In [ ]:
```
# By Ritik Jalisatgi, ritik@ucla.edu, along with help from SpuCo quickstart code
# https://github.com/BigML-CS-UCLA/SpuCo/tree/master/quickstart
```

In [168]:
```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from spuco.datasets import SpuCoMNIST, SpuriousFeatureDifficulty
from spuco.datasets import SpuCoMNIST, SpuriousFeatureDifficulty
from spuco.datasets.group_labeled_dataset_wrapper import GroupLabeledDatasetWrapper
from spuco.evaluate import Evaluator
from spuco.group_inference import GeorgeInference
from spuco.models import model_factory
from spuco.robust_train.group_dro import GroupDRO
from spuco.robust_train.group_balance_batch_erm import GroupBalanceBatchERM
from spuco.robust_train import ERM
from spuco.utils import Trainer, set_seed
from spuco.utils.misc import get_model_outputs
from torch.optim import SGD
import matplotlib.pyplot as plt
```

In [144]:
```
classes = [[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]]
# First we initialize the classes that the model will be trying to guess.
# These classes will also be used for constructing a training set that has spurious correlations.
```

In [145]:
```
def visualize(data, class_index, num):
    count = 0;
    index = 0

    plt.figure()
    fig, axes = plt.subplots(1, num, figsize=(15, 1))

    while count < num:

        if data[index][1] == class_index:


            to_pil = torchvision.transforms.ToPILImage()
            img = to_pil(data[index][0])

            axes[count].imshow(img)
            axes[count].axis("off")

            count+=1

        index+=1

# This is a visualization method for seeing the types of images in each dataset, such as the training, and valset.
```

In [146]:
```
training = SpuCoMNIST(root="/data/mnist",
                      spurious_feature_difficulty=SpuriousFeatureDifficulty.MAGNITUDE_LARGE,
                      spurious_correlation_strength=0.9,
                      classes=classes,
                      split="train")

training.initialize()
# We initialize our training set with a spurious correlation strength of 0.9, and large difficulty.
# What this means is the background color will be quite clear to the model, and around 90% of the images in each class will have
# This pretty much forces us to use spurious correlation mitigation methods, such as the George method.
```

In [147]:
```
visualize(training, 0, 30)
# As you can see for class with index 0, there are lots of images (~90%) with clear red backgrounds.
```

```
<Figure size 640x480 with 0 Axes>
```

In [148]:
```
visualize(training, 1, 30)
# Then, for class with index 1, there are lots of images with clear green backgrounds.
# This means that if we were to train a model normally, it would likely rely on the spurious feature (background color) rather th
# As a result, the model would perform poorly for data where the spurious feature doesn't exist, or real-world situations.
```

```
<Figure size 640x480 with 0 Axes>
```

In [149]:
```
valset = SpuCoMNIST(
            root="/data/mnist/",
            spurious_feature_difficulty=SpuriousFeatureDifficulty.MAGNITUDE_LARGE,
            classes=classes,
            split="val")
valset.initialize()
# This is the evaluation set that will be used, which does not have the spurious feature.
# This means that all kinds of background colors are there for each class.
```

In [150]:
```
visualize(valset, 0, 30)
# As you can see for class index 0, there are all kinds of background colors.
# This is quite different from the training set where they have mostly red background colors.
# This would likely throw off the regular model, as it'd be relying on the background color to guess.
# For example, it'd probably guess that zero with a green background to be of class 1, despite being class 0.
```

```
<Figure size 640x480 with 0 Axes>
```

In [151]:
```
visualize(valset, 1, 30)
# The same is for other classes within the evaluation dataset.
```

```
<Figure size 640x480 with 0 Axes>
```

In [155]:
```
device = torch.device(f"cuda:0" if torch.cuda.is_available() else "cpu")
# Here, we just run the model on GPU for faster speeds.
```

In [156]:
```
model = model_factory("lenet", training[0][0].shape, 5).to(device)
# Then, we construct our first model to be trained without any sampling methods.
# We use the LeNet, which is based on Yann LeCun's papers about convolutional neural networks.
```

In [157]:
```
print(model)
# The model follows the methodology described by Yann Lecun: https://en.wikipedia.org/wiki/LeNet
```

```
SpuCoModel(
  (backbone): LeNet(
    (features): Sequential(
      (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): ReLU(inplace=True)
      (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
      (4): ReLU(inplace=True)
      (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (fc_1): Linear(in_features=400, out_features=120, bias=True)
    (fc_2): Linear(in_features=120, out_features=84, bias=True)
  )
  (classifier): Linear(in_features=84, out_features=5, bias=True)
)
```

In [158]:
```
erm_trainer = ERM(
    trainset=training,
    model=model,
    batch_size=32,
    optimizer=SGD(model.parameters(), lr=0.001, weight_decay=0.01, momentum=0.9),
    device=device,
    num_epochs=1,
    verbose=True
)
# Now, we train it using ERM, which will just sample the training set regularly, not accounting for the spurious feature in the d
```

In [159]:
```
erm_trainer.train()
# On the training set, the model performs decently, with about 90% accuracy.
# However this will not be the case for the evaluation set.
```

```
ERM | Epoch 0 | Loss: 0.4458264410495758 | Accuracy: 90.625%
ERM | Epoch 0 | Loss: 0.7453802824020386 | Accuracy: 81.25%
ERM | Epoch 0 | Loss: 0.22563818097114563 | Accuracy: 96.875%
ERM | Epoch 0 | Loss: 0.5421980023384094 | Accuracy: 87.5%
ERM | Epoch 0 | Loss: 0.5082492232322693 | Accuracy: 87.5%
ERM | Epoch 0 | Loss: 0.24782295525074005 | Accuracy: 96.875%
ERM | Epoch 0 | Loss: 0.22987522184848785 | Accuracy: 96.875%
ERM | Epoch 0 | Loss: 0.33848580718040466 | Accuracy: 93.75%
ERM | Epoch 0 | Loss: 0.33933985233306885 | Accuracy: 93.75%
ERM | Epoch 0 | Loss: 0.43228837847709656 | Accuracy: 90.625%
ERM | Epoch 0 | Loss: 0.23030555248260498 | Accuracy: 96.875%
ERM | Epoch 0 | Loss: 0.5703487992286682 | Accuracy: 87.5%
ERM | Epoch 0 | Loss: 0.3483372628688812 | Accuracy: 93.75%
ERM | Epoch 0 | Loss: 0.41484469175338745 | Accuracy: 90.625%
ERM | Epoch 0 | Loss: 0.4653094410896301 | Accuracy: 90.625%
ERM | Epoch 0 | Loss: 0.684420108795166 | Accuracy: 84.375%
ERM | Epoch 0 | Loss: 0.6105901002883911 | Accuracy: 84.375%
ERM | Epoch 0 | Loss: 0.12641319632530212 | Accuracy: 100.0%

Epoch 0: 100%|██████████████████████████████████████| 1501/1501 [01:43<00:00, 14.54batch/s, accuracy=100.0%, loss=0.126]
```

In [160]:
```
valid_evaluator1 = Evaluator(testset=valset,
                      group_partition=valset.group_partition,
                      group_weights=valset.group_weights,
                      batch_size=32,
                      model=model,
                      device=device,
                      verbose=True)
# Here we construct the evaluator using the valset
```

In [161]:
```
valid_evaluator1.evaluate()
# As you can see, the model performs poorly when the spurious feature is wrong or cannot be relied on
# In the training set, relying on the background color would have worked, but not for the actual test set.
# The model is getting 100% accuracy when the spurious feature is there, but getting 0% for something like class 0 with a green b
```

```
 (1, 0): 0.0,
 (1, 1): 100.0,
 (1, 2): 0.0,
 (1, 3): 0.0,
 (1, 4): 0.0,
 (2, 0): 0.0,
 (2, 1): 0.0,
 (2, 2): 100.0,
 (2, 3): 0.0,
 (2, 4): 0.0,
 (3, 0): 0.0,
 (3, 1): 0.0,
 (3, 2): 0.0,
 (3, 3): 100.0,
 (3, 4): 0.0,
 (4, 0): 0.0,
 (4, 1): 0.0,
 (4, 2): 0.0,
 (4, 3): 0.0,
 (4, 4): 100.0}
```

In [170]:
```
training_outputs = get_model_outputs(model, training, device, True, True)
# To mitigate this, we can change how we sample from the trainingset.
# We can gather the features the model had for each image it ran on, and create clusters which may avoid different types of spuri
```

```
Getting model outputs:   0%|                                                              | 0/751 [00:00<?, ?batch/s]Exceptio
n ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x0000020393EB8670>
Traceback (most recent call last):
  File "C:\Users\Ritik\AppData\Local\Programs\Python\Python310\lib\site-packages\torch\utils\data\dataloader.py", line 1478, in
__del__
    self._shutdown_workers()
  File "C:\Users\Ritik\AppData\Local\Programs\Python\Python310\lib\site-packages\torch\utils\data\dataloader.py", line 1436, in
_shutdown_workers
    if self._persistent_workers or self._workers_status[worker_id]:
AttributeError: '_MultiProcessingDataLoaderIter' object has no attribute '_workers_status'
Getting model outputs: 100%|██████████████████████████████████████████████████████| 751/751 [00:54<00:00, 13.82batch/s]
```

In [171]:
```
george_infer = GeorgeInference(Z=training_outputs,
                               class_labels=training.labels,
                               device=device,
                               max_clusters=5,
                               verbose=True)
# Using the gathered training model features and associated labels for the data it was trained on, we can create the clusters.
# I'm not actually sure which clustering method is used, but seems it is based on "No subclass left behind"
# https://github.com/BigML-CS-UCLA/SpuCo/tree/master/src/spuco/group_inference/george_utils
```

In [172]:
```
group_partition = george_infer.infer_groups()
# Begin determination of the clusters which will be used for partitioning the training set
```

```
C:\Users\Ritik\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
Clustering class-wise: 100%|████████████████████████████████████████████████████████████| 5/5 [00:53<00:00, 10.72s/it]
```

In [185]:
```
print(group_partition.keys())
# The different clusters that the model came out with
```

```
dict_keys([(2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11)])
```

In [173]:
```
model2 = model_factory("lenet", training[0][0].shape, training.num_classes).to(device)
# Now we can train an entirely new model which samples equally from each group.
```

In [174]:
```
robust_trainset = GroupLabeledDatasetWrapper(training, group_partition)
# We create a training set which is labeled according to the group partition it belongs to.
```

In [175]:
```
visualize(robust_trainset, 0, 30)
# It's the same training set as before, except each image is labeled a certain group partition.
```

```
<Figure size 640x480 with 0 Axes>
```

In [180]:
```
group_balance = GroupBalanceBatchERM(model=model2,
                    group_partition=group_partition,
                    num_epochs=10,
                    trainset=training,
                    batch_size=64,
                    optimizer=SGD(model2.parameters(), lr=1e-3, momentum=0.9),
                    device=device,
                    verbose=True)
# The GroupBalanceBatchERM method will train the model in a way which samples equally from the group partitions.
# This basically makes it difficult for the model to rely on the background color, as we're making it correlated with nothing rea
# For example, in class 0, because we sample equally, red will appear as much as any other color, so the model doesn't gain anyth
# As a result, it must rely on something else, like the shape of the number itself.
```

In [181]:
```
group_balance.train()
# As can be seen, the model takes longer to train, but in the end has higher accuracy.
```

```
GB | Epoch 9 | Loss: 0.03632992506027222 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.02461932972073555 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.03556530177593231 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.04515538364648819 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.03588464856147766 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.053310323506593704 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.06339140981435776 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.06394767761230469 | Accuracy: 98.4375%
GB | Epoch 9 | Loss: 0.09243414551019669 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.045264966785907745 | Accuracy: 98.4375%
GB | Epoch 9 | Loss: 0.025020882487297058 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.054721981287002563 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.11033645272254944 | Accuracy: 95.3125%
GB | Epoch 9 | Loss: 0.06864310055971146 | Accuracy: 95.3125%

Epoch 9:  99%|████████████████████████████████████ | 740/751 [00:57<00:00, 153.56batch/s, accuracy=100.0%, loss=0.0152]

GB | Epoch 9 | Loss: 0.06862547993659973 | Accuracy: 96.875%
GB | Epoch 9 | Loss: 0.04166870191693306 | Accuracy: 100.0%
GB | Epoch 9 | Loss: 0.015196019783616066 | Accuracy: 100.0%
```

In [182]:
```
valid_evaluator2 = Evaluator(testset=valset,
                      group_partition=valset.group_partition,
                      group_weights=valset.group_weights,
                      batch_size=32,
                      model=model2,
                      device=device,
                      verbose=True)
# We will evaluate the model in the same way as we did to the very first one.
```

In [186]:
```
valid_evaluator2.evaluate()
# As you can see, the model has high accuracy across all types of images, and not just for paritcular spurious correlations.
# This makes the model much better for real world situations, and more generalizable.
# For example, we could even introduce new types of images, where numbers have black backgrounds, and it'd likely still work.
# This is because we trained the model in a way it couldn't rely on any specific spurious feature.
```

```
 (1, 0): 90.49586776859505,
 (1, 1): 92.14876033057851,
 (1, 2): 86.95652173913044,
 (1, 3): 87.78467908902691,
 (1, 4): 89.02691511387164,
 (2, 0): 96.23059866962306,
 (2, 1): 94.90022172949003,
 (2, 2): 98.44444444444444,
 (2, 3): 87.33333333333333,
 (2, 4): 94.88888888888889,
 (3, 0): 96.72131147540983,
 (3, 1): 96.30390143737166,
 (3, 2): 95.68788501026694,
 (3, 3): 99.38398357289527,
 (3, 4): 96.30390143737166,
 (4, 0): 87.92372881355932,
 (4, 1): 80.9322033898305,
 (4, 2): 84.32203389830508,
 (4, 3): 89.19491525423729,
 (4, 4): 87.04883227176221}
```

In [ ]:
```
# By Ritik Jalisatgi, ritik@ucla.edu, along with help from SpuCo quickstart code
# https://github.com/BigML-CS-UCLA/SpuCo/tree/master/quickstart
```