Load libraries

In [25]:
```
!git clone https://github.com/recursionpharma/rxrx1-utils.git && mv rxrx
1-utils rxrxutils
```

```
Cloning into 'rxrx1-utils'...
remote: Enumerating objects: 118, done.
remote: Total 118 (delta 0), reused 0 (delta 0), pack-reused 118
Receiving objects: 100% (118/118), 1.59 MiB | 0 bytes/s, done.
Resolving deltas: 100% (59/59), done.
mv: cannot move 'rxrx1-utils' to 'rxrxutils/rxrx1-utils': Directory not
empty
```

In [26]:
```
import sys
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import rxrxutils.rxrx.io as rio
from scipy import misc

from PIL import Image

import torch
import torch.nn as nn
import torch.utils.data as D
from torch.optim.lr_scheduler import ExponentialLR
import torch.nn.functional as F

from torchvision import models, transforms

from ignite.engine import Events, create_supervised_evaluator, create_su
pervised_trainer
from ignite.metrics import Loss, Accuracy
from ignite.contrib.handlers.tqdm_logger import ProgressBar
from ignite.handlers import  EarlyStopping, ModelCheckpoint

from tqdm import tqdm_notebook

from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [27]:
```
!ls -1 ../input
```

```
pixel_stats.csv
recursion_dataset_license.pdf
sample_submission.csv
test
test.csv
test_controls.csv
train
train.csv
train_controls.csv
```

Define dataset and model

In [28]:
```
path_data = '../input'
device = 'cuda'
batch_size = 32
torch.manual_seed(0)
```

Out[28]:
```
<torch._C.Generator at 0x7f237906cc50>
```

In [29]:
```
class ImagesDS(D.Dataset):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    def __init__(self, df, mode='train', site=1, channels=[1,2,3,4,5,6
]):
        self.records = df.to_records(index=False)
        self.channels = channels
        self.site = site
        self.mode = mode
        self.len = df.shape[0]
        self.first = None

    def _get_img(self, index):
        record = self.records[index]
        return transforms.ToTensor()(rio.load_site(self.mode, record.exp
eriment, record.plate, record.well, self.site, base_path=path_data))

    def __getitem__(self, index):
        img = self._get_img(index)
        if self.mode == 'train':
            return img, int(self.records[index].sirna)
        else:
            return img, self.records[index].id_code

    def __len__(self):
        return self.len
```

In [30]:
```
# dataframes for training, cross-validation, and testing
df = pd.read_csv(path_data+'/train.csv')
df_train, df_val = train_test_split(df, test_size = 0.025, random_state=
42)
df_test = pd.read_csv(path_data+'/test.csv')

# pytorch training dataset & loader
ds = ImagesDS(df_train, mode='train')
loader = D.DataLoader(ds, batch_size=batch_size, shuffle=True, num_worke
rs=4)

# pytorch cross-validation dataset & loader
ds_val = ImagesDS(df_val, mode='train')
val_loader = D.DataLoader(ds_val, batch_size=batch_size, shuffle=True, n
um_workers=4)

# pytorch test dataset & loader
ds_test = ImagesDS(df_test, mode='test')
tloader = D.DataLoader(ds_test, batch_size=batch_size, shuffle=False, nu
m_workers=4)
```

In [31]:

```
classes = 1108

def create_model_from_resnet50():
    model = models.resnet50(pretrained=True)
    num_ftrs = model.fc.in_features
    model.fc = torch.nn.Linear(num_ftrs, classes)

    # let's make our model work with 6 channels
    trained_kernel = model.conv1.weight
    new_conv = nn.Conv2d(6, 64, kernel_size=7, stride=2, padding=3, bias
=False)
    with torch.no_grad():
        new_conv.weight[:,:] = torch.stack([torch.mean(trained_kernel, 1
)]*6, dim=1)
    model.conv1 = new_conv

    return model

model = create_model_from_resnet50()

print(model)
```

```
ResNet(
  (conv1): Conv2d(6, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3,
3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=F
alse)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
```

```
ck_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=F
alse)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
```

```
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias
=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=
False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
```

```
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer4): Sequential(
```

```
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias
=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1108, bias=True)
)
```

In [32]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

In [33]:
```
metrics = {
    'loss': Loss(criterion),
    'accuracy': Accuracy(),
}

trainer = create_supervised_trainer(model, optimizer, criterion, device=
device)
val_evaluator = create_supervised_evaluator(model, metrics=metrics, devi
ce=device)
```

In [34]:
```
@trainer.on(Events.EPOCH_COMPLETED)
def compute_and_display_val_metrics(engine):
    epoch = engine.state.epoch
    metrics = val_evaluator.run(val_loader).metrics
    print("Validation Results - Epoch: {}  Average Loss: {:.4f} | Accura
cy: {:.4f} "
          .format(engine.state.epoch,
                      metrics['loss'],
                      metrics['accuracy']))
```

In [35]:
```
lr_scheduler = ExponentialLR(optimizer, gamma=0.95)

@trainer.on(Events.EPOCH_COMPLETED)
def update_lr_scheduler(engine):
    lr_scheduler.step()
    lr = float(optimizer.param_groups[0]['lr'])
    print("Learning rate: {}".format(lr))
```

In [36]:
```
@trainer.on(Events.EPOCH_STARTED)
def turn_on_layers(engine):
    epoch = engine.state.epoch
    if epoch == 1:
        for name, child in model.named_children():
            if name == 'fc':
                pbar.log_message(name + ' is unfrozen')
                for param in child.parameters():
                    param.requires_grad = True
            else:
                pbar.log_message(name + ' is frozen')
                for param in child.parameters():
                    param.requires_grad = False
    if epoch == 3:
        pbar.log_message("Turn on all the layers")
        for name, child in model.named_children():
            for param in child.parameters():
                param.requires_grad = True
```

In [37]:
```
handler = EarlyStopping(patience=6, score_function=lambda engine: engine
.state.metrics['accuracy'], trainer=trainer)
val_evaluator.add_event_handler(Events.COMPLETED, handler)
```

In [38]:
```
checkpoints = ModelCheckpoint('models', 'Model', save_interval=2, n_save
d=5, create_dir=True)
trainer.add_event_handler(Events.EPOCH_COMPLETED, checkpoints, {'ResNet5
0': model})
```

In [39]:
```
pbar = ProgressBar(bar_format='')
pbar.attach(trainer, output_transform=lambda x: {'loss': x})
```

In [40]:
```
trainer.run(loader, max_epochs=15)
```

```
conv1 is frozen
bn1 is frozen
relu is frozen
maxpool is frozen
layer1 is frozen
layer2 is frozen
layer3 is frozen
layer4 is frozen
avgpool is frozen
fc is unfrozen


Validation Results - Epoch: 1  Average Loss: 7.0860 | Accuracy: 0.0044
Learning rate: 0.000285


Validation Results - Epoch: 2  Average Loss: 6.9459 | Accuracy: 0.0120
Learning rate: 0.00027075
Turn on all the layers


Validation Results - Epoch: 3  Average Loss: 5.5644 | Accuracy: 0.0657
Learning rate: 0.0002572125


Validation Results - Epoch: 4  Average Loss: 5.3126 | Accuracy: 0.0997
Learning rate: 0.000244351875


Validation Results - Epoch: 5  Average Loss: 4.3463 | Accuracy: 0.1884
Learning rate: 0.00023213428124999998


Validation Results - Epoch: 6  Average Loss: 4.1457 | Accuracy: 0.2202
Learning rate: 0.00022052756718749997


Validation Results - Epoch: 7  Average Loss: 4.5430 | Accuracy: 0.1993
Learning rate: 0.00020950118882812497


Validation Results - Epoch: 8  Average Loss: 4.2016 | Accuracy: 0.2355
Learning rate: 0.00019902612938671872


Validation Results - Epoch: 9  Average Loss: 4.1894 | Accuracy: 0.2563
Learning rate: 0.00018907482291738277


Validation Results - Epoch: 10  Average Loss: 4.2727 | Accuracy: 0.2782
Learning rate: 0.00017961961081771513


Validation Results - Epoch: 11  Average Loss: 4.7170 | Accuracy: 0.2596
Learning rate: 0.00017064002768293793


Validation Results - Epoch: 12  Average Loss: 4.7693 | Accuracy: 0.2552
Learning rate: 0.00016210802629879103
```

```
Validation Results - Epoch: 13  Average Loss: 4.9412 | Accuracy: 0.2640
Learning rate: 0.00015400262498385146
```

```
Validation Results - Epoch: 14  Average Loss: 5.2648 | Accuracy: 0.2640
Learning rate: 0.00014630249373465888
```

```
Validation Results - Epoch: 15  Average Loss: 4.9992 | Accuracy: 0.2683
Learning rate: 0.00013898736904792593
```

Out[40]:
```
<ignite.engine.engine.State at 0x7f2331870b38>
```

In [41]:
```
model.eval()
with torch.no_grad():
    preds = np.empty(0)
    for x, _ in tqdm_notebook(tloader):
        x = x.to(device)
        output = model(x)
        idx = output.max(dim=-1)[1].cpu().numpy()
        preds = np.append(preds, idx, axis=0)
```

In [ ]:
```
submission = pd.read_csv(path_data + '/test.csv')
submission['sirna'] = preds.astype(int)
submission.to_csv('submission_firststep.csv', index=False, columns=['id_
code','sirna'])
```

In [44]:
```
model.load_state_dict(torch.load('models/Model_ResNet50_6.pth'))
model.eval()
with torch.no_grad():
    preds = np.empty(0)
    for x, _ in tqdm_notebook(tloader):
        x = x.to(device)
        output = model(x)
        idx = output.max(dim=-1)[1].cpu().numpy()
        preds = np.append(preds, idx, axis=0)

submission = pd.read_csv(path_data + '/test.csv')
submission['sirna'] = preds.astype(int)
submission.to_csv('submission_firststep_epoch6.csv', index=False, column
s=['id_code','sirna'])
```

In [45]:
```
model.load_state_dict(torch.load('models/Model_ResNet50_10.pth'))
model.eval()
with torch.no_grad():
    preds = np.empty(0)
    for x, _ in tqdm_notebook(tloader):
        x = x.to(device)
        output = model(x)
        idx = output.max(dim=-1)[1].cpu().numpy()
        preds = np.append(preds, idx, axis=0)

submission = pd.read_csv(path_data + '/test.csv')
submission['sirna'] = preds.astype(int)
submission.to_csv('submission_firststep_epoch10.csv', index=False, colum
ns=['id_code','sirna'])
```

Download submission file for one-step model

Download submission file for one-step model epoch 6

Download submission file for one-step model epoch 10

Download weights file for one-step model

Conclusion for the first step of Resnet50 model

This gives us a cross-validation score of `0.27` (`27%` accuracy), and a test score of 0.104 (`10.4%` accuracy). This score is way better than chance since we have 1108 classes. An accuracy reflecting chance would be 1/1108, which is equivalent to ~0.09% accuracy.

We will explore how we can improve on this score in a next kernel. Chances are that the data distribution of the cross-validation set is different than the test set in some ways and we may be overfitting.

Also, it seems that the lost function may not be the best indicator of our model performance. We obtained the minimum loss at the 6th epoch, but it resulted in a lower accuracy score compared to the 10th epoch: `8.4%` vs `10.4%`.