Load libraries

In [1]:
```
!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.
```

In [2]:
```
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 [3]:
```
!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 [4]:
```
path_data = '../input'
device = 'cuda'
batch_size = 32
torch.manual_seed(0)
```

Out[4]:
```
<torch._C.Generator at 0x7f1ab9828b30>
```

In [5]:
```
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 [6]:
```
# 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 [7]:

```
classes = 1108
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=Fal
se)
with torch.no_grad():
    new_conv.weight[:,:] = torch.stack([torch.mean(trained_kernel, 1)]*6
, dim=1)
model.conv1 = new_conv

print(model)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pt
h" to /tmp/.cache/torch/checkpoints/resnet50-19c8e357.pth
100%|██████████| 102502400/102502400 [00:00<00:00, 106210823.73it/s]
```

```
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 [8]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0006)
```

In [9]:
```
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 [10]:
```
@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 [11]:
```
lr_scheduler = ExponentialLR(optimizer, gamma=0.9)

@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 [12]:
```
@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 [13]:
```
handler = EarlyStopping(patience=6, score_function=lambda engine: engine
.state.metrics['accuracy'], trainer=trainer)
val_evaluator.add_event_handler(Events.COMPLETED, handler)
```

In [14]:
```
checkpoints = ModelCheckpoint('models', 'Model', save_interval=3, n_save
d=3, create_dir=True)
trainer.add_event_handler(Events.EPOCH_COMPLETED, checkpoints, {'ResNet5
0': model})
```

In [15]:
```
pbar = ProgressBar(bar_format='')
pbar.attach(trainer, output_transform=lambda x: {'loss': x})
```

In [ ]:
```
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
```

In [ ]:
```
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'])
```

Download submission file for one-step model

Download weights file for one-step model

Conclusion for the first step of Resnet50 model

This gives us a cross-validation score of `0.0011` (`.1%` accuracy), and a test score of 0.002 (`.2%` accuracy). This score is a bit 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.

Second-step, training on each cell line

In [ ]:

```
categories = df['category'].unique()
preds = np.empty(0)

for category in categories:
    # Retrieve desired category
    category_df = df[df['category'] == category]
    cat_test_df = df_test[test_df['category'] == category].copy()

    print('\n' + '=' * 40)
    print("CURRENT CATEGORY:", category)
    print('-' * 40)

    train_idx, val_idx = train_test_split(
        category_df.index,
        random_state=2019,
        test_size=0.15
    )

    # pytorch training dataset & loader
    ds = ImagesDS(df_train, mode='train')
    loader = D.DataLoader(ds, batch_size=batch_size, shuffle=True, num_w
orkers=0)

    # pytorch cross-validation dataset & loader
    ds_val = ImagesDS(df_val, mode='train')
    val_loader = D.DataLoader(ds_val, batch_size=batch_size, shuffle=Tru
e, num_workers=0)

    # pytorch test dataset & loader
    ds_test = ImagesDS(df_test, mode='test')
    tloader = D.DataLoader(ds_test, batch_size=batch_size, shuffle=False
, num_workers=0)

    # Restore previously trained model
    model.load_state_dict(torch.load('models/ADD_MODEL_STATE'))

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    metrics = {
    'loss': Loss(criterion),
    'accuracy': Accuracy(),
    }

    trainer = create_supervised_trainer(model, optimizer, criterion, dev
ice=device)
    val_evaluator = create_supervised_evaluator(model, metrics=metrics,
device=device)

    @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} | Ac
curacy: {:.4f} "
              .format(engine.state.epoch,
                      metrics['loss'],
                      metrics['accuracy']))
```

```
    lr_scheduler = ExponentialLR(optimizer, gamma=0.9)
    @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))

    checkpoints = ModelCheckpoint('models', 'Model', save_interval=5, n_
saved=3, create_dir=True)
    trainer.add_event_handler(Events.EPOCH_COMPLETED, checkpoints, {f'Re
sNet50_{category}': model})

    pbar = ProgressBar(bar_format='')
    pbar.attach(trainer, output_transform=lambda x: {'loss': x})

    trainer.run(loader, max_epochs=40)

    # Make prediction and add to output dataframe
    model.eval()
    with torch.no_grad():
        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_secondststep.csv', index=False, columns=[
'id_code','sirna'])
```

Download submission file for second-step model