Inside get_dataset(): torch.utils.data.Subset

```
indices_train = []
indices_class = [[] for c in range(num_classes)]
labels_all = [dst_train[i][1] for i in range(len(dst_train))]
for i, lab in enumerate(labels_all):
    indices_class[lab].append(i)

idx_to_del = []
for c in range(classes_modified):
    num_delete = len(indices_class[c]) - train_subset
    idx_to_del += indices_class[c][:num_delete]

   idx_shuffle = np.random.permutation(indices_class_train[c])[:train_subset_size]
    idx_shuffle = indices_class_train[c][:train_subset_size]
    indices_train += idx_shuffle
    indices_class_train[c] = idx_shuffle

indices_train = list(itertools.chain.from_iterable(indices_class_train))
    dst_train2 = torch.utils.data.Subset(copy.deepcopy(dst_train), indices_train)
```

Inside get_dataset(): Modifying dataset.data and dataset.targets → Np.delete

```
dst_train2 = copy.deepcopy(dst_train)
dst_train2.data = np.delete(dst_train2.data, idx_to_del, axis = 0)
dst_train2.targets = np.delete(dst_train2.targets, idx_to_del, axis = 0)
```

Inside get_dataset(): Modifying dataset.data and dataset.targets → Index the array

In main.py, modifying images_all with indices:

```
images_all = []
labels_all = []
indices_class = [[] for c in range(num_classes)]

images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
labels_all = [dst_train[i][1] for i in range(len(dst_train))]
for i, lab in enumerate(labels_all):
    indices_class[lab].append(i)

images_all = torch.cat(images_all, dim=0).to(args.device)
labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)


for c in range(args.classes_modified):
    indices_class[c] = indices_class[c][:args.train_subset]

final_index = []

for c in range(num_classes):
    final_index += indices_class[c]

images_all = images_all[np.array(final_index)]

images_all = torch.cat(images_all, dim=0).to(args.device)
```

In main.py, modifying dst_train.data and dst_train.targets:

```
indices_train = []
indices_class = [[] for c in range(num_classes)]
labels_all = [dst_train[i][1] for i in range(len(dst_train))]
for i, lab in enumerate(labels_all):
    indices_class[lab].append(i)

idx_to_del = []
for c in range(args.classes_modified):
    num_delete = len(indices_class[c]) - args.train_subset
    idx_to_del += indices_class[c][:num_delete]

dst_train2 = copy.deepcopy(dst_train)
dst_train2.data = np.delete(dst_train2.data, idx_to_del, axis = 0)
dst_train2.targets = np.delete(dst_train2.targets, idx_to_del, axis = 0)
```

In main.py, modifying dst_train with torch.tuils.data.Subset

```
    images_all = []
    labels_all = []
    indices_train = []
    indices_class = [[] for c in range(num_classes)]

    images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
    labels_all = [dst_train[i][1] for i in range(len(dst_train))]
    for i, lab in enumerate(labels_all):
        indices_class[lab].append(i)
    images_all = torch.cat(images_all, dim=0).to(args.device)
    labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)

    for c in range(args.classes_modified):
      idx_shuffle = indices_class[c][:args.train_subset]
      indices_class[c] = idx_shuffle
```

```
for c in range(num_classes):
 indices_train += idx_shuffle

torch.manual_seed(1234)

dst_train2 = torch.utils.data.Subset(copy.deepcopy(dst_train), indices_train)

images_all = [torch.unsqueeze(dst_train2[i][0], dim=0) for i in range(len(dst_train2))]
labels_all = [dst_train2[i][1] for i in range(len(dst_train2))]
images_all = torch.cat(images_all, dim=0).to(args.device)
labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)
```

CIFAR10 csv into custom pytorch dataset, with and without manual normalising (mean and std)

```
class Imbalanced(Dataset):

   def __init__(self, name):
     xy = np.array(data)
     xy = xy.astype(np.float)
     self.len = xy.shape[0]
     self.x_data = torch.from_numpy(xy[:, 1:])
     self.y_data = torch.from_numpy(xy[:, [0]])
     self.name = name

   def __getitem__(self, index):
     img, target = self.x_data[index], self.y_data[index]

     if self.name == "CIFAR10":
      mean = [0.4914, 0.4822, 0.4465]
      std = [0.2023, 0.1994, 0.2010]
      img = img.reshape(3,32,32)
      for i in range(3):
       img[i] = (img[i] - mean[i]) / std[i]

      img = torch.reshape(img, (1,3,32,32))

     return (img, target)

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

Pickle with Kai’s code

1. **With transpose and cv2.cvtColor, then transpose back and unsqueeze**
2. **With transpose and without cv2.cvtColor**

```
def unpickle(file):
          import pickle
          with open(file, 'rb') as fo:
              dict = pickle.load(fo, encoding='bytes')
          return dict


        data_dir =
"/mnt/cpfs/users/gpuwork/zheng.zhu/sixian_research/DatasetCondensation/data/cifar-10-bat
ches-py/"

        train_data = []
        train_labels = []

        for i in range(1, 6):
            data_dic = unpickle(data_dir + "/data_batch_"+str(i))

            train_labels += data_dic[b'labels']

            for j in range(10000):
                img = data_dic[b'data'][j]
                img = img.reshape(3,32,32)
                img = img.transpose(1,2,0)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


                # modification 1

                #img = img.transpose(2,0,1)
                #img = torch.from_numpy(img)
                #img = torch.unsqueeze(img, dim = 0)

                # modification 2

                mean = [0.4914, 0.4822, 0.4465]
                std = [0.2023, 0.1994, 0.2010]

                transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize(mean=mean, std=std)])
                img = transform(img)
                img = torch.unsqueeze(img, dim = 0)

                train_data.append(img)

        train_labels = np.array(train_labels)

        #modifying dataset:
```

```
        if classes_modified > 0:
            indices_class = [[] for c in range(num_classes)]

            for i, lab in enumerate(train_labels):
                if lab < classes_modified:
                    if len(indices_class[lab]) < train_subset:
                        indices_class[lab].append(i)
                else:
                    indices_class[lab].append(i)

            final_indices = list(itertools.chain.from_iterable(indices_class))
            dst_train = train_data[np.array(final_indices)]
            train_labels = train_labels[np.array(final_indices)]

        else:
          dst_train = train_data


        class_names=unpickle(data_dir+'/batches.meta')[b'label_names']
```

Pickle 16/9

→ 5.51pm version: no cv2.cvt, normalize with pytorch, unsqueeze at the end

→ 11:51pm version: added converting train_data to np array to support indexing

```
            img = data_dic[b'data'][j]
            img = img.reshape(3,32,32)
            img = img.transpose(1,2,0)

            #===modifications=========

            mean = [0.4914, 0.4822, 0.4465]
            std = [0.2023, 0.1994, 0.2010]

            transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize(mean=mean, std=std)])
            img = transform(img)


            img = torch.unsqueeze(img, dim = 0)
            train_data.append(img)
```

Csv 17/9 11.09am version

```
    img = np.array(img)
    img = img.astype("uint8")
    img = img.reshape(3,32,32)
    img = img.transpose(1,2,0)
    img = transform(img)
    img = torch.unsqueeze(img, dim = 0)
```

Old csv MNIST and FashionMNIST version

```
if self.name == "MNIST" or self.name == "FashionMNIST":
          if self.name == "MNIST":
            mean = [0.1307]
            std = [0.3081]
          elif self.name == "FashionMNIST":
            mean = [0.2861]
            std = [0.3530]

          img = (img - mean[0]) / std[0]
          img = torch.reshape(img, (1,1,28,28))
```

```
class Imbalanced(Dataset):

      def __init__(self, name):
        xy = np.array(data)
        xy = xy.astype(np.float)
        self.len = xy.shape[0]
        self.x_data = torch.from_numpy(xy[:, 1:])
        self.y_data = torch.from_numpy(xy[:, [0]])
        self.name = name

      def __getitem__(self, index):
        img, target = self.x_data[index], self.y_data[index]

        if self.name == "CIFAR10":
          mean = [0.4914, 0.4822, 0.4465]
          std = [0.2023, 0.1994, 0.2010]
          img = img.reshape(3,32,32)
          img = np.array(img)
          img = img.astype("uint8")
          img = img.reshape(3,32,32)
          img = img.transpose(1,2,0)
          img = transform(img)
          img = torch.unsqueeze(img, dim = 0)


        return (img, target)

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