Implementation Details

I used Google Collab for the implementation with Pytorch and GPU. To work with the pretrained AlexNet, I resized the images with `transforms.Resize(224)`, because AlexNet trained on ImageNet dataset, which has images size of 224x224, we need to match it to be able to use pretrained AlexNet. For normalization of input, I calculated means and standard deviations of images in both channels and use it. For example `transforms.Normalize([0.7108, 0.6784, 0.7242], [0.1603, 0.1857, 0.1047])` is the normalization for train dataset. In the pretrained AlexNet, I just modified the classifier head of it.

```
    (classifier): Sequential(
      (0): Dropout(p=0.5, inplace=False)
      (1): Linear(in_features=9216, out_features=4096, bias=True)
      (2): ReLU(inplace=True)
      (3): Dropout(p=0.5, inplace=False)
      (4): Linear(in_features=4096, out_features=4096, bias=True)
      (5): ReLU(inplace=True)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
```

Figure 1. Classifier head of original AlexNet.

```
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=3, bias=True)
  )
)
```

Figure 2. Classifier head after modification.

As it seems in the Figure 1 and Figure 2, I changed the out_features from 1000 to 3. It was 1000 at first because in ImageNet, there are 1000 classes. In our case, we got only 3 classes so we need to modify that. I modify it with the following lines:

```
num_ftrs = model.classifier[6].in_features

model.classifier[6] = nn.Linear(num_ftrs, 3)
```

For backpropagation, I used Cross Entropy Loss. As optimizer, I decided to stick with SGD with Momentum, but also Adam can be a good alternative. For the optimizer, I have to decide the learning rate and momentum. I decided to use learning rate = 0.001 and momentum = 0.9 which are common hyperparameter choices for optimizer. I used batch size as 4 and run it for 20 epochs.

For class imbalance problem, I add a helper function to calculate weights for each class and assign weight to the images. I applied this to the train set.

	Training portion of the training set				Validation portion of the training set				Test set			
	Class 1	Class 2	Class 3	Overall	Class 1	Class 2	Class 3	Overall	Class 1	Class 2	Class 3	Overall
With input normalization and with addressing the class-imbalance problem	**1.0**	**0.98**	**0.98**	**0.99**	**1.0**	**0.90**	**1.0**	**0.95**	**0.98**	**0.93**	**0.85**	**0.92**
With input normalization and <u>without</u> addressing the class-imbalance problem	**1.0**	**1.0**	**1.0**	**1.0**	**0.93**	**0.92**	**0.8**	**0.89**	**0.96**	**0.98**	**0.82**	**0.93**
<u>Without</u> input normalization and with addressing the class-imbalance problem	**0.91**	**1.0**	**0.94**	**0.95**	**1.0**	**0.90**	**0.75**	**0.89**	**0.98**	**0.82**	**0.87**	**0.89**