Net**(Proposal Network), **R-Net**(Refinement Network), **O-Net**(Output Network). P-Net is a fully convolutional neural network, with final output layer of size $1 \times 1 \times 16$. R-Net and O-Net are convolutional neural networks with last two layers as fully connected layers, both of their output layers are of 16-dimesions. These 16 dimensions correspond to the tasks listed above:

- Dimension 1, 2 for face classification.
- Dimension 3-6 for bounding box position in offset coordinates, as described in Appendix X-C.
- Dimension 7-16 for facial landmark position in offset coordinates of left eye, right eye, nose, left mouth corner, right mouth corner, as described in Appendix X-C.

The structure of these networks is illustrated in figure 1, and we list a detailed description of this structure in the appendix X-D

A. Model Inference

1) Stage 1: P-Net: P-Net is a fully convolutional neural network, thus it can accept input image of any size bigger than its required input size 12×12. For input images not of size 12×12, P-Net produces a feature map, arranged identically with sliding window scanning. When inferencing, input image is scaled down to a image cascade with respect to a given factor, until image is only bigger than some given minimal size. For each image in the cascade, P-Net produces a class feature map (prediction confidence) and a box offset feature map, corresponding to dimensions 1-2 and dimension 3-6 in its output layer. We filter out the index of feature map that has confidence above a given threshold, and use this index to derive the position offset of its corresponding bounding boxes. We next apply *Non-Maximum Suppression (NMS)* algorithm with threshold 0.7 (described in X-B) to remove overlapping redundancy. The remaining boxes are calibrated using box offset feature map. The calibrated boxes are passed to the R-Net for further refinement.

2) Stage 2: R-Net: R-Net need two kinds of data: the original image and output calibrated boxes by P-Net. The calibrated boxes output by P-Net are first converted into squares by extend the smaller dimension, and are used to crop from the original image. These cropped images are resized to 24×24 and fed into R-Net. R-Net will output a confidence score and offset coordinate for each of these images. We keep the input calibrated boxes with high confidence score (above a given threshold), apply NMS algorithm with a larger threshold than in P-Net to further remove overlapping redundancy, and calibrate its position with offset box information. The calibrated boxes are passed to next stage.

3) Stage 3: O-Net: O-Net takes the calibrated boxes output by R-Net. It produce confidence score and offset box in the same process as in R-Net. O-Net also outputs facial landmark. We again filter the input boxes by confidence score, adjust box position with offset coordinate, and apply NMS algorithm to eliminate redundancy. This ends the inference stage, and the final outputs are calibrated boxes and facial landmarks.

B. Model Training

P-Net, R-Net and O-Net have very similar output layer, consisting of three parts. Each part have an associated loss function:

Face classification Let y_i denote the label of image x_i, p_i be the predicted probability, we use the cross-entropy loss:

$$L^{det}(x_i) = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i))) \quad (1)$$

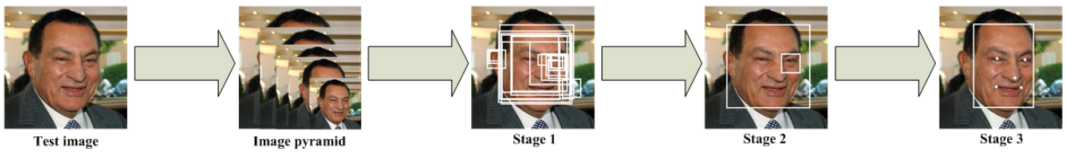


Fig. 2: Inferecne workflow of MTCNN

Bounding box reggression Let $\hat{o}_i$ denote the offset coordinate of image x_i, and o_i be the predicted offset. Since both $\hat{o}_i, o_i \in \mathbb{R}^4$, we use the Euclidean loss:

$$L^{box}(x_i) = \|\hat{o}_i - o_i\|_2^2 \tag{2}$$

Facial landmark localization Let $\hat{m}_i, m_i$ denote the true and predicted facial landmarks in offset coordinates respectively. Since $\hat{m}_i, m_i \in \mathbb{R}^{10}$, we use the Euclidean loss:

$$L^{landmark}(x_i) = \|\hat{m}_i - m_i\|_2^2 \tag{3}$$

We will generate training data for each of these training target, and these loss functions will be triggered on right type of training data. The form of the data is a cropped image, with labels to indicate whether it contains a human face. If the image does not contain human face, only loss 1 will be activated. If it does contain a human face, we will calculate its offset to the nearest ground truth box, and activate loss 1, 2. If the image contains facial landmark annotation, we will use if for landmark localization, and activate loss 1, 3 The overall loss of image x_i is

$$L(x_i, net) = \sum_{j \in \{det, box, landmark\}} \alpha_j(net)\beta_j(x_i)L^j(x_i)$$

Here β is activation indicator, having the property:

$$\beta_j(x_i) = \begin{cases} 1 & \text{if } x_i \text{ belongs to category } j \\ 0 & \text{otherwise} \end{cases}$$

Here α is weight factor that varies according to network. The net argument represents which net we are using.

1) P-Net: The training data of P-Net comprises two parts: bounding boxes and facial landmarks. The bounding boxes are generated by cropping from image the WiderFace training set. Cropped boxes are labeled according to its *Intersection-over-Union* (IoU, described in X-A) with respect to some ground truth box. More specifically:

- Negative boxes are randomly cropped from the image, with square box of a size randomly chosen in the range $(12, \min(\text{width}, \text{height}))$. A box is kept only if its IoU value with any ground truth box is smaller than 0.3;
- Positive boxes are cropped around the ground truth boxes. The size of box is determined by the anchor ground truth box, with a random deviation of 20%. A box is kept only if its IoU with that ground truth box is above 0.65.
- Part boxes are cropped using the same method as in positive boxes, but are kept only if its IoU is between 0.4 and 0.65.
- We calculate the offset coordinates of positive and part boxes.

Positive and Negative boxes are used for face classification training. Positive and part boxes are used for bounding box position regression.

The loss function of P-Net use the parameter set below:

$$\alpha_{\text{det}} = 1, \alpha_{\text{box}} = 0.5, \alpha_{\text{landmark}} = 0.5$$

2) R-Net: We use P-Net to generate training data for R-Net. For a image in WiderFace training set, propose a series of bounding boxes using P-Net as in IV-A.1. Compare these boxes with ground truth boxes and label them according to the same IoU interval as in IV-B.1. For each positive or part example, we find the ground truth box with the highest IoU value.

The loss function of P-Net use the parameter set below:

$$\alpha_{\text{det}} = 1, \alpha_{\text{box}} = 0.5, \alpha_{\text{landmark}} = 0.5$$

3) O-Net: Training data generation for O-Net is basically the same as in R-Net. The only difference is O-Net use both P-Net and O-Net for detection.

The loss function of P-Net use the parameter set below:

$$\alpha_{\text{det}} = 1, \alpha_{\text{box}} = 0.5, \alpha_{\text{landmark}} = 1$$

4) Facial Landmark Data: We use CelebA dataset for facial landmark regression. To generate the actual training data, for each image in this dataset, we crop out the face using the ground truth box, and calculate offset coordinate landmark as in X-C based on the given annotations. We further perform data augmentation, including generating mirrored version, rotating the cropped image clockwisely and anti-clockwisely. All these cropped image will be resized to appropriate size before feeding into P-Net, R-Net and O-Net.

5) Training Process: On completion of loss function definition and training data generation, we start the actual training process.

The data points in the training set have different loss values. Those with low loss does not contribute much to the training process, but demands same amount computation resources for gradient calculation and back-propagation. To exploit the training data more efficiently, we use the **Online Hard Example Mining** ([12]) technique.

In particular, we sort each batch by forward propagation loss and only keep the top 70% of the samples. These samples are regarded as hard examples and we only compute the gradient for them instead of the entire batch.

By ignoring the easy samples that are less helpful to strengthen the detector while training, we can greatly reduce

computation load and improve training result. Experiments show that this strategy yields better performance without manual sample selection [11].

Figure 3 shows the training process of our model. We can see from the figure that the class accuracy increases as we go from P-Net to O-Net, and class loss shows a similar decreasing tendency. The landmark loss also decreases from P-Net to O-Net. However, the bounding box loss of R-Net is higher than P-Net. This is because the input bounding boxes is generated by P-Net instead of randomly cropping from the original image, which might deviate from the ground truth box more than in the case of P-Net.

V. Evaluation

A. Evaluation on Face Classification

We evaluate face classification of the trained model on WiderFace validation set. We treat each image in the dataset that our model successfully detected a face as a successful detection, and assess the model by success rate. Our model achieves a detection rate of 0.8397 on this set, leaving us quite some room to improve. We list the comparison of this result and improved result in Table I.

B. Evaluation on Box Regression

The conventional evaluation benchmark is **mAP** (mean-average precision), described in [13]. The general method is to calculate the **average precision (AP)** for each category of object, and take the mean of these APs. However, our crowd counting problem is different in the way that we have a single type of object to detect, and mAP may not be expressive on its performance. We introduce **Image-wise precision**, which is very intuitive and can be used to assess the model effectively.

Given the image i, together with its ground truth boxes and predicted boxes, we define precision and recall by the following rule.

- Any predicted box that has $IoU \geq 0.5$ with some ground truth box is TP_i **(true positive)**;
- Any ground truth box that has has $IoU < 0.5$ with any predicted box is FN_i **(false negative)**

Denote the number of predicted boxes and ground truth boxes for image i as pb_i and tb_i

$$\text{precision}_i = \frac{TP_i}{pb_i} \tag{4}$$

$$\text{recall}_i = \frac{TP_i}{tb_i} \tag{5}$$

$$\text{precision} = \frac{\sum_i TP_i}{\sum_i pb_i} \tag{6}$$

$$\text{recall} = \frac{\sum_i TP_i}{\sum_i tb_i} \tag{7}$$

The image-wise precision and recall distribution is illustrated in figure 4. We can see from the image that a great potion of the testing images have good detection precision, but the recall rate is less ideal, with a significant number of training image with very low recall rate. This implies

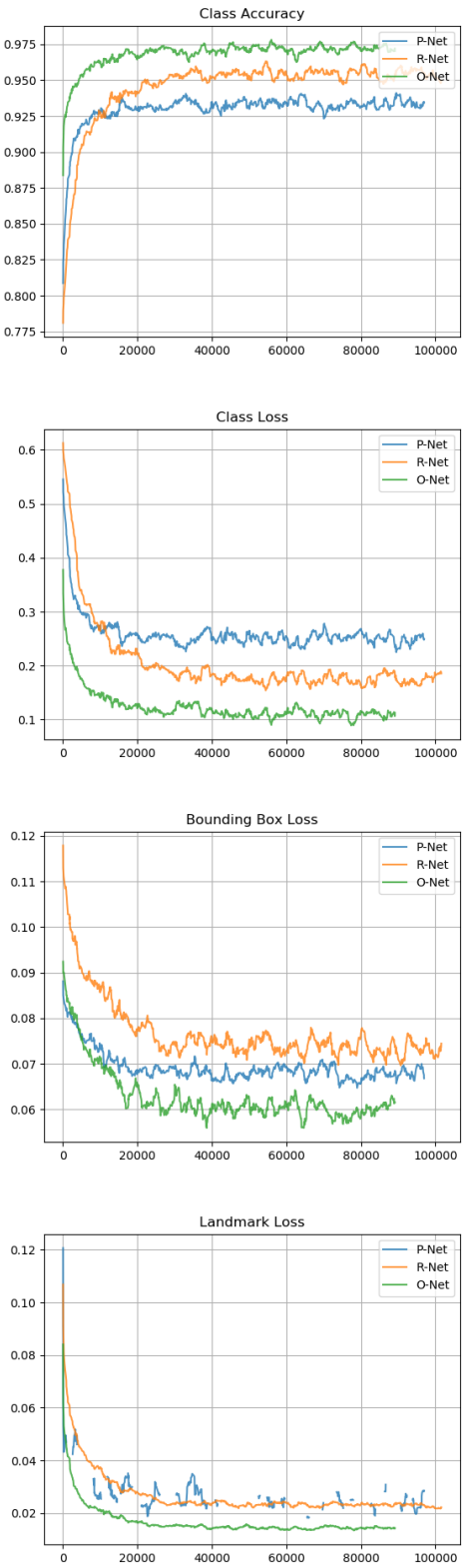


Fig. 3: Traing Process

that our model seldom makes wrong predictions, but may be too conservative that it misses a lot of faces, and this conservation is caused by some image specific traits. We

looked into the cases with low recall rate, and finds that these images are usually taken from a very far perspective from the crowd, leaving the faces hardly recognizable even by human eye. We proposed an improvement on this problem, which we described in section VII.

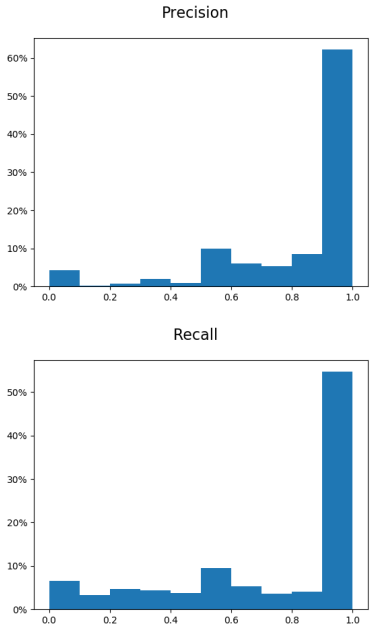


Fig. 4: Precision Recall Distribution

VI. PREDICTION PERFORMANCE

In this section, we will present how our model performance in the different cases such as different quantity of people, various poses, occlusion and different illumination.

Fig. 5: Test Result of Image Los Angeles Lakers

From the figure 5 we could see that all targets are labeled by the drawing tool from the openCV with a white bounding box surrounding the faces, points of five landmarks and the classification confidence on the upper left. The reason we choose this image as our example is that it contains all three challenges mentioned in the first section that we need to overcome.

Fig. 6: Occlusion Cases

Figure 6 shows the occlusion cases presents in our test image, as we can see, no matter the overlapping occurs in vertical or horizontal, it can be well handled by our model.

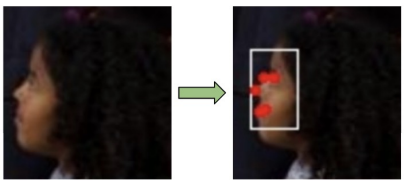


Fig. 7: Various Poses Cases

Figure 7 shows that the side face only presents the left eye can be successfully detected, evaluating that our model is effective to various poses.

Fig. 8: Illumination Cases

Figure 9 shows that even the face with extremely low illumination can be well detected. Although the face is hardly viewed even by human eyes, the data augmentation implemented during the training process ensures that our model is adequate to handle these low illumination cases.

Fig. 9: Large Number Cases

Figure 9 is the photo of all marvel stuff for movie The Avengers, our model successfully caught all 826 faces presented in the image and labeled each face with bounding box, landmark points and corresponding classification confidence with the average prediction time 9.68 seconds. From the result can we found that our model performed well in large number case where targets had different depth of field, poses and overlapping situations.

VII. ANALYSIS AND IMPROVEMENT

The original model only use scaled down image in the inference cascade. This design may due to consideration on efficiency, as P-Net works in a slide window way, iterate

Rate	Precision	Recall	Detection Rate
Original	0.9717	0.3353	0.8397
Scaled Up	0.9534	0.3873	0.8527

TABLE I: Testing Result On WiderFace

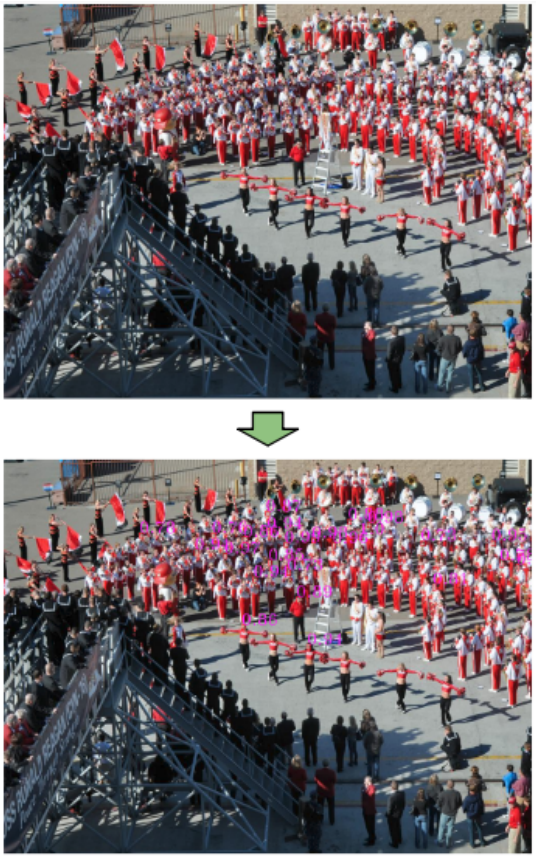


Fig. 10: Prediction Comparison of Original Model and Model with Modified Cascade

through a large image could significantly increase executing time. The cost of this is bad recall rate, especially in images with very tiny faces. As our application scenario demands recognition ablility in this case, we added a scaled up image, with the same scaling ratio in the original model, to the cascade. Table I lists the comparison result on WiderFace model and our modified model. We can see a significant improvement on the recall, while not affecting the precision greatly. The recall and precision follows the definition in 7. The detection rate also sees an moderate improvement.

Figure 10 compares the prediction result of the original model and improved model. This picture is taken from the WiderFace validation set, sketching a parade crowd from above. At the top is the prediction result of the original model, we can see that it fails to detect any face in the image. At the bottom is the prediction result of the improved model, each pink box and number implies a predicted face with its confidence. People in the middle of the picture facing the camera are successfully detected by the improved model

We implemented this system in Google Tensorflow framework closely after this code (`https://github.com/AITTSMD/MTCNN-Tensorflow`). As we profile this implementation, we noticed that the inference stage of P-Net is a bit time consuming, especially when we feed in an image with several hundred faces. Our modified input image cascade adds to this situation. The original code preprocess the cascade sequentially, and call a separate session run for each image in the cascade. We multi-threaded this cascade process, and runs the session only once with all processed data. For an image with several hundred faces, this gets us a performance gain at around 10%.

As for the model implementation attempts, we have tried reproduced the original model as well as the corresponding improvement on two different platform, tensorflow and MXNet, both via the Python API. We use the validation set from the WIDER FACE as our test case. As the result, Tensorflow took 2.16 times more average prediction time than MXNet whilst needed 20.8 times more GPU storage than MXNet. It is obvious that MXNet present a significantly faster, more memory-efficiency performance compared to the tensorflow. The difference is mainly determined its different bottom implementation mechanism. Firstly, MXNet's data structure uses NDArray which can automatically allocate the data to be exacuted to multiple GPUs and CPUs for calculation, thus completing a high-speed parallelism. However, tensorflow's data structure uses tensor and cannot be automatically paralleled. Secondly, tensorflow's matrix operation uses eigen while MXNet uses Mshadow which is faster than the former one.

VIII. Application

In this section, we are going to introduce the application system we built to make the detection tasks more complete and practical. There were three main platforms we took into account, IOS and Android for mobile application which enable users to upload their photo to get the corresponding total number of crowds, another independent single-page web application that visualized all information data such as time point, locations and average crowds counts which made it easier to directly show the quantity relationship among time period,quantity and locations, it is also helpful to do some further statistic analysis such as the pedestrian estimation analysis in CBD area or the variation of peak time for certain places from week days to weekend.

A. Architecture

The architecture of our application system is shown as below:

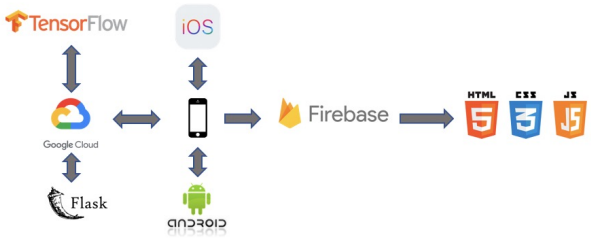


Fig. 11: The architecture flow of application system

Google Cloud Platform: We set up a computing engine for model training and prediction work, it also served as a server providing the back-end logic to our applications. It is a remote virtual machine that is easy to access and deploy Here is a short summary of the parameters of this instance:

- CPU: Intel Broadwell 4vCPU
- Memory: 15GB
- GPU: NVIDIA Tesla P4 $\times$ 1
- Storage: 300GB hard drive

And the summary of software dependency

- Operating System: Ubuntu 16.04 LTS
- CUDA 10.0.130
- Python package management: conda 4.6.14
- Python 2.7.16
- Tensorflow 1.13.1
- OpenCV 2.4.11
- EasyDict 1.9

Flask: is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. Flask is used to create RESTful web service in our application system, where we defined the interfaces between client and server to transfer data via HTTP requests and send the response. In another word, it served as the back end which handle the whole business logic such as receiving the image files, running the detection task and returning the results. There are two main interfaces we defined which are shown as below:

URL	Request Type	Parameter & Data Type	Response
/up_photo	POST	photo: String latitude: Float longi: Float	status: String file: String num: Integer
/download	GET	file: String	image: imageFile

TABLE II: Interfaces for Image Upload and Download

According to the interfaces listed above, the first interface is responsible to image uploading via POST HTTP request, it accepts a String which indicates the image file generated by the mobile end, two Float data represents latitude and longitude. After running the detection on the server it sends the response to the front-end, the certain response contains the request status, filename of the result image and the total number of people from the detection result. The second interface, which handles the acquirement of corresponding image that user requests for, accepts a String to look up the image file on line then return the whole image files.

Mobile Application: In order to provide service to all users, we have developed the mobile applications on the both two mainstream platform, IOS and Android. Both platforms support the function that uploads image from camera capture and photo library then checks detection result(the image and the total number).After acquiring the result to a certain task, it immediately communicates to the real-time database(Firebase) and uploads all the information data of the certain task to support the subsequent data visualization and statistics function on the web application. The process of design and development will be elaborated in details in the next session.

Firebase: The data storage of our system is supported by Google Firebase, the option we chose is the real-time databse which is a NoSQL and cloud-hosted database. Data is stored in JSON format and get synchronized in realtime to each client who is connected. When we need to build cross-platform apps with Android, IOS and Web, we can use the respective SDKs to communicate with the real-time database to do the read and write operations. All clients from different platform share one real-time data base instance and will receive the updates automatically with the latest data. The structure of our data storage is shown as below:

```
1557813452547:{
    "Address": "The University
        of Melbourne,Parkville,
        Australia",
    "Lati":-37.798736572265625,
    "Longi":144.96318440897852,
    "Num":18
}
```

Listing 1: Data Structure Used on Real-time Database

Each uploaded data is structured as a JSON object, where we use the timestamp as the key which is generated in the back-end logic when the detection task is triggered. Each value of certain key contains the information data of the detection task such as the coordinates of the location, its reversed address and the total number of the targets. The data is collected by the mobile end and updated by IOS and Android SDKs and claimed by the web app similarly use the independent SDK. The main steps to implement the Firebase are listed as below:

1) Add Firebase to the app by generating the configuration file using project bundle ID.
2) Add and install the SDK which is platform dependent.
3) Initialize Firebase in app which is programming language independent.
4) Command the query language and code the query statements according to the business logic.

Web Application: In order to make better use of data collected to realize its value as much as possible, the single-paged web application was developed to visualize the location and quantity data. It is developed based on HTML, CSS and related javascript frameworks. The design and the development process will be discussed in the next session.

B. IOS Implementation

The IOS application for the detection works is built for devices with IOS system version in 11.0 or the later and it is developed by Swift 4.2 on Xcode. There are several main frameworks we used to achieve a high-level user experience. **Alamofire:** is an HTTP networking library written in Swift, it provides an elegant interface on top of Apples Foundation networking stack that simplifies a number

of common networking tasks. Alamofire also provides chainable request/response methods, JSON parameter and response serialization, authentication, and many other features that make it easier to apply the RESTful API.

AlamofireImage: is an image component library for Alamofire that supports Image Response Serializer, image cache and image downloader that enable us to achieve the image file downloading and caching operations in a more neat way.

Firebase Series: a series of SDKs enable the communications between mobile ends and Firebase server end.

As for the UX design, we finally implemented six pages including the launching page to complete all requirements in terms of the detection execution and result checking, those are New Task page, Image Adding page, Image Selection page, Confirmation page and Result Checking page as well. The image below shows the UX model of our application:

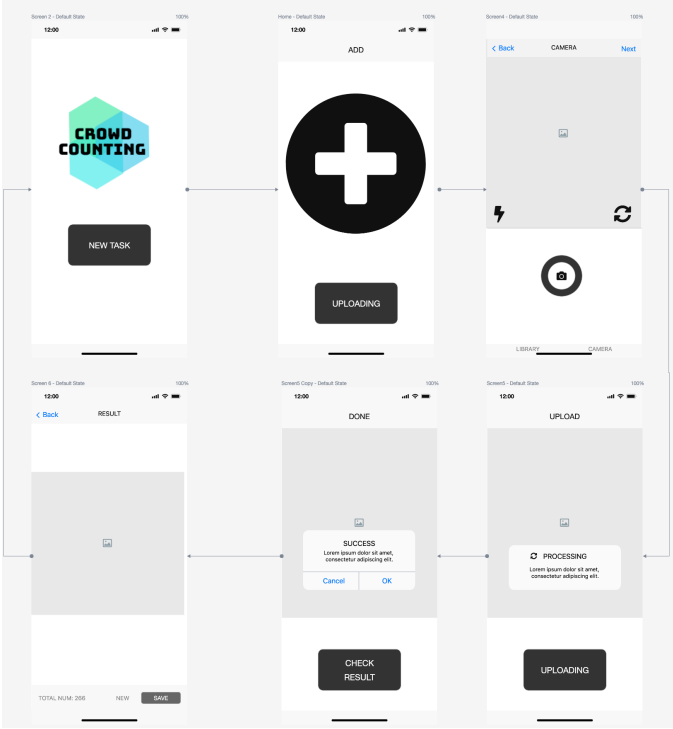


Fig. 12: UX Flow for IOS Application

According to the figure 12 can we find that the UX flow is a closed loop, it starts with a launching page with a button which is linked to the second page where users could start importing the images, after tapping the central button it will be navigated to the camera view as default, users can also import photo from the local library by switching the tab bar on the bottom of the layout, if users tap the Next button on the top right, the POST request containing the selected image file will be sent to the server by calling the uploading function implemented by Alamofire, during the uploading and detection process, there will be a toast floating on the central screen noticing that it is processing, once the mobile end receives the response, it parses the JSON strings and extract all the parameters then all the key-value pairs will be stored on the Firebase including the reversed address generated by its coordinates via CLGeocoder class. At this point, there will be a alert popped up stating that the detection result is ready to check, if users tap the confirmation option, all the data will be sent to the subsequent controller and the current page will be skip to the last page on which the result image and total counts are displayed. The image file is downloaded by sending the GET request with the parameter of filename passed from the last controller. Besides that, users can also save it in the local photo library. To start a new task on this page, users are able to tap the New button to return back to the homepage.

The final demonstration of the IOS application with the UI effect is show by the video with its link listed in the Appendix F.

C. Android Implementation

Our Android implementation has the similar business logic with that in IOS but subtle difference in design to meet the user behavior for Android devices.

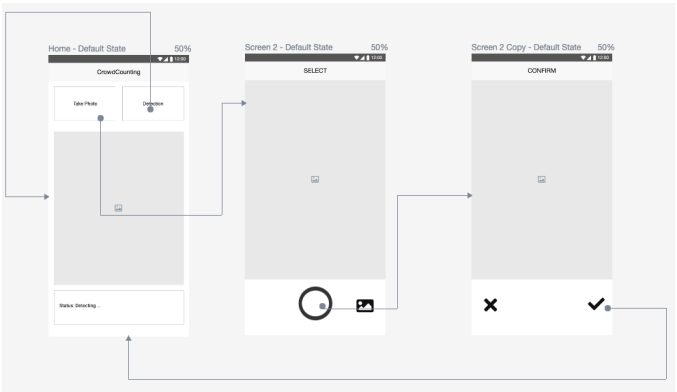


Fig. 13: UX Flow for Android Application

There are total three kinds of layout in our applications, the first page is reusable in both homepage, uploading page and result page. To start from the first page, the upper left button links to the image importing page, after choosing the target image, users are able to tap confirmation button and the view is brought back to the first page with the selected photo displayed on the central image view, by tapping the upper right button can user trigger the uploading and detection work, at this step, the processing statue is keep changing at the bottom of this page. Then the result image with the bounding boxes will take place of the original one, the corresponding counts will be shown on the status area as well.

The final demonstration of the Android application with the UI effect is show by the video with the link in the

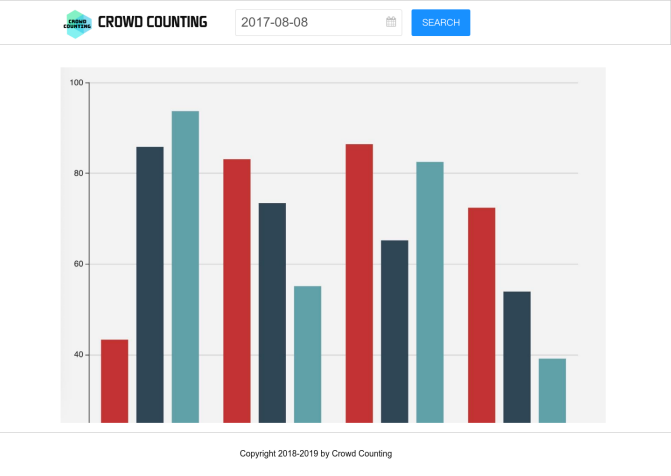


Fig. 14: UX Flow for Web Statistics Platform

Appendix F.

D. Web Statistics Platform

This single-paged web application is built to visualise all the data collected by users to provided a visual display to administrators or any one who may concern about the feature that the data presents. In the header of our web page the selection box can be found which enables users to select the time period they want to check, after the SEARCH button is clicked,the time period will be converted into timestamp to match corresponding data retrieved from the database that uses timestamp as the key for each data object. Then the data will be grouped by its location and ordered by its average counts up to the current date. The X axis of the chart represents all the locations where users uploaded the photo and ran the tasks during the selected period while the Y axis shows the corresponding average count for each location in the period up to the current date. Figure 4.3 shows the user experience design for our platform, the real implementation is presented by the video with the link in Appendix F.

IX. CONCLUSION AND FUTURE WORKS

This thesis mainly introduces a crowd counting application system for object detection using multi-task cascaded convolutional neural network. The model finally performed well even on the image consists of occlusion, various poses and illumination situations. A amount of work was also done to improve the original model such as adding scaling up operations during the inference stage of proposal net to make it more sensitive to tiny objects therefore improve the recall value, we also parallelized the detection phases for each scaled image in the cascade, it accelerates the P-net phase by average 10 percent approximately, the performance is much more obvious when the image contains hundreds of people. Different implementations were done on both MXnet and tensorflow and finally we found that model implemented by mxtnet presented more time and storage efficiency than that in tensorflow. In addition, the application was developed simultaneously on three platforms, IOS and Android releases support users to proceed detection tasks, Web platform to visualize the statistic information among the data. All those applications provides reasonable user experience and user interface design that achieved our goal of building an user-friendly application system.

There are a number of possible future works we can apply to improve the current system. As for the model section, the first thing we will consider is to use the dynamic minsize in the P-net process according to the input image size instead of using the constant value, since when the input size is very large, the minimum face size is not 12px anymore. In that way, we calculate the corresponding resize iterations based on the raw image size therefore avoid unnecessary waste. Secondly, when generating the image pyramid, the resize process can be done based on the previous result instead of the original image. All those two methods aim to accelerate P-net as it is the most time consuming part among those three networks. Besides, the output of our model can be fed into another pre-trained model to do the further gender recognition via transfer learning. As for the application work, the mobile end can be rebuilt with React Native framework to unify the user interface across different platforms therefore making it much easier to maintain and also easier to carry out further extension in the future. The last improvement we plan to do is to add more additional statistic tools in our web platform to make it more comprehensive and practical.

X. APPENDIX

A. Intersection-over-Union (IoU)

Intersecton-over-Union value is a measure of how two boxes in $\mathbb{R}^2$ resemble each other. Denote the two boxes as $B_1, B_2 \subset \mathbb{R}^2$, their area as $|B_1|, |B_2|$,

$$IoU(B_1, B_2) = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|} = \frac{|B_1 \cap B_2|}{|B_1| + |B_2| - |B_1 \cap B_2|}$$

It is obvious that totally overlapping boxes will have IoU value of 1, while disjoint boxes will have IoU value of 0.

A direct attempt to compute IoU value may involve a lot of corner cases, as the relative position of B_1, B_2 can be numerous, which greatly affects the calculation of the intersection area. Here we discribe a short algorithm of simplicity.

Assume the top-left corner and bottom-right corner coordinates are $(x_1^1, y_1^1), (x_2^1, y_2^1)$ for B_1, $(x_1^2, y_1^2), (x_2^2, y_2^2)$ for B_2. First calculate:

$$x_1 = \max(x_1^1, x_1^2) \quad (8)$$
$$y_1 = \max(y_1^1, y_1^2) \quad (9)$$
$$x_2 = \min(x_2^1, x_2^2) \quad (10)$$
$$y_2 = \min(y_2^1, y_2^2) \quad (11)$$

And the area of $B_1 \cap B_2$ is given by

$$|B_1 \cap B_2| = \begin{cases} 0 & \text{if } x_1 \geq x_2 \text{ or } y_1 \leq y_2 \\ (x_2 - x_1)(y_1 - y_2) & \text{otherwise} \end{cases}$$

Note that this algorithm works in $\mathbb{R}^2$. Actual image might be stored differently in an array, and certain coordinates should be adjusted accordingly.

B. Non-Maximum Suppression (NMS) Algorithm

Non-Maximum Suppression is an algorithm used for eliminating overlapping redundancy in a series of bounding boxes. Assume $B_1, B_2, \ldots, B_n \subset \mathbb{R}^2$ is a series of boxes, with corresponding confidence score $c_1, c_2, \ldots, c_n$. Given a threshold T, this algorithm proceed as follows:

1) Sort the box sequence in descending order with respect to confidence score, the resulting sequence is $B_1^1, B_2^1, \ldots, B_n^1$, with confidence $c_1^1, c_2^1, \ldots, c_n^1$
2) Keep B_1^1, iterate through $B_2^1, \ldots, B_n^1$, discard B_i^1 if $IoU(B_1^1, B_i^1) > T$, the resulting sequence is $B_{i_1}^1, B_{i_2}^1, \ldots, B_{i_{n'}}^1$.
3) Repeat step 1, 2 for $B_{i_1}^1, B_{i_2}^1, \ldots, B_{i_{n'}}^1$, until sequence is empty.

C. Offset Coordinate

The scale of coordinates of bounding boxes and facial landmarks are affected by the size of a given image or the scaling of that image. We normalize these coordinated by introducing the offset coordinates.

1) Box Offset: Box offset is defined for positive and part crops. Assume the crop $\mathbf{c}$ has left top point (x_1, y_1) and right bottom point (x_2, y_2), the ground truth box $\mathbf{t}$ has coordinates (xt_1, yt_1), (xt_2, yt_2). The **offset coordinates** of $\mathbf{c}$ with respect to $\mathbf{t}$ is defined as:

$$\begin{aligned} x_{off,1} &= \frac{xt_1 - x_1}{x_2 - x_1} \\ y_{off,1} &= \frac{yt_1 - y_1}{y_1 - y_2} \\ x_{off,2} &= \frac{xt_2 - x_2}{x_2 - x_1} \\ y_{off,1} &= \frac{yt_2 - y_2}{y_1 - y_2} \end{aligned}$$

This definition is suitable for measuring the difference of positions between cropped box and ground truth box.

2) Facial Landmark Offset: Let the coordinates of left eye, right eye, nose, left mouth corner, right mouth corner be respectively (x_{le}, y_{le}),(x_{re}, y_{re}),(x_n, y_n),(x_{lm}, y_{lm}),(x_{rm}, y_{rm}). The coordinates of anchor points of the corresponding ground truth box is (x_1, y_1), (x_2, y_2). The offset coordinate of these landmarks are defined with respect to the top left anchor of the ground truth:

$$\begin{aligned} x_{off,le} &= \frac{x_{le} - x_1}{x_2 - x_1} \\ y_{off,le} &= \frac{y_{le} - y_1}{y_1 - y_2} \\ x_{off,n} &= \frac{x_n - x_1}{x_2 - x_1} \\ y_{off,n} &= \frac{y_n - y_1}{y_1 - y_2} \\ x_{off,re} &= \frac{x_{re} - x_1}{x_2 - x_1} \\ y_{off,re} &= \frac{y_{re} - y_1}{y_1 - y_2} \end{aligned}$$

D. Detailed Model Parameter

1) P-Net:

- Input Images: $12 \times 12 \times 3$
  - Conv 1: 10 kernels of 3×3, with stride 1, no padding.
  - Max Pool filter of 3×3, with stride 2, same padding.
- Hidden Layer 1: $5 \times 5 \times 10$
  - Conv 2: 16 kernels of 3×3, with stride 1, no padding.
- Hidden Layer 2: $3 \times 3 \times 16$
  - Conv 3: 32 kernels of 3×3, with stride 1, no padding.
- Hiden Layer 3: $1 \times 1 \times 32$
  - Conv 4: 16 kernels of 1×1, with stride 1, no padding.
- Output Layer: $1 \times 1 \times 16$.

2) R-Net:

- Input Images: $24 \times 24 \times 3$
  - Conv 1: 28 kernels of 3×3, with stride 1, no padding.
  - Max Pool filter of 3×3, with stride 2, same padding.
- Hidden Layer 1: $11 \times 11 \times 28$
  - Conv 2: 48 kernels of 3×3, with stride 1, no padding.
  - Max Pool filter of 3×3, with stride 2, no padding.
- Hidden Layer 2: $4 \times 4 \times 48$
  - Conv 3: 64 kernels of 2×2, with stride 1, no padding.
- Hidden Layer 3: $3 \times 3 \times 64$, flattened to a 576-d vector
  - FC 1: Weight matrix of 128×576, bias vector of 128×1.
- Hidden Layer 4: 128-d vector.
  - FC 2: Weight matrix of 16×128, bias vector of 16×1.
- Output Layer: 16-d vector.

3) O-Net:

- Input Images: $48 \times 48 \times 3$
  - Conv 1: 32 kernels of 3×3, with stride 1, no padding.
  - Max Pool 1: 3×3, with stride 2, same padding.
- Hidden Layer 1: $23 \times 23 \times 32$
  - Conv 2: 64 kernels of 3×3, with stride 1, no padding.
  - Max Pool 2: filter of 3×3, with stride 2, no padding.
- Hidden Layer 2: $10 \times 10 \times 64$
  - Conv 3: 64 kernels of 3×3, with stride 1, no padding.
  - Max Pool 3: filter of 2×2, with stride 2, same padding
- Hidden Layer 3: $4 \times 4 \times 64$, flattened to a 576-d vector
  - Conv 4: 128 kernels of 2×2, with stride 1, no padding.
- Hidden Layer 4: $3 \times 3 \times 128$, flattened to a 1152-d vector.
  - FC 1: Weight matrix of 256×1152, bias vector of 256×1.
- Hidden Layer 5: 256-d vector.
  - FC 1: Weight matrix of 16×256, bias vector of 16×1.

E. GIT Repository

Repository: `https://github.com/Raelyn-Lyu/CrowdCounting`

F. Video Demonstration

IOS Implementation: `https://youtu.be/VFL_A_316eQ`
Android Implementation: `https://youtu.be/7J1_9Szpu1Y`
Web Implementation: `https://youtu.be/I6h30ZQktEM`

REFERENCES

[1] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[3] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. pages 589–597, 06 2016.

[4] V.A. Sindagi and V.M. Patel. Generating high-quality crowd density maps using contextual pyramid cnns. pages 1861–1870, 2017.

[5] Zhang X. Li, Y. and D. Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. pages 1091–1100, 2018.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013.

[7] R. Girshick. Fast r-cnn. pages 1440–1448, 2015.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015.

[9] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[11] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alighment using multi-task cascaded convolutional neural networks. *IEEE Sigmal Processing Letters*, 23(10):1499–1503, 2016.

[12] Abhinav Shrivastava, Harikrishna Mulam, and Ross Girshick. Training region-based object detectors with online hard example mining. pages 761–769, 06 2016.

[13] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object class (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[14] Chunnu Khawas and Pritam Shah. Application of firebase in android app development-a study. *International Journal of Computer Applications*, 179:49–53, 06 2018.

[15] Justin R. Erenkrantz Michael M. Gorlick Jim Whitehead Rohit Khare Roy T. Fielding, Richard N. Taylor and Peyman Oreizy. Reflections on the rest architectural style and principled design of the modern web architecture. 2017.

[16] Juan Ortz Nicols and Marco Aurisicchio. A scenario of user experience. volume 7, pages 182–193, 08 2011.

[17] Xiaobin Li and Shengjin Wang. Object detection using convolutional neural networks in a coarse-to-fine manner. *IEEE Geoscience and Remote Sensing Letters*, PP:1–5, 09 2017.

[18] Mithun Haridas T P and Supriya M H. Face recognition based surveillance system using facenet and mtcnn on jetson tx2. 03 2019.

[19] Mei Ma and Jianji Wang. Multi-view face detection and landmark localization based on mtcnn. pages 4200–4205, 11 2018.

[20] Ferdousi Rahman, Israt Jahan Ritun, Nafisa Farhin, and Jia Uddin. An assistive model for visually impaired people using yolo and mtcnn. pages 225–230, 01 2019.

[21] Yuqian Zhang, Tao Wang, Guohui Li, Jun Lei, and Luyang Wang. Crowd counting using dmcnn. pages 138–144, 03 2019.

[22] Weizhe Liu, Mathieu Salzmann, and Pascal Fua. Context-aware crowd counting, 11 2018.

[23] Yu-qian ZHANG, Wen-qian WANG, Tao WANG, Guo-hui LI, and Jun LEI. Crowd counting in images via dsmcnn. *DEStech Transactions on Computer Science and Engineering*, 03 2019.

[24] Rothe R., Guillaumin M., and Van Gool L. Non-maximum suppression for object detection by passing messages between windows. pages 290–306, 2015.

[25] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 850–855, Aug 2006.

[26] Ashu Kumar, Harmandeep Kaur, and Munish Kumar. Face detection techniques: A review. *Artificial Intelligence Review*, 07 2018.

[27] Baosheng Yu and Dacheng Tao. Anchor cascade for efficient face detection. *IEEE Transactions on Image Processing*, PP:1–1, 12 2018.

[28] Grant Allen. *Layouts and UI Design*, pages 79–105. 01 2015.

[29] Tanveen Kaur. *STUDY OF DEEP LEARNING APPROACHES TO FACE DETECTION AND RECOGNITION, OBJECT DETECTION AND RECOGNITION*. PhD thesis, 01 2019.

[30] Mate Kisantal, Zbigniew Wojna, Jakub Murawski, Jacek Naruniec, and Kyunghyun Cho. Augmentation for small object detection, 02 2019.