Load libraries

```
In [1]: !git clone https://github.com/recursionpharma/rxrx1-utils.git && mv rxrx
        1-utils rxrxutils
        !git clone https://github.com/MichelML/ml-cellsignal.git && mv ml-cellsi
        gnal pre_models
```

```
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.
Cloning into 'ml-cellsignal'...
remote: Enumerating objects: 249, done.
remote: Total 249 (delta 0), reused 0 (delta 0), pack-reused 249
Receiving objects: 100% (249/249), 374.43 MiB | 46.28 MiB/s, done.
Resolving deltas: 100% (114/114), done.
```

```python
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 0x7f9106595b30>
```

```
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['category'] = df['experiment'].apply(lambda x: x.split('-')[0])
         df_train, df_val = train_test_split(df, test_size = 0.025, random_state=
         42)
         df_test = pd.read_csv(path_data+'/test.csv')
         df_test['category'] = df_test['experiment'].apply(lambda x: x.split('-')
         [0])
```

```
In [7]:  def create_model_from_resnet50():
             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
         =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()
```

Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pt
h" to /tmp/.cache/torch/checkpoints/resnet50-19c8e357.pth
100%|████████████| 102502400/102502400 [00:01<00:00, 83008526.31it/s]

Training on each cell line

```
In [8]: categories = df['category'].unique()
        category = categories[3]
        preds = np.empty(0)

        # Retrieve desired category
        category_df = df[df['category'] == category]
        cat_test_df = df_test[df_test['category'] == category].copy()

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

        cat_train_df, cat_val_df = train_test_split(
            category_df,
            random_state=2019,
            test_size=0.05
        )

        # pytorch training dataset & loader
```

```python
ds = ImagesDS(cat_train_df, mode='train')
loader = D.DataLoader(ds, batch_size=batch_size, shuffle=True, num_worke
rs=4)

# pytorch cross-validation dataset & loader
ds_val = ImagesDS(cat_val_df, 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(cat_test_df, mode='test')
tloader = D.DataLoader(ds_test, batch_size=batch_size, shuffle=False, nu
m_workers=4)

# Restore previously trained model
model.load_state_dict(torch.load('pre_models/models/Model_ResNet50_12.pt
h'))

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0004)
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)

@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']))

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))

@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:
```

```python
                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

checkpoints = ModelCheckpoint('models', 'Model', save_interval=2, n_saved=5, create_dir=True)
trainer.add_event_handler(Events.EPOCH_COMPLETED, checkpoints, {f'ResNet50_{category}': model})

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

```python
trainer.run(loader, max_epochs=15)
```

```
=======================================
CURRENT CATEGORY: U2OS
----------------------------------------
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: 1.2576 | Accuracy: 0.6886
Learning rate: 0.00038


Validation Results - Epoch: 2  Average Loss: 1.1072 | Accuracy: 0.6946
Learning rate: 0.000361
Turn on all the layers


Validation Results - Epoch: 3  Average Loss: 0.5908 | Accuracy: 0.8443
Learning rate: 0.00034294999999999996


Validation Results - Epoch: 4  Average Loss: 0.4574 | Accuracy: 0.9162
Learning rate: 0.00032580249999999994


Validation Results - Epoch: 5  Average Loss: 0.2991 | Accuracy: 0.9581
Learning rate: 0.0003095123749999999


Validation Results - Epoch: 6  Average Loss: 0.2737 | Accuracy: 0.9521
Learning rate: 0.0002940367562499999


Validation Results - Epoch: 7  Average Loss: 0.2551 | Accuracy: 0.9701
Learning rate: 0.0002793349184374999


Validation Results - Epoch: 8  Average Loss: 0.2427 | Accuracy: 0.9760
Learning rate: 0.00026536817251562487


Validation Results - Epoch: 9  Average Loss: 0.2546 | Accuracy: 0.9521
Learning rate: 0.0002520997638898436
```

```
Validation Results - Epoch: 10  Average Loss: 0.2606 | Accuracy: 0.9641
Learning rate: 0.00023949477569535144
```

```
Validation Results - Epoch: 11  Average Loss: 0.2472 | Accuracy: 0.9641
Learning rate: 0.00022752003691058385
```

```
Validation Results - Epoch: 12  Average Loss: 0.2441 | Accuracy: 0.9701
Learning rate: 0.00021614403506505466
```

```
Validation Results - Epoch: 13  Average Loss: 0.2476 | Accuracy: 0.9701
Learning rate: 0.00020533683331180191
```

```
Validation Results - Epoch: 14  Average Loss: 0.2553 | Accuracy: 0.9641
Learning rate: 0.00019506999916462118
```

```
Validation Results - Epoch: 15  Average Loss: 0.2500 | Accuracy: 0.9641
Learning rate: 0.00018531649206390122
```

Out[8]: `<ignite.engine.engine.State at 0x7f91031c0668>`

In [9]:
```python
# pytorch test dataset & loader
ds_test = ImagesDS(cat_test_df, mode='test')
tloader = D.DataLoader(ds_test, batch_size=batch_size, shuffle=False, num_workers=4)

model.load_state_dict(torch.load('models/Model_ResNet50_U2OS_8.pth'))
model.eval()
preds = np.empty(0)
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)

submission = pd.read_csv(path_data + '/test.csv')
submission = submission[submission['experiment'].str.contains(category)]
submission['sirna'] = preds.astype(int)
submission.to_csv('submission_U2OS.csv', index=False, columns=['id_code','sirna'])
```

Download submission file

Download weights file for U2OS model

In [ ]: