Graduation Research 1

Supervisor: Do Phan Thuan
Academic year: 2020/2021
Student: Giap Thi Thuy Van - 20176905

Content

1. Introduction
2. Theory
3. Practice

Introduction

Introduction

People would like to create a machine which can learn, think and act as human.

One highlight problem is that the machine need to distinguish between objects, so that it can know that the thing is -> The most important field of Machine Learning: Classification.

Theory

2.1 Neural Network and Deep Learning

- Introduction to Deep Learning
- Neural Networks Basics
- Shallow Neural Networks
- Deep Neural Networks

2.2 Improving Deep Neural Networks

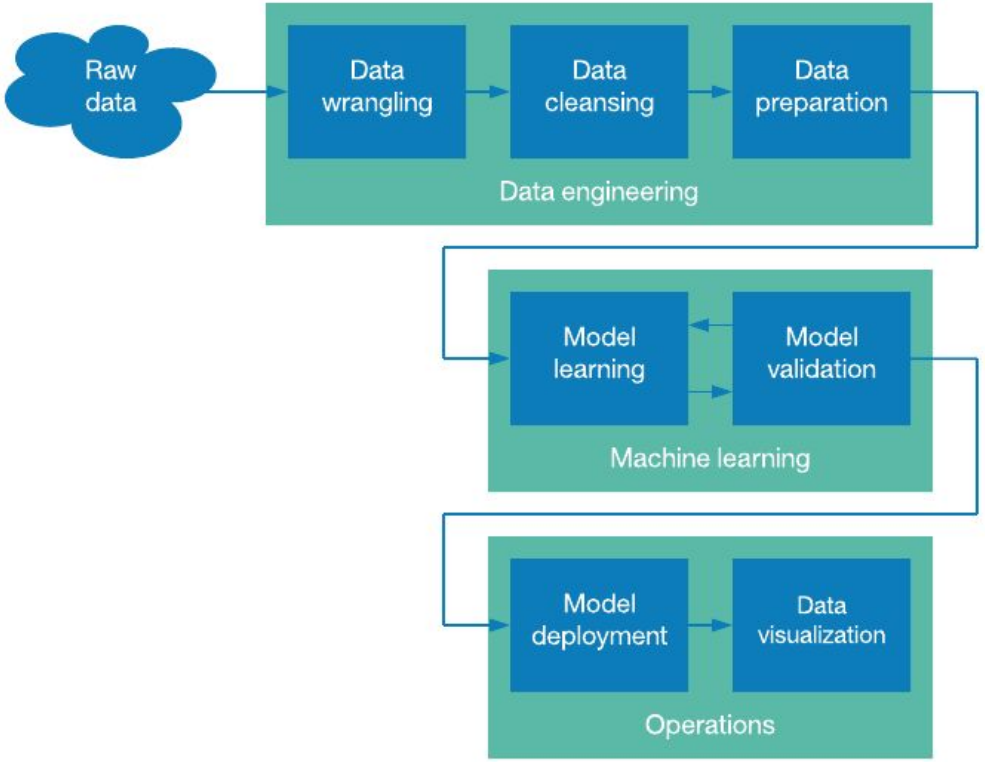
Hyperparameter Tuning, Regularization and Optimization

There are 3 sections:

- Practical Aspects of Deep Learning
- Optimization Algorithms
- Hyperparameter Tuning, Batch Normalization and programming Frameworks

2.3 Structuring Machine Learning Projects

Machine learning strategy



2.4 Convolution Neural Networks

- Foundations of Convolution Neural Networks
- Deep Convolution Models: Case Studies
- Object Detection
- Special Application: Face recognition & Neural Style Transfer

2.5 Sequence Model

- Recurrent Neural Networks
- Natural Language Processing & Word Embedding
- Sequence Model & Attention Mechanism
- Transformer Networks

Practice

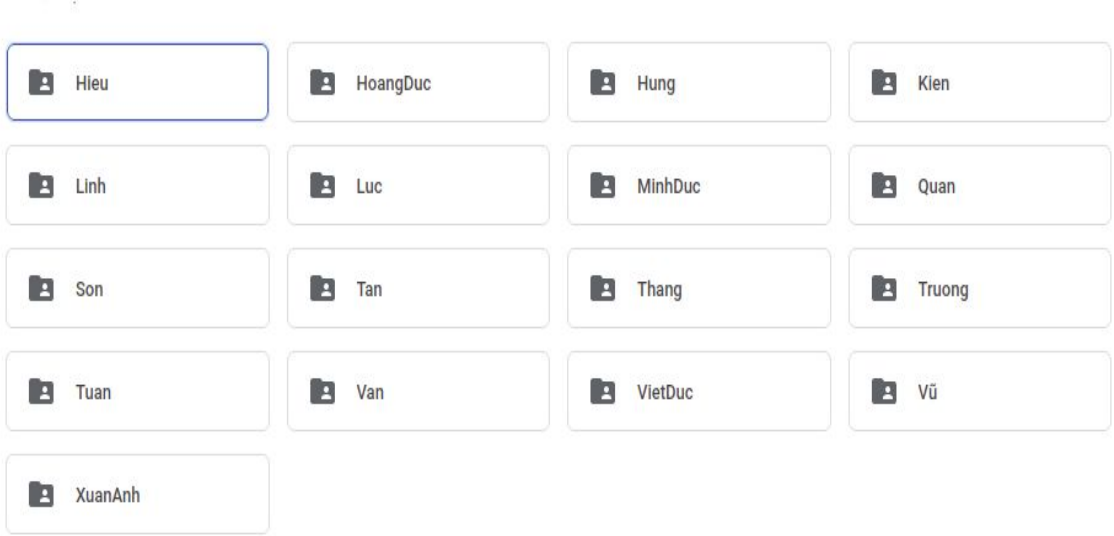
3.1 Face recognition

There are 4 steps in processing:

- Loading images from dataset
- Detecting faces in images
- Pre-processing to embed faces
- Training model to recognize whose the faces belong to

Loading images from dataset

Input data from folders by name of people. Each folder contains personal images by name; and the folder names are automatically used as classes for classification model.

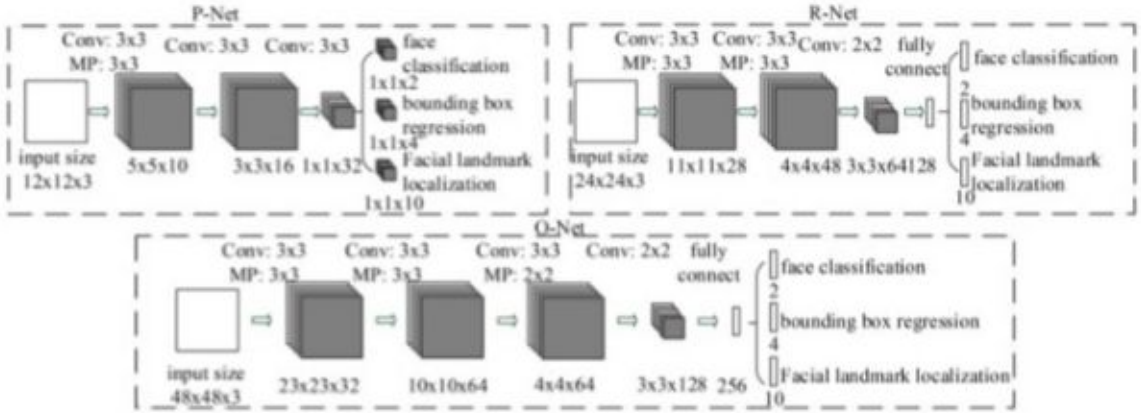


Detecting faces in images

MTCNN stands for Multi-task Cascaded Convolutional Networks.

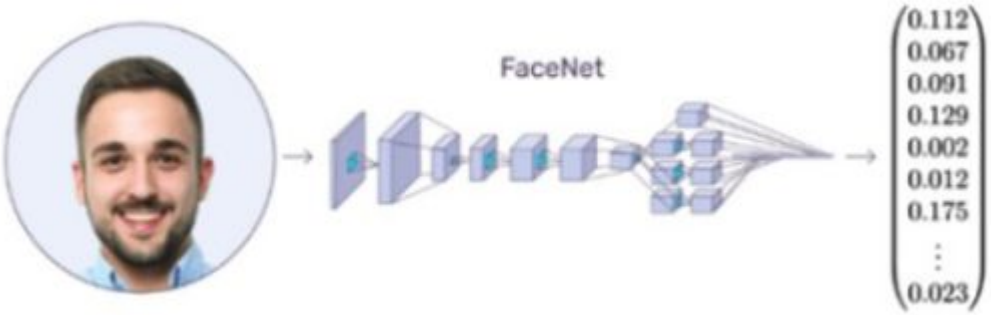
Composing of 3 stacked CNNs and works at the same time when detecting faces. Each network has a different structure and plays a different role in the task.

The output of MTCNN is the position of the face and points on the face such as eyes, nose, mouth.



Pre-processing to embed faces

FaceNet takes a personal face and compresses it into a vector of 128 numbers. Ideally, embeddings of similar faces are also similar.



Label Encoder is used to encode the labels of images to numeric value

Validation labels:

[2 1 0 0 1 0 1 1 0 3 1 0 2 0 3 3 2 0 3 0 3 2 0 1 0 2 3 0 1 1 0 1 0 3 31 1 0 3 1 2 1 0 3 2 2 1 1 1 0 0 2 2 1 0 0 1 3 1 0 1 1 2 1 1 1 0 2 0 3 2 2]

Training model

The following are some Machine Learning models I choose to classification an face recognition:

1. Support Vector Classification
   - Effective in high dimensional spaces
   - Good when there is a clear margin of separation between classes.
2. Logistic Regression
   - Easy to implement, and extend to multiple classes
   - Fast at classifying unknown records
3. Random Forest Classifier
   - Reduces over-fitting in decision trees and helps to improve the accuracy
   - Flexible to boh classification and regression problems

Evaluation and conclusion

three models learn pretty well on my dataset. Even though there are some little differences in validation accuracy of them, these distinctions are insignificant.

-> In order to obtain a result with clearer differences between more, I need to build a dataset with more images with variety of image-capturing condition.

-> Furthermore, I need to adjust and choose some more appropriate parameters so that my models converge as expected in a time-and-memory-limited condition.

Model	Training Accuracy	Validation score
Support Vector Classification	0.997	1.000
Logistic Regression	0.990	0.972
Random Forest Classification	0.965	0.931

3.2 Text classification:

There are 4 steps in processing:

- Data preprocessing
- Word embedding
- Packing data as batches
- Training models

Data pre-processing

Some below steps help documents in dataset to be cleaner and easier for processing.

```
def text_preprocess(document):
    # chuẩn hóa unicode
    document = covert_unicode(document)
    # đưa về lowercase
    document = document.lower()
    # xóa các kí tự không cần thiết
    document = re.sub(r'[^\s\wàáảãạăắằẳẵặâấầẩẫậèéẻẽẹêếềểễệđìíỉĩịòóỏõọôốồổỗộơớờởỡợùúủũụưứừửữựỳýỷỹỵđ_]',' ',document)
    # xóa khoảng trắng thừa
    document = re.sub(r'\s+', ' ', document).strip()
    # tách từ
    document = word_tokenize(document)
    return document
```

Word embedding:

- Word2Vec and FastText representation methods are used to represent words as list of numbers (100 numbers)
- Using a dictionary named w2i (word to index) to install keys that are words in train text dataset; and values are the corresponding ordinal number of the keys in the dict.
- Using a dictionary named i2w (index to word): store the opposite value of w2i.
- Using a list named vocab_embed with a number of rows equal to len(w2i) and a number of columns of 100 (the length of a list representing a word); meaningful at the i-th position is the numeric list representing the i-th word in i2w.

Word embedding:

- Using vocab_embed list to create id_train and id_test that are two lists containing numeric lists representing documents, including lists (length 100) representing words in train set and words in test set.

=> Each document is represented as a list of n (n is the number of words extracted from the text) list of length 100.

- Encoding labels of documents to number type

Packing data as batches

Setting batch_size = 32

=> There are 32 input texts and corresponding labels in each batch.

Since the data set is too large, we cannot put all the data set in for training at a time. Force us to split the data set into batches (smaller size).

Training models

- CNN model structure:

```
CNN(
  (word_embeddings): Embedding(82281, 100)
  (conv1): Conv2d(1, 128, kernel_size=(2, 100), stride=(1, 1))
  (conv2): Conv2d(1, 128, kernel_size=(3, 100), stride=(1, 1))
  (conv3): Conv2d(1, 128, kernel_size=(4, 100), stride=(1, 1))
  (dropout): Dropout(p=0.5, inplace=False)
  (label): Linear(in_features=384, out_features=27, bias=True)
)
```

- RNN model structure:

```
RNN(
  (word_embeddings): Embedding(82281, 100)
  (rnn): RNN(100, 32, num_layers=2, bidirectional=True)
  (label): Linear(in_features=128, out_features=27, bias=True)
)
```

Training models

- LSTM model structure:

```
LSTMClassifier(
  (word_embeddings): Embedding(82281, 100)
  (lstm): LSTM(100, 128)
  (label): Linear(in_features=128, out_features=27, bias=True)
)
```

- Selfattention model structure:

```
SelfAttention(
  (word_embeddings): Embedding(82281, 100)
  (bilstm): LSTM(100, 32, dropout=0.8, bidirectional=True)
  (W_s1): Linear(in_features=64, out_features=350, bias=True)
  (W_s2): Linear(in_features=350, out_features=30, bias=True)
  (fc_layer): Linear(in_features=1920, out_features=2000, bias=True)
  (label): Linear(in_features=2000, out_features=27, bias=True)
)
```

Evaluation and conclusion

Model	Word2Vec	FastText
CNN	81.48%	81.12%
RNN	65.49%	64.62%
LSTM	71.39%	81.99%
SelfAttention	76.43%	85.08%

=> The representation model from FastText gives better accuracy results than the Word2Vec model; that is clearly shown when training with LSTM and Self Attention models.

Question & Answer

Thank you for listening