

# 浙江大学

## 本科周报告

姓 名:

赖梓林

学 院:

计算机科学与技术

系:

计算机科学与技术

专 业:

计算机科学与技术

学 号:

3170104684

# 目录

第一章 摘要 .....	3
第二章 正文 .....	4
<b>1.Pytorch 部分.....</b>	<b>4</b>
1.1 环境配置 .....	4
1.2 数据集准备.....	6
1.3 代码部分 .....	7
1.4 实际训练 .....	9
1.5 细节部分 .....	9
<b>2.Topaz 部分 .....</b>	<b>10</b>
2.1 简述 .....	10
2.2 教程部分 .....	10
2.3 整体流程 .....	10
2.4 去噪部分 .....	12
2.5 其他 .....	13
<b>3.参考资料.....</b>	<b>14</b>

## 摘要

报告记录了本周对于 Pytorch 简单实践的相关内容，简述了 Topaz 基本流程步骤。

## 二、正文：

### 1. Pytorch 实践部分：

在经过了几周的理论知识学习后，这一周我们开始进行针对 Pytorch 的实践练习。在开始投入实践前，我们首先将整个实践练习过程分为几个步骤，分别是环境配置、数据集准备、概念明确、代码、实际训练和小结。

在这里简要叙述每个部分的内容：

首先**环境配置**列出了实践过程中所用到的部件版本等环境信息。

**数据集准备**部分说明这次实践中使用的数据集和相关信息。

**代码**部分大致给出用于实践的代码，和部分说明。

**细节**部分写出了一些在实践过程中注意到的细节。

**实际训练**部分给出最后的结果。

#### 1.1. 环境配置：

在定义中，配置管理是一个系统工程过程，用于建立和维护产品性能，功能和物理属性在其整个生命周期中的需求，设计和操作信息的一致性。

在我们这个小型课设中，所用到的环境配置相关信息如下：

Windows 10 (64-bit)
Python 3.7
Anaconda 3 (64-bit)

在环境配置过程中，我原本已在 Windows 系统中安装有 Python 和 pip，但并未安装 Anaconda，而在 Topaz 相关内容中看到项目中推荐使用 Anaconda，所以在 Pytorch 实践过程中环境配置的目标是在 Windows 10 系统上成功安装 Anaconda，配置虚拟环境并导入相关包。最后使用 Jupyter notebook 完成简单实践过程。

但在实际过程中，我发现配置步骤略为复杂，而且在以往都是使用 pip，所以在配置中出现了许多问题，在下面列出：

1. 未理解 conda activate 的目的和作用。
2. 导入包时发生错误显示找不到模块。

3. 在 conda 终端中安装好包之后使用 Pycharm 时找不到模块。
4. 在安装好相关库的环境中打开 jupyter notebook 找不到模块。

在经过数次尝试后，终于完成了环境配置，步骤如下：

下载 Anaconda 3 安装包并安装，

在 conda 终端中安装 jupyter notebook 和 pytorch 等相关包，

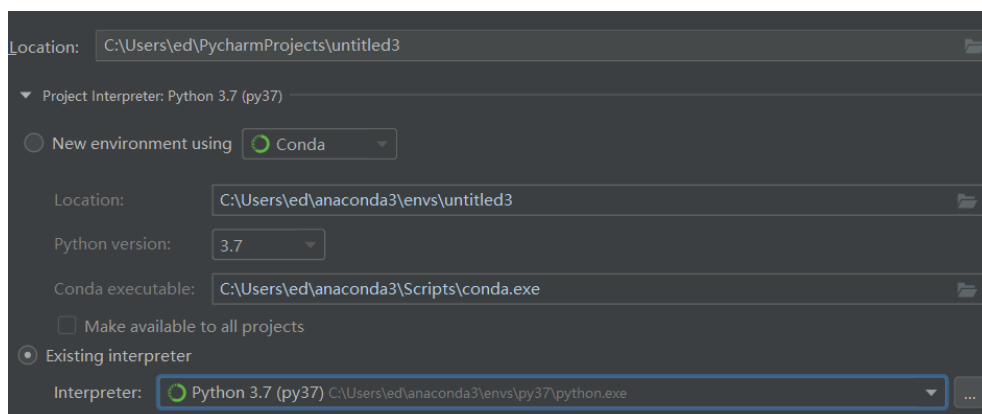
输入 `conda create` 建立一个虚拟环境，

输入 `conda activate` 启用虚拟环境。

输入 `python` 并通过 `import` 命令尝试导入。

如无报错则安装完成，

在使用 Pycharm 时，建立新项目步骤中，将使用环境改为 Conda，解释器改为启用的虚拟环境（如图中，已启用名为 `py37` 的虚拟环境）。



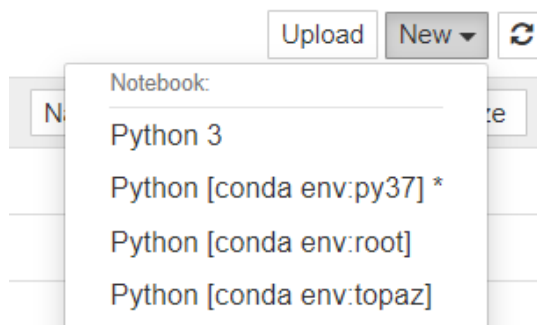
在使用 Jupyter Notebook 时，需额外操作：

输入 `conda install nb_conda` 以及 `conda install ipykernel`，

安装完成后，输入 `conda activate py37` 进入建立好的虚拟环境，

输入 `jupyter notebook` 打开 jupyter notebook 网页，

在新建文件时选择虚拟环境即可（如图中 `py37` 等）。



在摸索并成功配置环境后，之前的几个问题也得到了解答。

1. 未理解 conda activate 的目的和作用。

这个命令的作用是激活、启用一个虚拟环境，虚拟环境之间互相独立。

2. 导入包时发生错误显示找不到模块。

根据使用的平台不同，要将安装好相关包的虚拟环境部署到平台上。

另外需要注意原本已安装好的 Python 的版本。

3. 在 conda 终端中安装好包之后使用 Pycharm 时找不到模块。

Pycharm 中在建立新项目时需将使用环境更改为 conda，解释器更改为安装好包的虚拟环境。

4. 在安装好相关库的环境中打开 jupyter notebook 找不到模块。

需额外安装两个包，并在新建文件时选择虚拟环境。

环境配置结束后，就可以进行下一个步骤。

## 1.2. 数据集准备：

由于实践部分没有指定任务，所以我决定简单实现一个卷积神经网络分类器，在确定了目标之后，便可以开始着手数据集的准备。由于自行建立一个数据集不太现实，这次实践我采用网络中提供的适用于网络训练的数据集。

我首先想到被称为深度学习领域的“Hello World!”的 MNIST 数据集，MNIST 数据集是一个由手写体的 0-9 数字图像组成的数据集。也是评估模型好坏的一个简单标准。

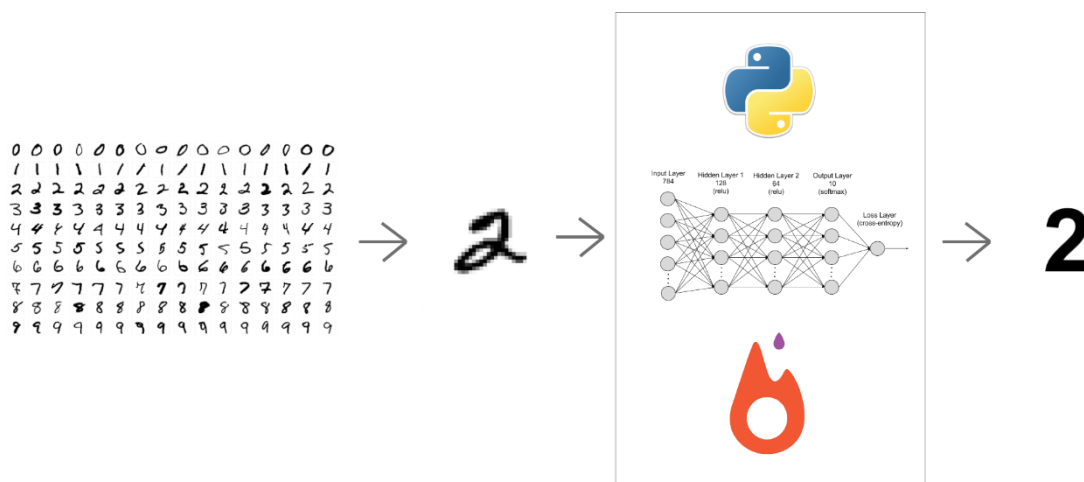


图 MNIST 数据集

在搜索的过程中，我想起看过的课程中有提到类似 MNIST 数据集的 Fashion-MNIST 数据集，于是最后决定使用 Fashion-MNIST 作为这次实践的数据集。

Fashion-MNIST 是由名为 Zalando 的时尚杂志提供的数据集，这个数据集中有一万张图片作为测试集，六万张图片作为训练集。每张图片具有相同的格式，是大小为 28x28 的灰度图片。图片内容皆取自杂志上的不同时装衣物，并经过处理打上标签，最后共分为十个类别。

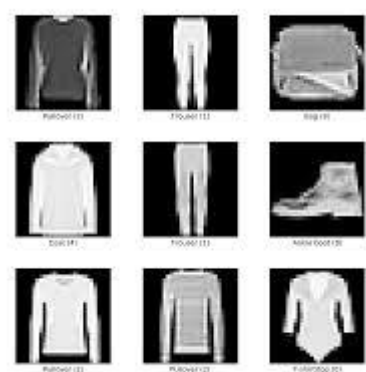


图 Fashion-MNIST

Fashion-MNIST 十分适合我们的简单任务，首先图像大小较小仅 28x28，并且是灰度图像，只有单通道，这大幅减少了我们的训练过程耗时。另外数据集可以通过 torchvision 包直接调用，并且已打好标签，我们可以利用这几点快速进入下一个步骤。

### 1.3. 代码部分：

和 TensorFlow 相比，Pytorch 代码更接近 Python 语言。

在确定了数据集后，首先，我们需要导入相关的包。（最后使用 jupyter 完成这次内容）

随后，我们读取数据集，本次使用的数据集 Fashion-MNIST 可以直接通过 torchvision 来下载、调用。

数据集读取完成后，我们首先要构建网络。在 Pytorch 中，网络的构建就是编写一个网络的类，并且这个类要继承 nn.Module，在 \_\_init\_\_ 方法中按层

将网络架构起来，并根据这个架构编写 forward 方法，在 forward 方法中添加 ReLu 和池化层等（或直接在构建时采用序列式模型将这两层包含进去）。

```
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5)
        self.fc1 = nn.Linear(in_features=12*4*4, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=60)
        self.out = nn.Linear(in_features=60, out_features=10)
```

上面的步骤完成后，就可以进行训练了。

```
network = Network()
loader = torch.utils.data.DataLoader(train_set, batch_size = 1000)
optimizer = optim.Adam(network.parameters(), lr=0.01)
for epoch in range(epochs):
    for batch in loader:
        images = batch[0]
        labels = batch[1]
        preds = network(images)
        loss = F.cross_entropy(preds, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

训练步骤中，我们首先实例化网络，使用 dataloader 读取数据集，并设定优化器，其中超参数部分如 epoch, batch size 和 learning rate 等可以根据第一次训练的结果进行对应修改。

随后我们便可以在 epoch 和 batch 的循环中，进行预测、计算损失函数、梯度归零、反向传播，完成整个训练过程。



#### 1.4. 实际训练:

在下图中可以看到简单进行 epoch 为 16, batch size 为 1000, learning rate 为 0.01 的训练结果。

```
print( loss: , loss.item(), acc: , acc, )  
  
loss: 0.618497371673584 acc: 0.759  
loss: 0.46339476108551025 acc: 0.812  
loss: 0.4071740210056305 acc: 0.839  
loss: 0.3850158154964447 acc: 0.852  
loss: 0.34838515520095825 acc: 0.865  
loss: 0.32700657844543457 acc: 0.865  
loss: 0.33583080768585205 acc: 0.865  
loss: 0.32542455196380615 acc: 0.871  
loss: 0.33333295583724976 acc: 0.865  
loss: 0.3043741285800934 acc: 0.878  
loss: 0.30258849263191223 acc: 0.874  
loss: 0.27586326003074646 acc: 0.888  
loss: 0.27015167474746704 acc: 0.892  
loss: 0.26695719361305237 acc: 0.894  
loss: 0.273512601852417 acc: 0.894  
loss: 0.26398423314094543 acc: 0.898
```

从训练结果可以看出, 模型对 Fashion-MNIST 的训练效果良好, 并且效率极高, 耗时很短。损失逐渐下降, 准确率上升至接近 90%。

#### 1.5. 细节部分:

这个简单的实践过程中, 我发现仍有许多细节部分可以改进, 并且在网络课程中也有提到许多改进的点例如:

可以引入数据增强的手段等, 让网络更好地识别特征。

可以将涉及到的超参数打包在一起, 便于修改。

可以添加一个 RunManager 类, 让我们能更轻松地改变训练计划。

可以使用 TensorBoard, 让我们更直观地评估模型最后的表现。

可以编写代码实现模型的读取和保存, 但在这个简单任务中这样做意义不大。

## 2. Topaz 部分:

### 2.1 简述:

在经过了几周的理论知识学习后,这一周我们同时也可以开始尝试使用 Topaz 并理解源码。首先我们从 github 的 Topaz 项目中的 Tutorial 入手。

Topaz Pipeline 大约分为以下几个步骤:

0 可选预处理)

运行 Topaz 前,使用其他软件在显微图像上标记少量颗粒

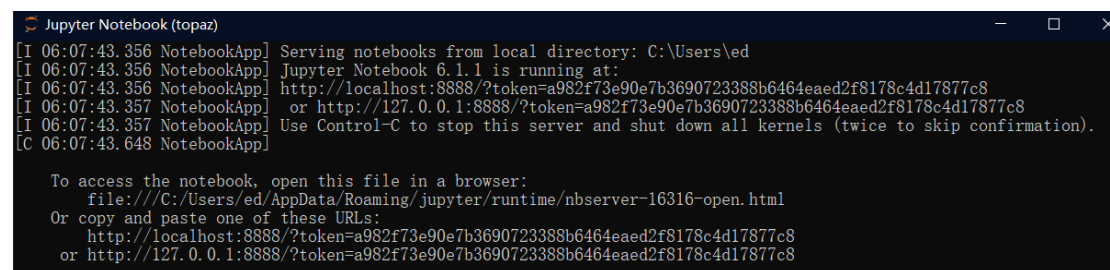
- 1) 对显微图像下采样、标准化,标记好的颗粒坐标准确记录。
- 2) 使用预处理显微图像中标记好的颗粒坐标训练颗粒检测模型。
- 3) 使用训练好的模型提取显微图像中的颗粒坐标以及关联分数。

4 可选后处理)

测试分类器的表现,重新测定颗粒坐标等。

### 2.2. Tutorial 部分:

打开项目中的 Tutorial 部分,我们能看到作者提供了四个 ipynb 格式的文件,经过之前 Pytorch 实践部分,我们已经预先配置好了环境,此时只要打开 Topaz 虚拟环境的 notebook 即可。



```
Jupyter Notebook (topaz)
[I 06:07:43.356 NotebookApp] Serving notebooks from local directory: C:\Users\ed
[I 06:07:43.356 NotebookApp] Jupyter Notebook 6.1.1 is running at:
[I 06:07:43.356 NotebookApp] http://localhost:8888/?token=a982f73e90e7b3690723388b6464eae2f8178c4d17877c8
[I 06:07:43.357 NotebookApp] or http://127.0.0.1:8888/?token=a982f73e90e7b3690723388b6464eae2f8178c4d17877c8
[I 06:07:43.357 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 06:07:43.648 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/ed/AppData/Roaming/jupyter/runtime/nbserver-16316-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=a982f73e90e7b3690723388b6464eae2f8178c4d17877c8
    or http://127.0.0.1:8888/?token=a982f73e90e7b3690723388b6464eae2f8178c4d17877c8
```

同时我们根据作者要求,下载范例数据集并解压缩至根目录。

我们主要先理解 walkthrough 文件,以此来感受整个过程。

### 2.3. 整体流程:

Walkthrough 部分对之前提到的标准流程的几个步骤进行了进一步的展开和解释,并通过剩余两个教程文件补充了对整个流程的讲解。这里完整地叙述相关流程内容。

- 1) 对显微图像下采样、标准化。

操作者首先将完整的显微图像进行八倍的下采样，随后使用高斯混合模型对它进行归一化。作者提到可以用 preprocess 命令或是 downsample 加上 normalize 命令完成这一步。

另外，作者提到下采样的倍数取决于使用者要解决的实际问题，八倍不一定是使用者的数据的最佳下采样倍数。作者建议，要调整下采样的倍数至适合使用者使用的卷积神经网络体系结构的接受域。且教程中使用 Resnet8。

## 2) 缩放颗粒的坐标来匹配下采样后的显微图像。

教程有提供数据集下载，其中有文本文档文件标注颗粒的坐标。

使用 convert 可以缩放颗粒坐标以匹配下采样的显微图像。

```
image_name      x_coord y_coord
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      866      310
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      632      697
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      366      190
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      350      448
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      470      817
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      567      147
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      530      102
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      867      617
14sep05c_c_00003gr_00014sq_00004hl_00004es_c      584      608
```

## 3) 训练模型

Topaz 的核心是训练一个可以通过 PU 学习探测到颗粒的卷积神经网络，通过训练好的分类器，Topaz 判断显微图像上的哪个区域包含我们想要的颗粒。完成上面两步后，我们就可以从预测到的包含有一些颗粒的区域提取颗粒。

所以模型的训练尤为重要，我们必须区分哪些显微图像和标记颗粒坐标是我们想要用来训练模型的。训练完成后我们也需要用没有训练过的数据来检测模型是否表现良好，并根据实际情况调整超参数。

## 4) 预训练颗粒挑选模型

Topaz 包含了一些预训练的颗粒挑选模型。这些模型已经通过大量多样化数据的训练，无需额外训练就可以运用在不同的数据集上，但表现不一定是最好的。有 32 或 64 个 unit 的 resnet8 或 resnet16 的神经网络结构可以使用预训练模型。

## 2.4. 去噪部分：

正如我们之前的报告中提到的，颗粒图像的去噪提升了 2D 图像的质量，从而提升了 2D 平均后的质量，最终获得更好的 3D 模型。

Topaz 去噪提供了几种不同结构的预处理模型，包括小型和两种完整型的 U-Net、全卷积神经网络模型以及一种简单线性去噪模型。具体哪一种模型的使用取决于数据集，这一点在我们之前的报告中有提到。

```
name = '14sep05c_c_00007gr_00013sq_00009hl_00002es_c'

# load the raw micrograph
mic_raw = np.array(load_image('data/EMPIAR-10025/rawdata/micrographs/' + name + '.mrc'), copy=False)
# load the denoised micrograph
mic_dn = np.array(load_image('data/EMPIAR-10025/denoised/' + name + '.mrc'), copy=False)

# scale them for visualization
mu = mic_dn.mean()
std = mic_dn.std()

mic_raw = (mic_raw - mu)/std
mic_dn = (mic_dn - mu)/std

_, ax = plt.subplots(1, 2, figsize=(24, 12))

ax[0].imshow(mic_raw, vmin=-4, vmax=4, cmap='Greys_r')
ax[1].imshow(mic_dn, vmin=-4, vmax=4, cmap='Greys_r')
```

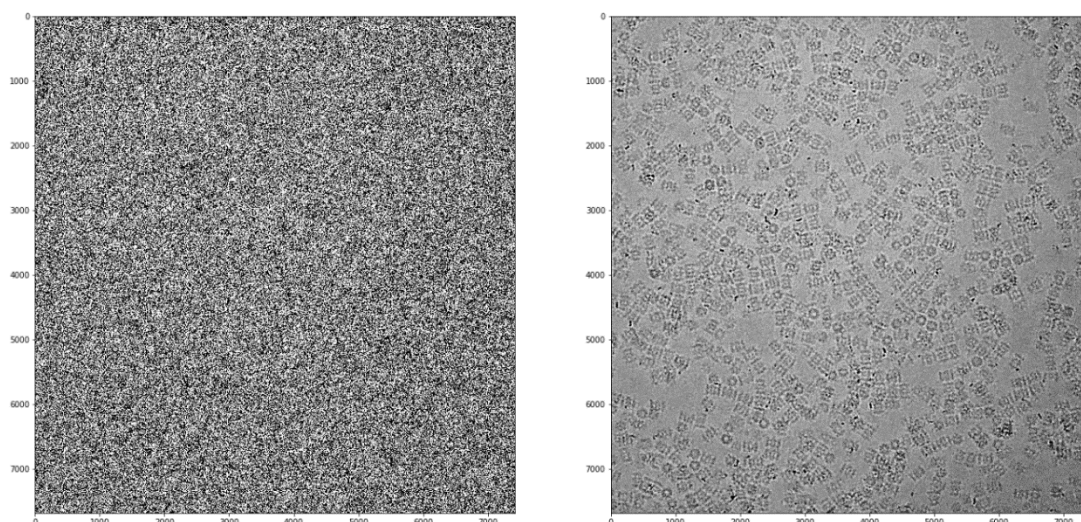


图 去噪效果可视化

从上面的去噪图片和原始图片的对比图我们可以看出，图像的整体质量有了很大的提升。显微图像中可以确定的颗粒数目肉眼可见地增多。

如果使用者需要训练新的去噪模型，需要提供给 Topaz 同样分布信号中的成对独立样本，这是因为 Topaz 是基于 Noise2Noise 框架的，是从噪声中学习去噪模型，有关 Noise2Noise 的内容已在上周提及。

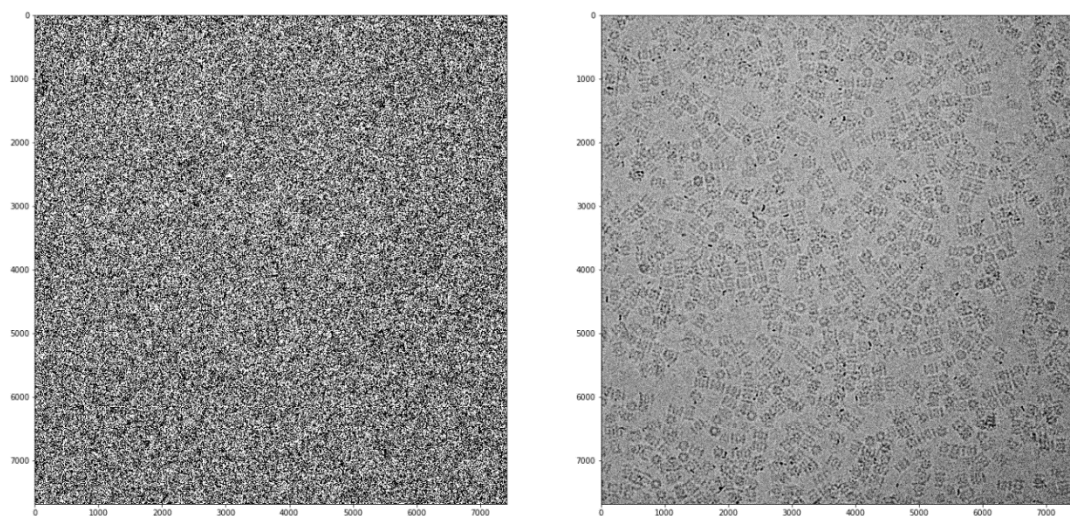


图 （训练后的模型）去噪效果可视化

## 2.5. 其他：

对于 Topaz Pipeline 中的区域分类器，作者提出了几种可用的网络结构，例如接受域为 71 的 Resnet8，接受域为 31、63、127 的卷积神经网络。

其中 Resnet8 良好地平衡了最终表现和接受域大小，而当我们更需要更简单一些的模型时，就可以使用接受域为 31、63 的卷积神经网络。

### 3. 参考资料:

1. CNN Training Loop Refactoring - Simultaneous Hyperparameter Testing
2. Pytorch Explained
3. Pytorch install - Quick and Easy
4. Tensors for Deep Learning - Broadcasting and Element-wise
5. CNN Image Preparation Code Project - Learn to Extract, Transform...
6. Build Pytorch CNN - Object Oriented Neural Networks

[https://www.youtube.com/watch?v=ozpv\\_peZ894&list=PLZbbT5o\\_s2xrfNyHZsM6ufI0iZENK9xgG&index=33](https://www.youtube.com/watch?v=ozpv_peZ894&list=PLZbbT5o_s2xrfNyHZsM6ufI0iZENK9xgG&index=33)

(网址同系列)

7. Topaz - tbepler 项目
  8. Topaz - Bepler, T., Morin, A., Brasch, J., Shapiro, L., Noble, A.J., Berger, B. (2019). Positive-unlabeled convolutional neural networks for particle picking in cryo-electron micrographs. Nature Methods. <https://doi.org/10.1038/s41592-019-0575-8>
  9. Topaz-Denoise Bepler, T., Kelley, K., Noble, A.J., Berger, B. (2020). Topaz-Denoise: general deep denoising models for cryoEM and cryoET. bioRxiv. <https://doi.org/10.1101/838920>
- 以及学姐提供的相关资料和论文。