**人工智能课程总结报告**

**施畅 2015201524**

1. **三次课程作业实践**
2. **第一次展示——小车组装、简单避障与物体追踪**

按照课程指导完成了小车的组装。通过声纳对前方障碍进行测距，并区分前方物体为静物还是运动物体，触发不同功能。

小车的运动速度为预先设定值，由声纳测距可以进一步计算出小车与前方物体相对速度差（该步计算不断循环）。若速度差变化很小，则判定为静止物体，视为障碍物，则转向自动躲避；若速度差变化大于一定阈值则代表前方为运动物体，锁定对象，加速向其进发，直到撞上为止，实现攻击，再继续掉头前进。

1. **第二次展示——基于图像识别的小鼠捉补**

主要负责图片处理机器学习方法，对视野内的小鼠方位进行判定。Droidcam实时传输小车上手机拍摄的视频，电脑端通过4747端口重定向，爬虫以0.5秒为间隔抓取训练图片。实验中具体使用了1000张训练图片。具体算法为HOG特征算子+SVM分类算法。

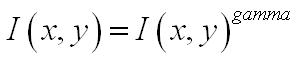
**HOG算法的原理和具体实现步骤：**

1）灰度化（由于颜色在此算法中的作用不大，将图像转化为灰度图）

Y=0.299R+0.587G+0.114B

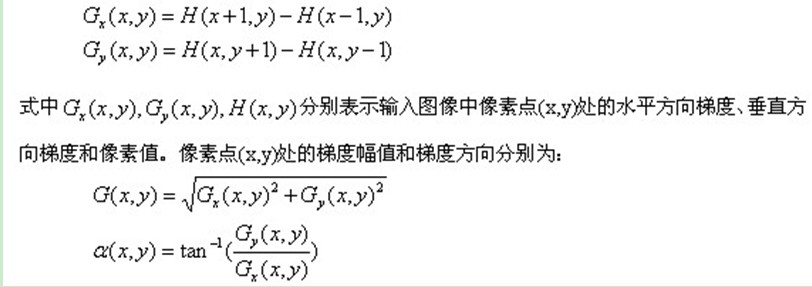
2）采用Gamma校正法对输入图像进行颜色空间的标准化（归一化）

目的是调节图像的对比度，降低图像局部的阴影和光照变化所造成的影响，同时可以抑制噪音的干扰。



一般gamma取0.4或0.45较为符合适中视觉光照

3）计算图像每个像素的梯度（包括大小和方向）；主要是为了捕获轮廓信息，同时进一步弱化光照的干扰。

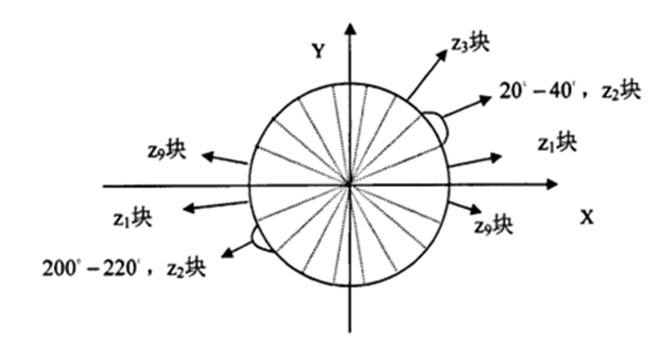


4）将图像划分成小cells（例如8\*8像素/cell）

5）加权投影法（weighted voting）每个cell的梯度直方图，即可形成每个cell的descriptor

具体方法：对每个cell的360度划分bin, 统计在每个梯度方向bin中的像素点个数进行投票，投票权重为改点的梯度幅值（也可使用与幅值相关的函数，但经试验效果没有简单幅值好）

具体操作中，往往使用三线性差值，即将当前像素的梯度方向大小、像素在cell中的x坐标与y坐标这三个值来作为插值权重，而被用来插入的值为像素的梯度幅值。采用三线性插值的好处在于：避免了梯度方向直方图在cell边界和梯度方向量化的bin边界处的突然变化。



6）将每几个cell组成一个block（例如3\*3个cell/block），块内归一化梯度直方图

由于局部光照的变化以及前景-背景对比度的变化，使得梯度强度的变化范围非常大。这就需要对梯度强度做归一化。归一化能够进一步地对光照、阴影和边缘进行压缩。

**注**：block之间的是“共享”的，也即是说，一个cell会被多个block“共享”。另外，每个“cell”在被归一化时都是“block”independent，也就是说每个cell在其所属的block中都会被归一化一次，得到一个vector。这就意味着：每一个单元格的特征会以不同的结果多次出现在最后的特征向量中。这种混叠效应增强了块与块之间的联系，使得局部信息得到连接。

一个block内所有cell的特征descriptor串联起来便得到该block的HOG特征descriptor。

7）将图像image内的所有block的HOG特征descriptor串联起来就可以得到该image（你要检测的目标）的HOG特征descriptor了。这个就是最终的可供分类使用的特征向量了。

**小鼠图片识别算法步骤：**

1）对图片素材进行切割，得到64\*64的样本块。含小鼠的positive样本tag为1，噪声背景的negative样本tag为0.

2）用HOG描述子提取特征后用SVM训练，将训练好的模型存储。后续小车实时拍摄的图片，直接加载训练好的模型进行预测，节省时间，以保证实时性。

3）以64\*64的block在每一张手机拍摄的640\*480的图像上移动，移动步长为64像素。

4）由训练好的SVM训练出每个block是否含有小鼠。将图片分区为左、中、右三个分区，统计出每个分区中含有小鼠的block数量，比较判断后，给图片归类。按照图片归属的“左”、“中”、“右”或“无”类别通过蓝牙发送信号，控制小车左转、直行、右转、持续左转。（当图片识别未发现老鼠时，小车持续原地左转，直到小车视野范围内出现小鼠，开始追踪）

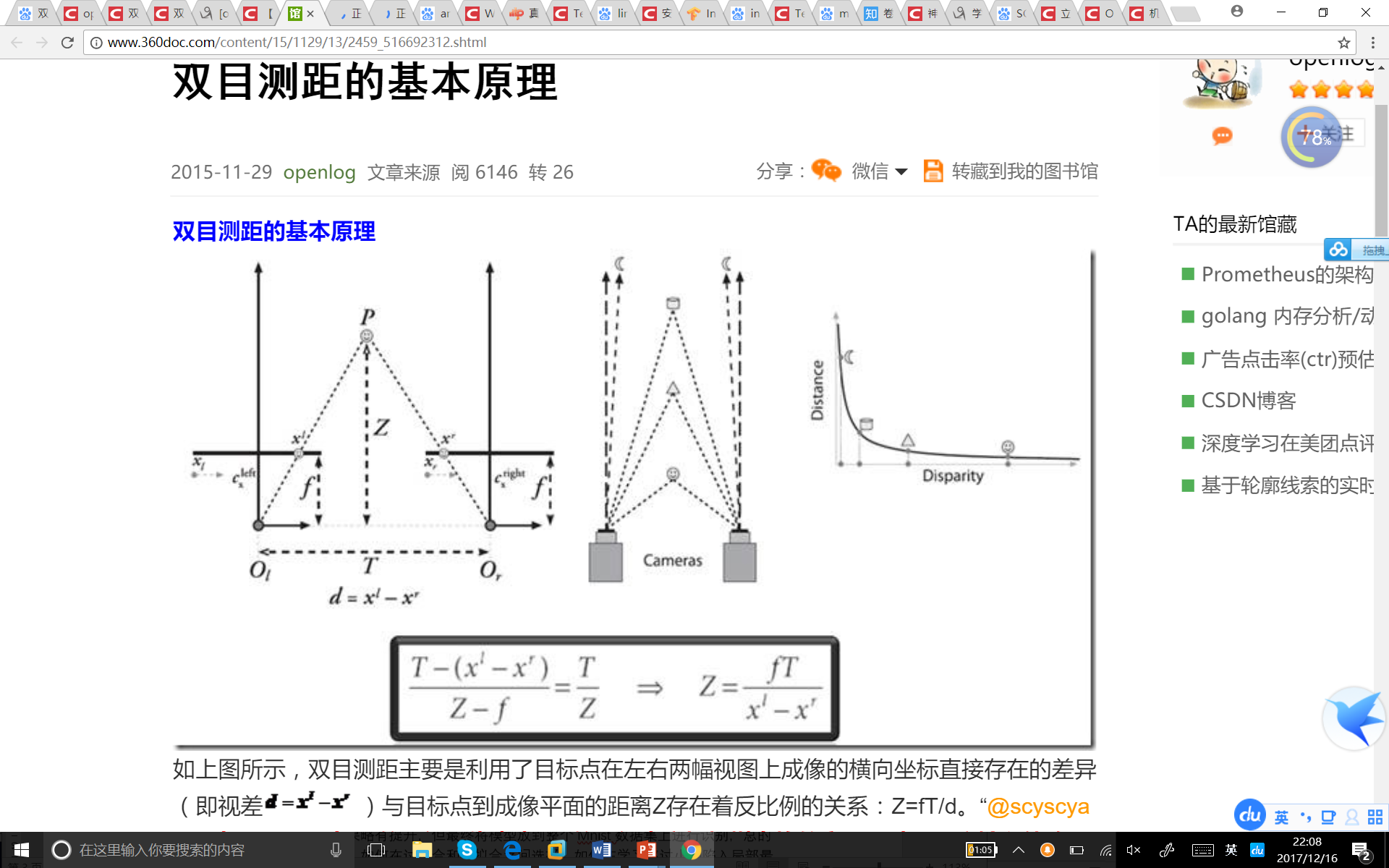
**实验结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| 混淆矩阵 | | | |
|  |  | 测试值 | |
| 真实值 | 总计 | 正例 | 反例 |
| 正例 | 283 | 104 |
| 反例 | 189 | 435 |

算法的准确率71.01%，准确率59.95%，召回率73.12%。我们用了1011张图片进行训练，可能由于训练集不够大，或是算法需要进一步优化，最终识别的准确率为71%。所以为弥补算法上的一些缺陷，我们在小车的追踪方式上采取保守策略，当图片中没有识别出小鼠时，小车原地打转，在2-3圈后基本可以保证准确识别，开始追踪。

**（三）第三次展示——基于双目测距的导盲小车和语音控制**

主要负责对图像使用双目测距算法计算图片纵深实现小车测距的部分。主要算法需要根据同一点在两个摄像头画面中的位移，结合两摄像头间距和焦距进行计算。具体步骤如下：



1）单目定标：需要一个放在摄像头前的特制的标定参照物（棋盘纸），摄像头获取该物体的图像，并由此计算摄像头的内外参数。标定参照物上的每一个特征点相对于世界坐标系的位置在制作时应精确测定，世界坐标系可选为参照物的物体坐标系。在得到这些已知点在图像上的投影位置后，可计算出摄像头的内外参数。

2）双目定标：通过标定来测量两个摄像头之间的相对位置（即右摄像头相对于左摄像头的三维平移 t 和旋转 R 参数）。通过两摄像头之间的位置关系对图像进行畸变消除，把消除畸变后的两幅图像严格地行对应，使得两幅图像的对极线恰好在同一水平线上，这样一幅图像上任意一点与其在另一幅图像上的对应点就必然具有相同的行号，只需在该行进行一维搜索即可匹配到对应点。

3）立体匹配：主要采用SGBM算法，opencv中提供了该算法。这是一种全局匹配算法，其中也应用了我们第二次展示中用过的sobel算子。具体步骤包括：

1、采用水平Sobel算子，对图像进行预处理。

2、用一个函数将处理后的图像映射乘一个新的图像。

3、进过基于采样的方法计算得到SAD（sum of absolute differences）代价

4、动态规划和后处理

具体的算法效果还在调优过程中，具体计算得到的图片深度和实际距离的误差将在下一次展示中展现。

**二、课程内容实践**

**（一）Mnist手写体数字识别**

根据神经网络的原理，使用c++构建了全连接层、softmax损失函数、ReLU激活函数。使用了一层神经网络加激活层，手写体数字识别的正确率已经达到了83%左右，在使用momentum对梯度下降的惯性进行体现后，识别率提高到88%，在继续增加了一层神经网络后，两层的识别率达到了90.3%。但是调整参数的过程很艰难，有时在训练集上训练完，在测试集上的效果略有提升，但最终将模型放到整个Mnist数据集上进行识别，总的识别正确率却下降了。如何在过拟合和欠拟合之间选择，如何在学习率过小和陷入局部最优之间选择，有很多技巧需要学习。

最后测试得到的最优的组合为学习率learning rate=0.1, momentum=0.75.此时在整个Mnist数据集上的识别正确率为92.5%.

(所用数据集为Mnist的一部分部，第一行有四个数2000 500 28 28，表示有2000行训练数据，500行测试数据，28\*28为每个图片的像素个数，从第二行开始每一行除了28\*28个像素点的数值还有一个label为0~9)。

在使用了最简单的神经网络下识别率已经很高，再考虑到图像识别上，局部信息的高度相关性，如果使用卷积神经网络CNN，识别率基本都在95%以上。

具体代码如下：

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <cstdlib>

#include <random>

#include <algorithm>

#include <cmath>

typedef std::vector<float> FLTARY;

class InnerProduct{

public:

int n, m;

std::vector<FLTARY> w,v;

FLTARY b, top, grads;

InnerProduct(int input, int output) {//使用高斯分布对权值和偏置进行初始化

n = output;

m = input;

std::random\_device rd; // random seed generator

std::mt19937 rg(rd()); // random generator

std::normal\_distribution<> normDist(0, 0.1); // normal distribution

w.resize(output);

v.resize(output);

b.resize(output);

top.resize(output);

grads.resize(input);

for(int i = 0; i < output; ++i) {

w[i].resize(input);

v[i].resize(input);

for(int j = 0; j < input; ++j) {

w[i][j] = normDist(rg);

v[i][j] = 0;

}

b[i] = normDist(rg);

}

}

void forward\_pass(FLTARY bottom) {//正向传播

for(int i = 0;i < n; ++i) {

top[i] = b[i];

for(int j = 0; j < m; ++j) {

top[i] += w[i][j] \* bottom[j];

}

}

}

void backward\_pass(FLTARY top\_grads, FLTARY bottom, double momentum, double base\_lr) {//反向传播

//grads.assign(m, 0);

for(int i = 0; i < n; ++i) {

for(int j = 0; j < m; ++j) {

grads[j] += top\_grads[i] \* w[i][j];

v[i][j] = momentum \* v[i][j] - base\_lr \* top\_grads[i] \* bottom[j];

w[i][j] += v[i][j];

}

b[i] -= base\_lr \* top\_grads[i];

}

}

};

class ReLU{//激活函数

public:

int n;

FLTARY top, grads;

ReLU(int output) {

n = output;

top.resize(output);

grads.resize(output);

}

void forward\_pass(FLTARY bottom) {

for(int i = 0; i < n; ++i) {

top[i] = bottom[i] > 0 ? bottom[i] : 0;

}

}

void backward\_pass(FLTARY top\_grads) {

for(int i = 0; i < n; ++i) {

grads[i] = top[i] > 0 ? top\_grads[i] : 0;

}

}

};

class SoftmaxWithLoss{

public:

int n;

FLTARY top, grads;

SoftmaxWithLoss(int output) {

n = output;

top.resize(output);

grads.resize(output);

}

void forward\_pass(FLTARY bottom) {

double ymax = \*std::max\_element(bottom.begin(), bottom.end()), sum = 0;

for(int i = 0; i < n; ++i) {

top[i] = exp(bottom[i] - ymax);

sum += top[i];

}

for(int i = 0; i < n; ++i) {

top[i] /= sum;

}

}

void backward\_pass(float label) {

for(int i = 0; i < n; ++i) {

grads[i] = top[i] - (label == i);

}

}

};

template<typename \_IS>

void LoadData(\_IS &inStream, int \*pImgRows, int \*pImgCols,

std::vector<FLTARY> &trainImages, FLTARY &trainLabels,

std::vector<FLTARY> &testImages) {

int nTrainCnt, nTestCnt;

inStream >> nTrainCnt >> nTestCnt >> \*pImgRows >> \*pImgCols;

int nImgArea = \*pImgRows \* \*pImgCols, n = 41;

trainImages.resize(nTrainCnt);

trainLabels.resize(nTrainCnt);

testImages.resize(nTestCnt);

for (int i = 0; i < nTrainCnt + nTestCnt; ++i) {

std::string strLine;

inStream >> strLine;

std::vector<float> fltBuf(nImgArea);

for (int j = 0; j < nImgArea / 2; ++j) {

const char \*p = strLine.c\_str() + j \* 3;

int rawCode = (int)(p[0] - '0') \* n \* n;

rawCode += (int)(p[1] - '0') \* n;

rawCode += (int)(p[2] - '0');

fltBuf[j \* 2 + 0] = ((rawCode & 0xFF) - 128.0f) / 255.0f;

fltBuf[j \* 2 + 1] = ((rawCode >> 8) - 128.0f) / 255.0f;

}

if (i < nTrainCnt) {

fltBuf.swap(trainImages[i]);

inStream >> trainLabels[i];

}

else fltBuf.swap(testImages[i - nTrainCnt]);

}

}

int main() {

int nImgRows, nImgCols, tp, fp;

double base\_lr = 0.1;

double momentum = 0.75;

std::vector<FLTARY> trainImgs, testImgs;

FLTARY trainLabels;

FLTARY testLabels;

testLabels.resize(500);

// std::ifstream ifs("train\_2000a.txt");

// LoadData(ifs, &nImgRows, &nImgCols, trainImgs, trainLabels, testImgs);

LoadData(std::cin, &nImgRows, &nImgCols, trainImgs, trainLabels, testImgs);

// ifs.close();

// ifs.open("label\_2000a.txt");

// for(int i = 0; i < 500; ++i) {

// ifs >> testLabels[i];

// }

// ifs.close();

// InnerProduct fc1(nImgRows \* nImgCols, 500), fc2(500, 10);

InnerProduct fc(nImgRows \* nImgCols, 10);

ReLU relu(500);

SoftmaxWithLoss loss(10);

for(int i = 0; i < 10000 / trainImgs.size(); ++i) {//max\_iter

for(int j = 0; j < trainImgs.size(); ++j) {//max\_iter

if (!(j % 2000)) {//stepsize

base\_lr \*= 0.1;//gammar

momentum += 0.05;

}

/\* fc1.forward\_pass(trainImgs[j]);

relu.forward\_pass(fc1.top);

fc2.forward\_pass(relu.top);\*/

fc.forward\_pass(trainImgs[j]);

loss.forward\_pass(fc.top);

loss.backward\_pass(trainLabels[j]);

/\* fc2.backward\_pass(loss.grads, relu.top, base\_lr);

relu.backward\_pass(fc2.grads);

fc1.backward\_pass(relu.grads, trainImgs[j], base\_lr);\*/

fc.backward\_pass(loss.grads, trainImgs[j], momentum,base\_lr);

}

}

for(int k = 0; k < testImgs.size(); ++k) {

/\* fc1.forward\_pass(testImgs[k]);

relu.forward\_pass(fc1.top);

fc2.forward\_pass(relu.top);

loss.forward\_pass(fc2.top);\*/

fc.forward\_pass(testImgs[k]);

loss.forward\_pass(fc.top);

std::cout << std::max\_element(loss.top.begin(), loss.top.end()) - loss.top.begin() << std::endl;

//std::max\_element(loss.top.begin(), loss.top.end()) - loss.top.begin() == testLabels[k] ? ++tp : ++fp;

}

system("pause");

return 0;

}

**（二）人脸识别实践**

主要应用了Adrian Bulat所做的一个face alignment模型进行了代码复现，体会了一下CNN等网络的架构（他的网络最近在最近取得了很好的成绩）。作者的论文中用了2D和2Dto3D两个模型。其核心结构是resNet残差神经网络

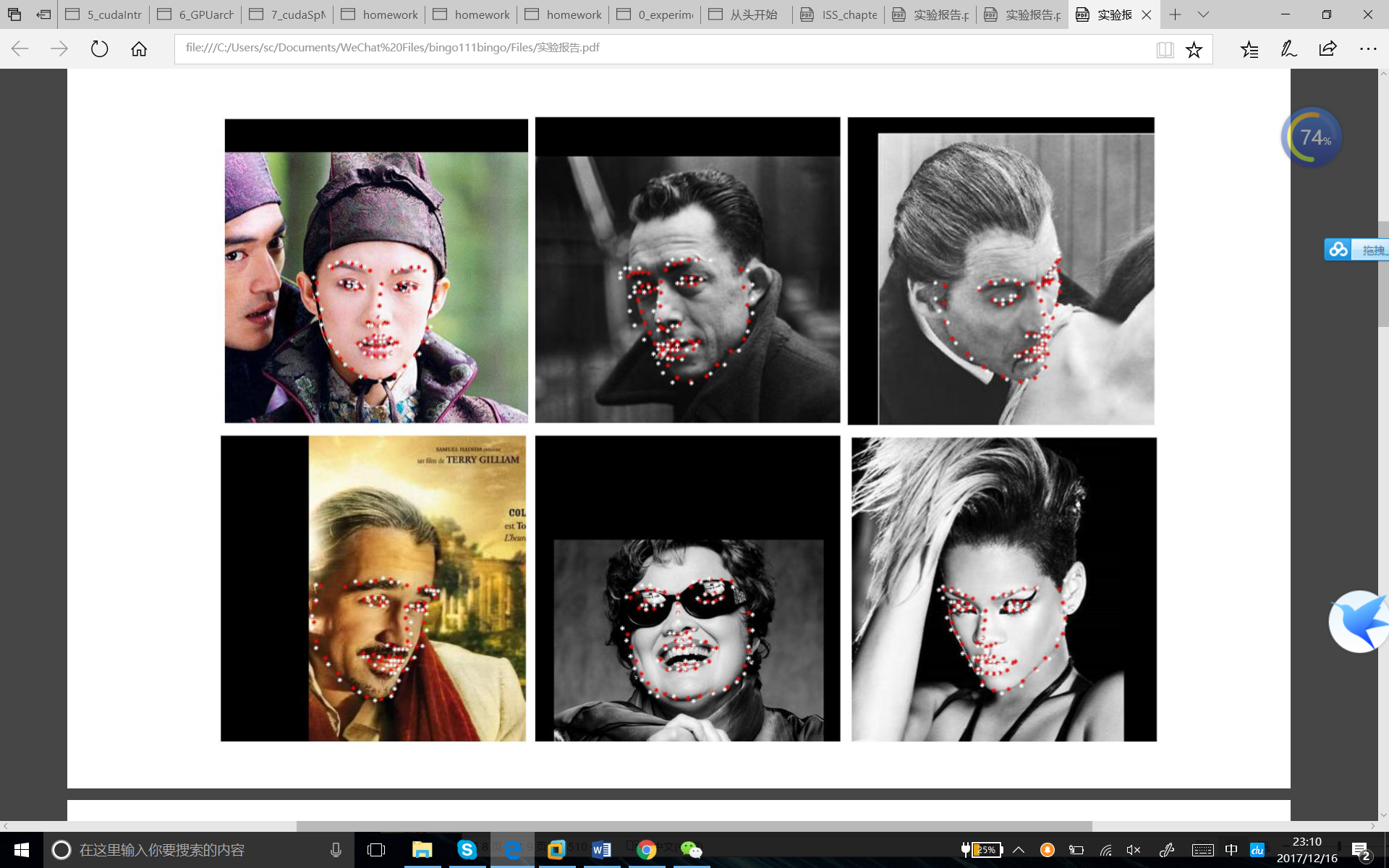
以目前最先进的技术MDM在LFPW测试集上的表现为baseline，作者测试了了2D-FAN的表现。由于近几年年在LFPW测试集上算法表现没有更好的进步，因此作者认为MDM基本达到了识别的饱和，所以作者将其认为是ground truth，并计算并画出2D-FAN识别结果与ground truth的差异，以判断2D-FAN方法的表现。

- 数据集：LFPW

- 部分测试结果：

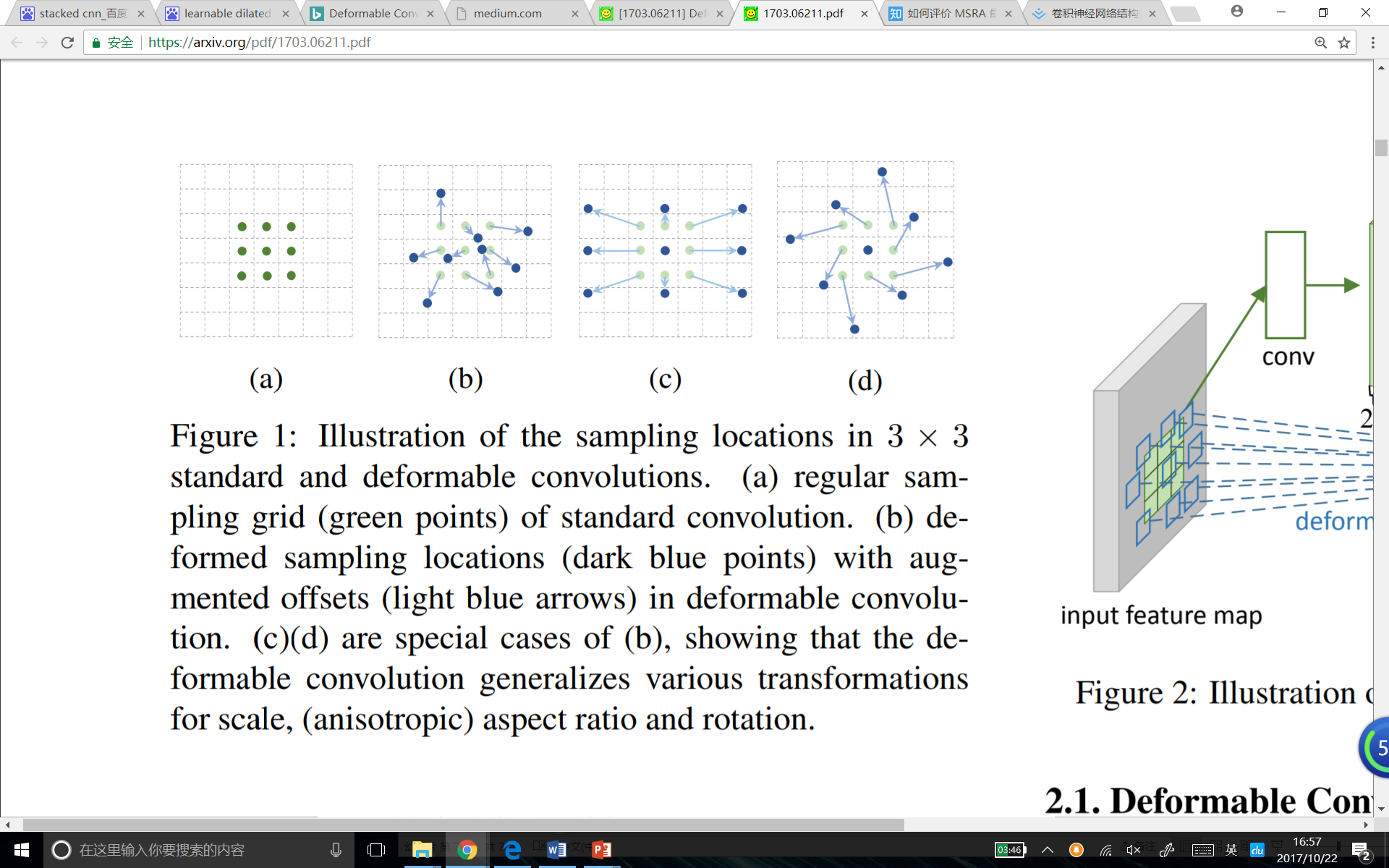
在重现实验结果时，使用作者在GitHub上提供的使用方法，调用 face\_alignment.FaceAlignment(face\_alignment.LandmarksType.\_2D, enable\_cuda=False, flip\_input=False)，进行2D图片识别，不使用cuda，获得面部68 个点的二维坐标值（68\*2的点坐标，需要转置为2\*68矩阵，方便与ground truth计算 NME），将之与MDM在LFPW测试集上获得的ground truth（2\*68矩阵）对比，画在图片上获得更好的观察效果。

下面展示部分画出的LFPW数据集中2D-FAN识别结果与ground truth。



**（三）课外学习**

最近看了2017年3月微软亚研院刚发的论文Deformable Convolutional Networks,首次在卷积神经网络中引入了学习空间几何形变的能力，得到可变形卷积网络，从而更好地解决了具有空间形变的图像识别任务。首次表明在深度卷积神经网络（deep CNN）中学习空间上密集的几何形变是可行的。



该论文中提出的CNN中的dilation不一定是认为设置的固定值，而是可变的，且其变化的偏移量offset是一个根据情况变化的量，可以进行学习，从而更好地适应形变。我觉得这个想法令我很豁然开朗。

事实上，可变形卷积单元中增加的偏移量是网络结构的一部分，通过另外一个平行的标准卷积单元计算得到，进而也可以通过梯度反向传播进行端到端的学习。加上该偏移量的学习之后，可变形卷积核的大小和位置可以根据当前需要识别的图像内容进行动态调整，其直观效果就是不同位置的卷积核采样点位置会根据图像内容发生自适应的变化，从而适应不同物体的形状、大小等几何形变，如图2、图3中所展示。

**三、课程感悟**

1）在刚讲深度学习各层网络的原理时，就自己尝试按照原理写了各层的实现，去组装自己的网络，但在过程中确实遇到了很多障碍需要一点点解决，例如反向传播时，对loss function对梯度需要事先头脑清晰地列一遍公式，实现的过程中才不会糊涂。而后面降到tenserflow框架可以自动完成求梯度部分，觉得很实用。所以善用深度学习框架是提高工程和科研效率的重要方法。

2）网络并不是越深越好的，反而不合理搭建的过深网络还会被梯度爆炸和梯度消失所困扰。在搭建网络时，借鉴生物学上的一些原理很重要。在理清了数据依赖性，变化方式后再搭建网络效果会好很多。例如：什么时候需要记忆网络，什么时候需要快速遗忘；什么时候需要短期记忆网络，什么时候又需要长期记忆配合shortcut。

3)具体问题一定要具体分析，且根据实验结果进行结构调整和finetune很重要。例如：图像处理上卷积神经网络最为适用，是因为图像的性质上具有高度的局部相关性，周边像素点之间有相互影响。而后续根据训练和预测结果，需要考虑感受野是否够大，是否需要调整dilation,又或者是否需要改变stride来提高训练效率等等。