20
23

Final Project
Group 11

110550063 張博凱
110550076 羅民棋
110550123 黃柏竣
110550157 徐翊芳

20
23

Table of contents

20
23

01

Introduction

20
23

Introduction

- AI applied to medical field
- COVID-19 CT images
  - Why CT?
- Three methods
  - Haar like features & adaboost
  - VGG 16
  - Inception V3

20
23

02

Related work

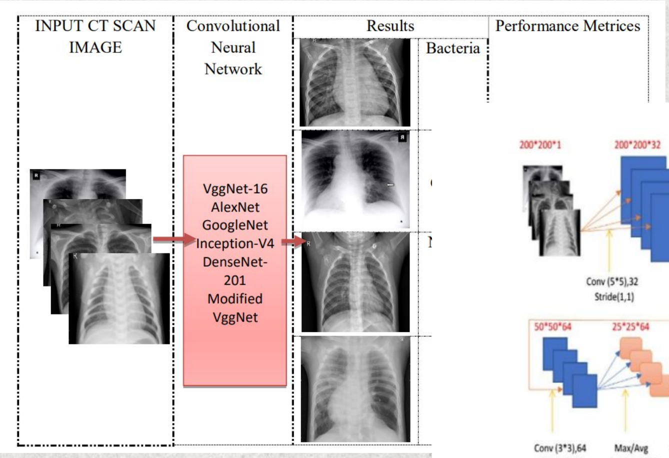
20
23

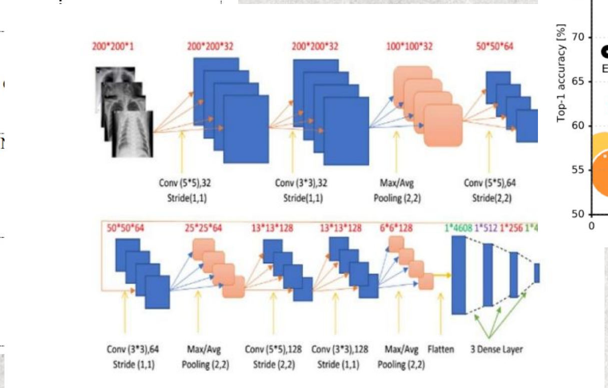
Haar-like feature & Adaboost

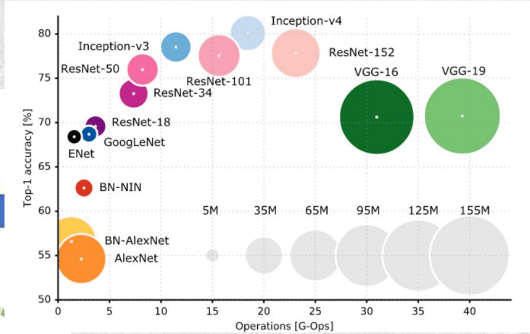
- **HW1**
  - Using openCV to load data (19 x 19) -> 24 x 24
  - Dataset only 400 images in total -> 7955
  - Classify face and non faces

VGG16, InceptionV3

- https://p.migdal.pl/2017/04/30/teaching-deep-learning.html/
- Used for multi-classes classification
- https://iopscience.iop.org/article/10.1088/1757-899X/1084/1/012001/pdf
- Used for collect data and select model that we use to train–







20
23

03

Dataset/Platform

20
23

- COVID and Non-COVID CT images (train:test = 7:3)

	COVID	Non-COVID
train	3729	1840
test	1598	788

2023

04

Baseline

2023

Baseline

- **Haar like feature & Adaboost :**
  - **Haar like feature detector resolution: 24 x 24**
  - **Grayscale**
  - **The number of weak classifiers(T) = 10**

- **VGG16 and InceptionV3:**
  - **Epoch = 10**
  - **Input size: 299 x 299**
  - **Batch size = 32**
  - **Normalize:**
    (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

05

Main approach

Current situation & problems

Current situation

01

Haar-like
Adaboost

02

Vgg16

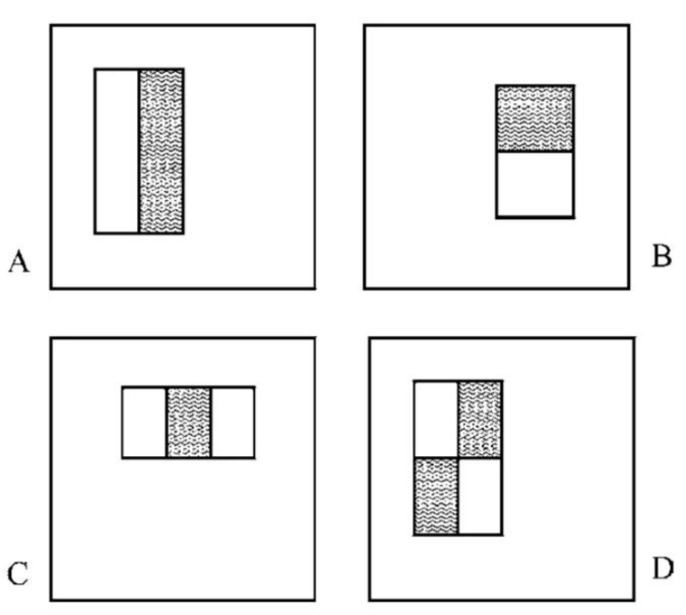
03

InceptionV3

Problems

Haar-like feature & Viola Jones

Haar-like feature



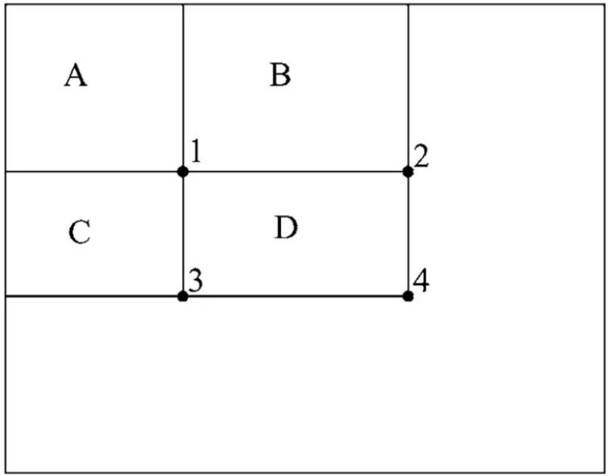
Features are used to extract some useful information, like edges and lines.

The value of the feature is the difference between the sum of the pixels within two regions.

Haar-like feature & Viola Jones

Integral Image

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y'),$$



Integral image is used for fast calculation of rectangle features.

We can easily get the sum of area D by calculating 4 - 2 - 3 +1.

Haar-like feature & Viola Jones

Adaboost

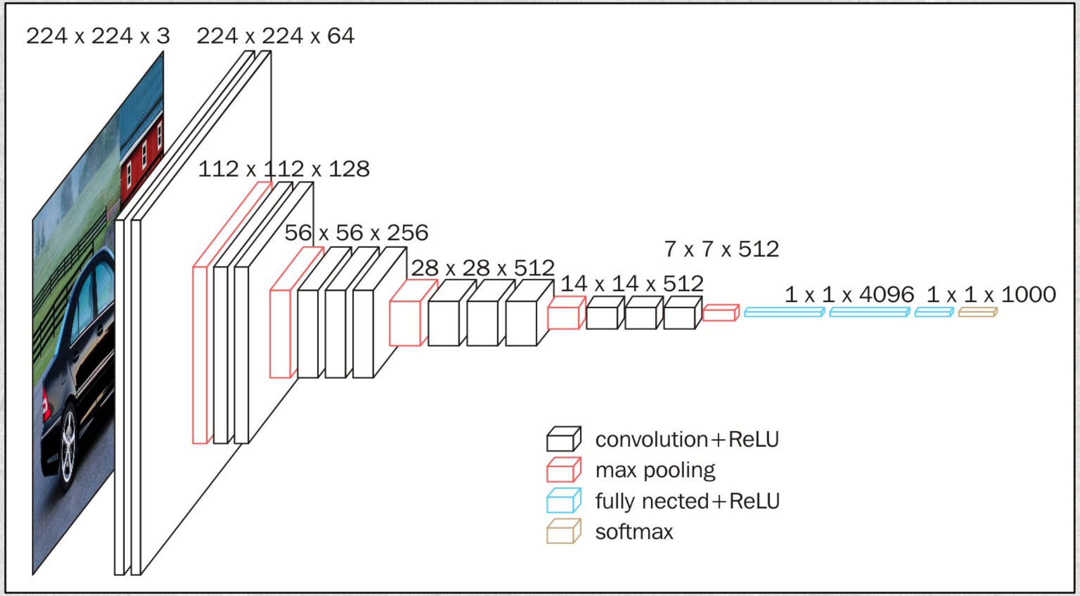
- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \, |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Initialize weights first.

For each loop, normalize the weights and train weak classifier. Next, choose the classifier with the smallest error, then update the weights.

Finally get the strong classifier.

VGG16



Vgg16 is a powerful DL model widely used for image analysis and classification.

- **VGG splits image into R,G,B three channels as input of CNN**
- **Consist of a series of convolutional layers followed by fully connected layers for classification.**
- **Output will be a tensor contains 1000 classifiers**

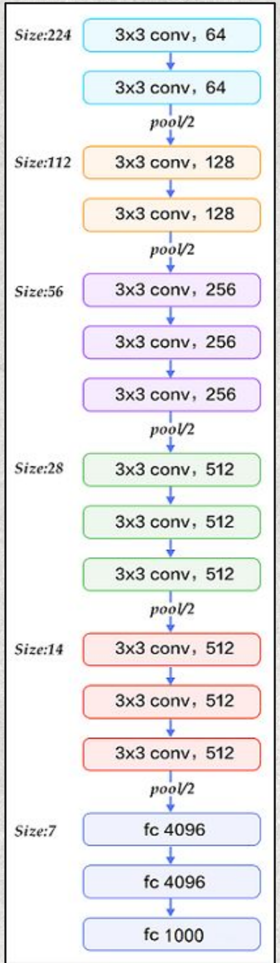
VGG16

Analysis of VGG16

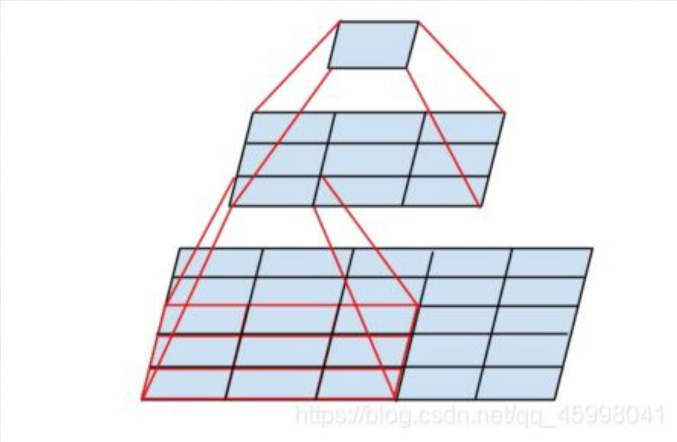
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG is a type of streamlined CNN architecture, known for its consistent size of convolutional layers across the network, making it easy to understand and manipulate. The right figure shows a rough overview of the different types of VGG architectures. It can be observed that all VGG types are divided into 5 blocks, followed by 3 fully connected layers, and finally a softmax layer. Among them, VGG-16 has a similar architecture to type D.

VGG16



The left image shows the architecture of VGG-16, which consists of five blocks and three fully connected layers. Each block undergoes a series of convolutional operations followed by pooling, which passes specific feature values to the next block.



The right image visualizes the process of performing convolution on a tensor.

VGG16

Maxpooling

```
[[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 9, 10, 11, 12],
 [13, 14, 15, 16]]

Maxpool2d (kernel_size=2, stride=2):
[[ 6,  8],
 [14, 16]]
```

Normalization

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

Image below illustrates the number of parameters and computations.

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

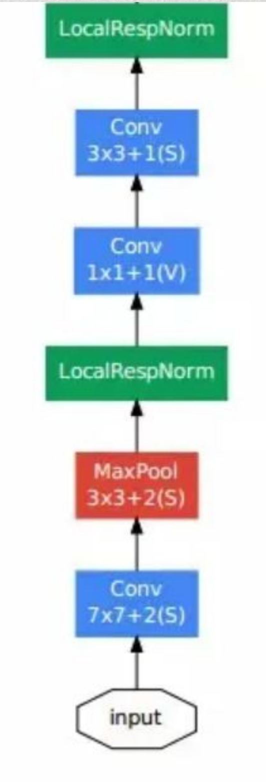
20
23

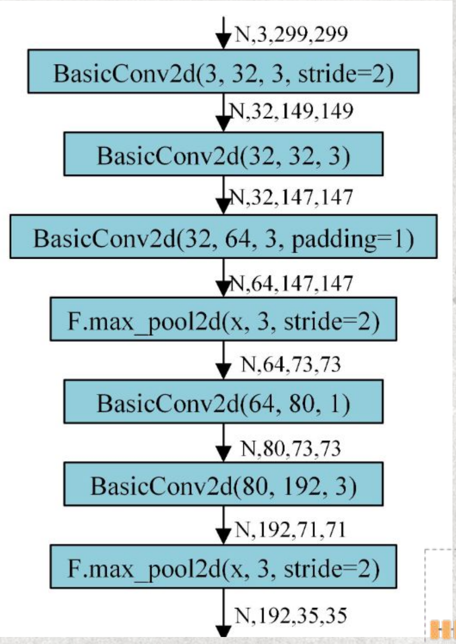
InceptionV3

A classification method

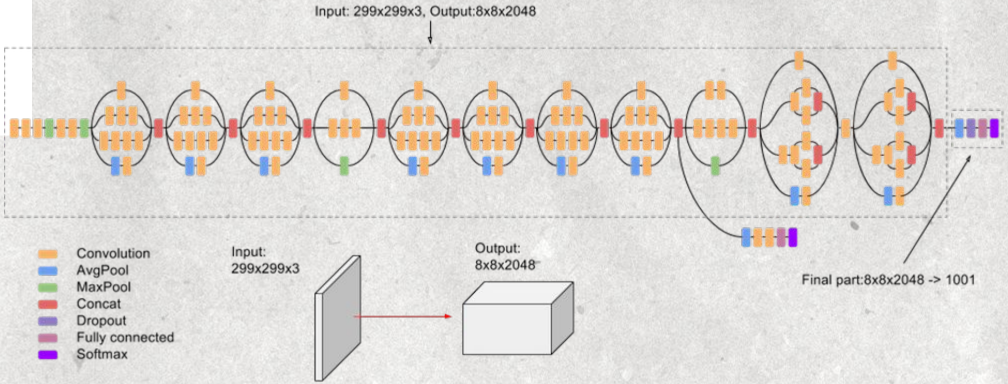
20
23

First Level

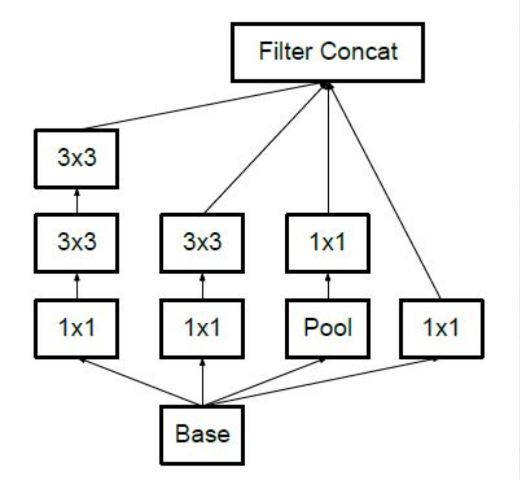




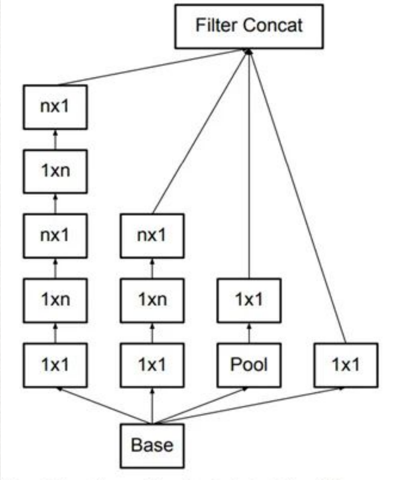
- InceptionV3 use modular organization method like Inception V1.
- The picture in the left is the first block of modular. Our input size is 3 * 299 * 299.
- Through some basic convolution, we get some feature first.
- And there are three Inception module later. They are Inception A, Inception B, and Inception C.



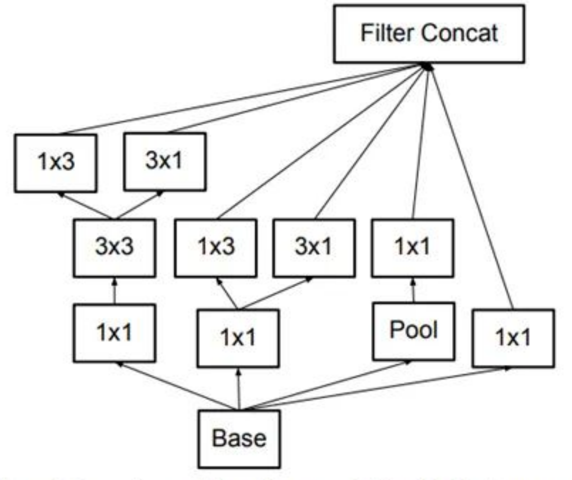
Inception Module



Inception Module A



Inception Module B

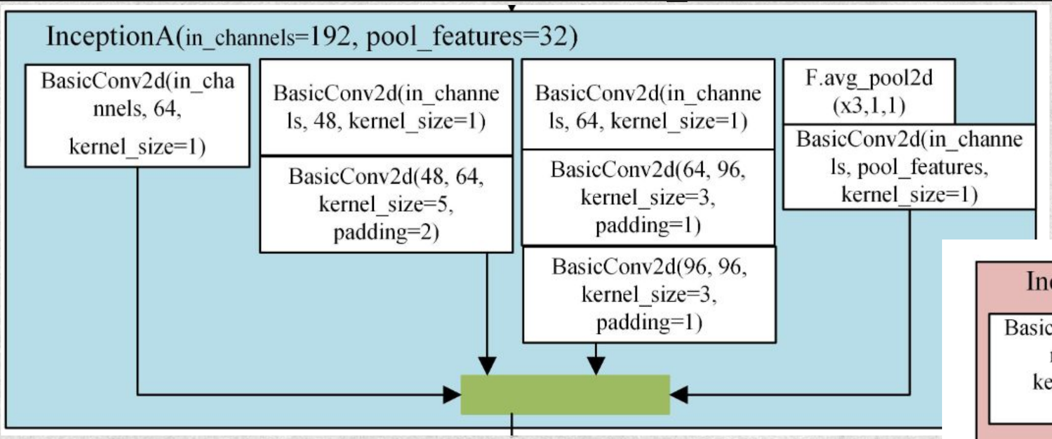


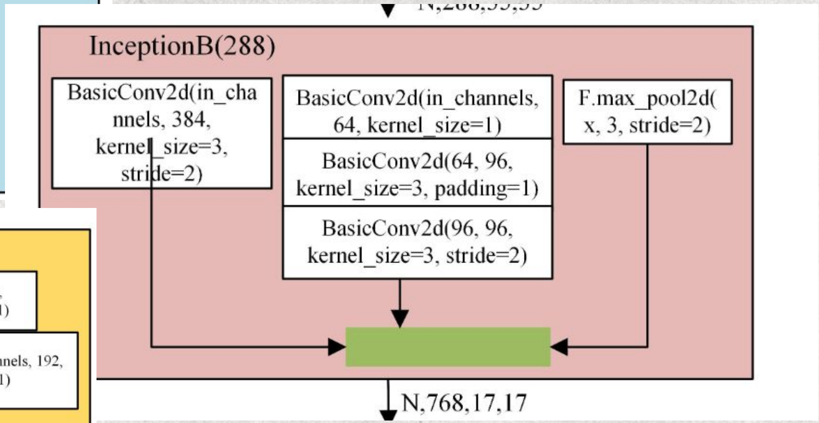
Inception Module C

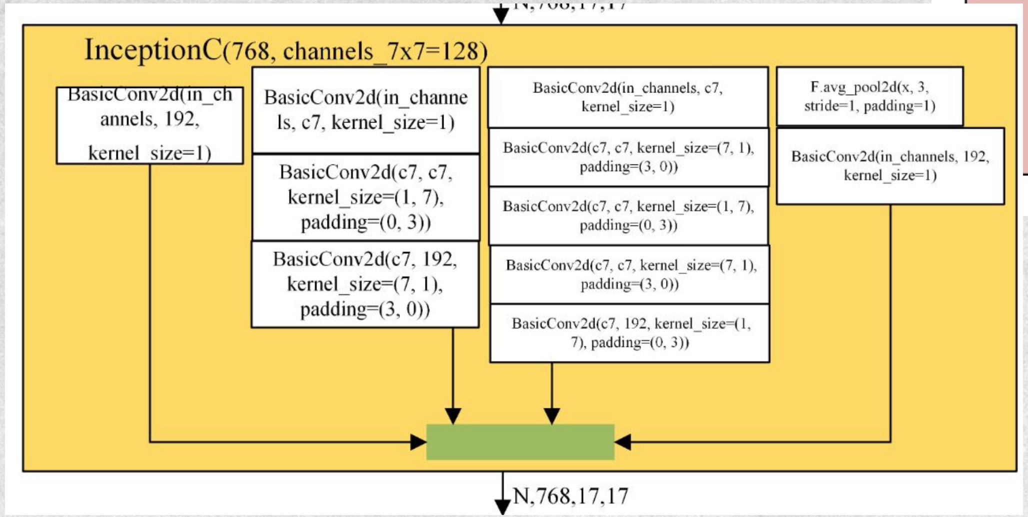
The concept of Inception is easy, we use divide data into four ways to compute. And concatenate them into one then transport it to next step.

- In the **module A** , it replaces a 7x7 convolution layer with three 3x3 convolution layer, Which can achieve same performance but with less computation.
- In the **module B** , it uses asymmetric convolution layer, which reduce the calculation and also have same performance.(In the implementation, n = 7)
- In the **module C** , it expands asymmetric convolution layer across width, which is for promoting high dimensional representation to more features.

Inception Module

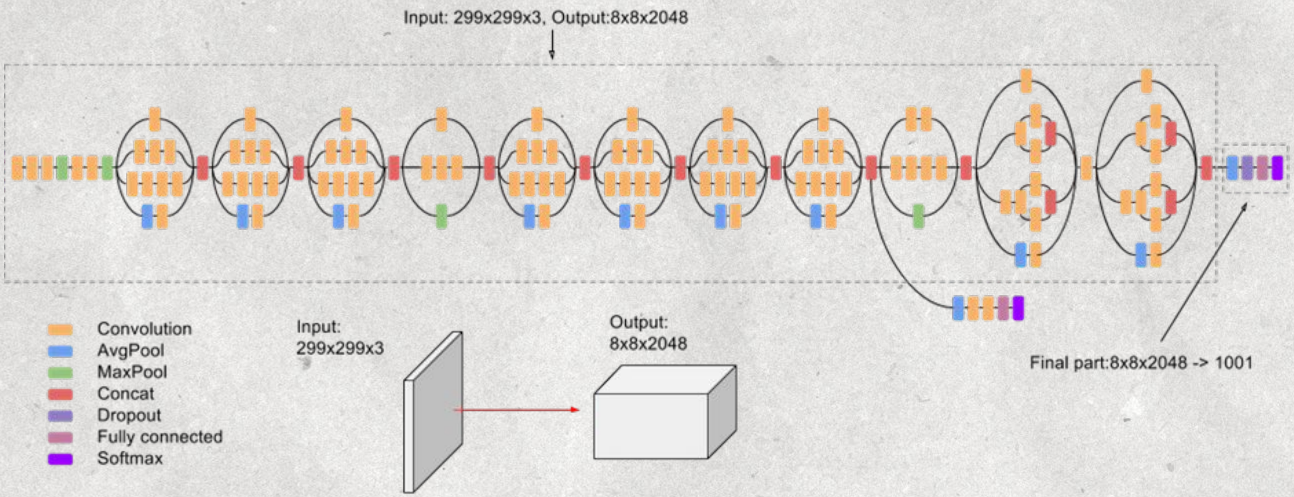






Auxiliary classifiers

- In the Inception V1 model, it uses two auxiliary classifiers. On the other hand, Inception V3 only use one auxiliary classifier. The reason is that we found that removing the classifier in the low-level didn’t have a impact on the performance.



06

Evaluation metrics

Metrics

Accuracy	F1-score
$$\frac{TP+TN}{TP+TN+FP+FN}$$ ● The percentage of correctly identified cases ● Used when the True Positives and True negatives are more important	$$\frac{TP}{TP+\frac{1}{2}(FP+FN)}$$ ● The harmonic mean of Precision and Recall ● Used when the False Negatives and False Positives are crucial

2023

07

Results & Analysis

2023

Results of testing dataset

	Pre trained	Accuracy	F1 score
Haar like & Adaboost (T=10)	X	0.763	0.808
VGG16 (epochs=10)	True	0.961	0.941
	False	0.884	0.814
InceptionV3 (epochs=10)	True	0.958	0.941
	False	0.868	0.815

Analysis

- Difference of performance between haar-like feature & adaboost and others
  - Haar : can only detect specific feature
  - CNN : more complex feature

- Difference between pre trained and non pre trained (when there is no time for large epochs training, it may be a good way to achieve nice performance)
  - Pre-trained has parameter that fits our data, non-pre-trained may need more epoch to get same performance

Limitation

- CT images are not easily to obtain
- Different types of data may have different results, e.g. X-ray

Appendix-References

- Source of dataset: https://data.mendeley.com/datasets/8h65ywd2jr/3?fbclid=IwAR3N9RbCdETV0E35TcUIpPFm1umGn-c2ogijjQuqHXr5mjn6ReRmifx-_3E
- Haar like features: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
- VGG16: https://iopscience.iop.org/article/10.1088/1757-899X/1084/1/012001/pdf https://hackmd.io/@TienYi/pytorch-stanford-dog
- Inception V3: https://medium.com/ching-i/inception-%E7%B3%BB%E5%88%97-inceptionv2-inceptionv3-93cd42054d23

Appendix-Github link

- https://github.com/Alanhsu1/Intro_AI/tree/main

Appendix-
Contribution of each member

- 110550063 張博凱:25% split dataset, debug, make slides, video edit
- 110550076 羅民棋:25% VGG16, collect data
- 110550123 黃柏竣:25% haar-like feature & adaboost and evaluation metrics code, debug, use GPU device, make slides, video
- 110550157 徐翊芳:25% InceptionV3, collect data, help others train model, make slides, Github resources preparation

Thanks for listening~