# ResNet18 and SEResNet18

In this section of Lab03, ResNet model was implemented on the CIFAR-10 dataset. With the same data set, 2 versions of ResNet were employed namely,

1. ResNet18
2. ResNet18 with Squeeze and Excitation (SEResNet18)

Since the given architechture of ResNet was not the same as what can be found on the original paper, the modification of the architechture was necessary. The modification includes:

e.g.

- The input image size
- The modification of the first convolutional layer
- The addition of a maxpool
- The padding
- The kernal size

The two versions were impleneted with the exact same optimizer as well as the loss functions and they are as follows:

- criterion = nn.CrossEntropyLoss()
- optimizer = optim.Adam(model.parameters(), lr=0.01)

The number of trainable parameters of each model are as follows:

1. ResNet18 has 11,181,642 trainable parameters
2. SEResNet18 has 11,268,682 trainable parameters

The models were both trained for 25 epochs with the batch size of 16 and their performace at the 25th epoch are:

- ResNet18: Train acc = 97.48% , Val acc = 89.20% , Test acc = 85.37% %
- SEResNet18: Train acc = 87.26% , Val acc = 84.63% , Test acc = 84.27% %

The average times taken per epoch are

- ResNet18: 1m 28s
- SEResNet18: 1m 30s

### Libaries

In [32]:

```
# Import libraries
import torch
import torchvision
from torchvision import datasets, models, transforms
import torch.nn as nn
import torch.optim as optim
import time
import os
from copy import copy
```

```python
from copy import deepcopy
import numpy as np
from torchvision.transforms.transforms import RandomCrop
```

# 1. Prepare data set

In [33]:

```python
## Resize to 256
train_preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))]

## Resize to 224

eval_preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))]

# Download CIFAR-10 and split into training, validation, and test sets.
# The copy of the training dataset after the split allows us to keep
# the same training/validation split of the original training set but
# apply different transforms to the training set and validation set.

full_train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True)

train_dataset, val_dataset = torch.utils.data.random_split(full_train_dataset
train_dataset.dataset = copy(full_train_dataset)
train_dataset.dataset.transform = train_preprocess
val_dataset.dataset.transform = eval_preprocess

test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                            download=True, transform=eval_pre


# Prepare the data loaders
BATCH_SIZE=4
NUM_WORKERS=2

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=BATC
                                            shuffle=True, num_workers=NUM_WOR
val_dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=BATCH_SI
                                            shuffle=False, num_workers=NUM_WO
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_
                                            shuffle=False, num_workers=NUM_WO

dataloaders = {'train': train_dataloader, 'val': val_dataloader}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

# 2. Define model

Since I did not change anything in the class BottleneckBlock, class BasicBlock as well as the function ResNet18, I omit them from this report

In [34]:

```python
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
```

```
        super().__init__()

        self.is_debug = False

        self.in_planes = 64
        # Initial convolution
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bia
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(3, stride = 2, padding = 1)
        # Residual blocks
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        # FC layer = 1 layer
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512 * block.EXPANSION, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.EXPANSION
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        if self.is_debug : print(f'conv1: {out.shape}')
        out = self.maxpool(out)
        if self.is_debug : print(f'max pool: {out.shape}')
        out = self.layer1(out)
        if self.is_debug : print(f'conv2_x: {out.shape}')
        out = self.layer2(out)
        if self.is_debug : print(f'conv3_x: {out.shape}')
        out = self.layer3(out)
        if self.is_debug : print(f'conv4_x: {out.shape}')
        out = self.layer4(out)
        if self.is_debug : print(f'conv5_x: {out.shape}')
        out = self.avgpool(out)
        if self.is_debug : print(f'avg pool: {out.shape}')
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

## 3. Define function for evaluation

In [35]:

```
def evaluate(model, iterator, criterion):

    total = 0
    correct = 0
    epoch_loss = 0
    epoch_acc = 0

    predicteds = []
    trues = []

    model.eval()

    with torch.no_grad():

        for batch, labels in iterator:
```

```
            #Move tensors to the configured device
            batch = batch.to(device)
            labels = labels.to(device)

            predictions = model(batch.float())

            loss = criterion(predictions, labels.long())

            predictions = nn.functional.softmax(predictions, dim=1)
            _, predicted = torch.max(predictions.data, 1)  #returns max value

            predicteds.append(predicted)
            trues.append(labels)
            total += labels.size(0)  #keep track of total
            correct += (predicted == labels).sum().item()  #.item() give the
            acc = 100 * (correct / total)

            epoch_loss += loss.item()
            epoch_acc += acc

    return epoch_loss / len(iterator), epoch_acc / len(iterator),predicteds,
```

# 4. Results

## 4.1 ResNet18

In [1]:

```
results_resnet18 = open("output.txt", "r")
print(results_resnet18.read())
```

```
====================== ResNet =======================
Epoch 0/24
----------
train Loss: 1.7385 Acc: 0.3537
Epoch time taken: 79.10425233840942
val Loss: 1.3343 Acc: 0.5266
Epoch time taken: 86.1614899635315
Epoch 1/24
----------
train Loss: 1.0930 Acc: 0.6095
Epoch time taken: 78.7505202293396
val Loss: 0.8933 Acc: 0.6902
Epoch time taken: 85.87395071983337
Epoch 2/24
----------
train Loss: 0.7773 Acc: 0.7301
Epoch time taken: 79.95591425895691
val Loss: 0.6817 Acc: 0.7647
Epoch time taken: 87.32848787307739
Epoch 3/24
----------
train Loss: 0.6345 Acc: 0.7805
Epoch time taken: 80.36202812194824
val Loss: 0.6076 Acc: 0.7932
Epoch time taken: 87.69425821304321
Epoch 4/24
----------
train Loss: 0.5268 Acc: 0.8183
Epoch time taken: 81.18930244445801
val Loss: 0.6157 Acc: 0.7901
Epoch time taken: 88.54764318466187
Epoch 5/24
----------
train Loss: 0.4451 Acc: 0.8463
Epoch time taken: 81.13013935089111
val Loss: 0.5607 Acc: 0.8123
```

```
Epoch time taken: 88.47652125358582
Epoch 6/24
----------
train Loss: 0.3797 Acc: 0.8705
Epoch time taken: 81.42113137245178
val Loss: 0.5120 Acc: 0.8336
Epoch time taken: 88.84501695632935
Epoch 7/24
----------
train Loss: 0.3241 Acc: 0.8887
Epoch time taken: 81.07497549057007
val Loss: 0.4944 Acc: 0.8343
Epoch time taken: 88.46439504623413
Epoch 8/24
----------
train Loss: 0.2755 Acc: 0.9051
Epoch time taken: 81.36177778244019
val Loss: 0.5401 Acc: 0.8361
Epoch time taken: 88.63901162147522
Epoch 9/24
----------
train Loss: 0.2352 Acc: 0.9184
Epoch time taken: 81.45833587646484
val Loss: 0.5576 Acc: 0.8399
Epoch time taken: 88.91071343421936
Epoch 10/24
----------
train Loss: 0.1993 Acc: 0.9317
Epoch time taken: 81.49073672294617
val Loss: 0.5457 Acc: 0.8391
Epoch time taken: 88.65237641334534
Epoch 11/24
----------
train Loss: 0.1737 Acc: 0.9394
Epoch time taken: 80.97939777374268
val Loss: 0.5219 Acc: 0.8484
Epoch time taken: 88.35349464416504
Epoch 12/24
----------
train Loss: 0.1543 Acc: 0.9473
Epoch time taken: 81.62639999389648
val Loss: 0.5630 Acc: 0.8450
Epoch time taken: 88.91089749336243
Epoch 13/24
----------
train Loss: 0.1371 Acc: 0.9518
Epoch time taken: 81.42808794975281
val Loss: 0.6266 Acc: 0.8372
Epoch time taken: 88.84452176094055
Epoch 14/24
----------
train Loss: 0.1296 Acc: 0.9560
Epoch time taken: 81.36918210983276
val Loss: 0.5218 Acc: 0.8522
Epoch time taken: 88.84109807014465
Epoch 15/24
----------
train Loss: 0.1141 Acc: 0.9609
Epoch time taken: 81.72880268096924
val Loss: 0.6209 Acc: 0.8432
Epoch time taken: 89.15606212615967
Epoch 16/24
----------
train Loss: 0.1012 Acc: 0.9649
Epoch time taken: 81.49277138710022
val Loss: 0.6121 Acc: 0.8511
Epoch time taken: 88.83721399307251
Epoch 17/24
----------
```

```
train Loss: 0.0980 Acc: 0.9670
Epoch time taken: 81.22549605369568
val Loss: 0.6306 Acc: 0.8470
Epoch time taken: 88.65475583076477
Epoch 18/24
----------
train Loss: 0.0909 Acc: 0.9691
Epoch time taken: 81.9236741065979
val Loss: 0.6568 Acc: 0.8485
Epoch time taken: 89.47668099403381
Epoch 19/24
----------
train Loss: 0.0863 Acc: 0.9703
Epoch time taken: 81.24633646011353
val Loss: 0.6754 Acc: 0.8429
Epoch time taken: 88.79005408287048
Epoch 20/24
----------
train Loss: 0.0784 Acc: 0.9733
Epoch time taken: 81.36391472816467
val Loss: 0.7252 Acc: 0.8430
Epoch time taken: 88.80256843566895
Epoch 21/24
----------
train Loss: 0.0782 Acc: 0.9722
Epoch time taken: 81.27300786972046
val Loss: 0.6668 Acc: 0.8524
Epoch time taken: 88.67334175109863
Epoch 22/24
----------
train Loss: 0.0788 Acc: 0.9743
Epoch time taken: 81.34923195838928
val Loss: 0.6903 Acc: 0.8430
Epoch time taken: 88.80050849914551
Epoch 23/24
----------
train Loss: 0.0727 Acc: 0.9745
Epoch time taken: 82.22561049461365
val Loss: 0.7527 Acc: 0.8469
Epoch time taken: 89.58947706222534
Epoch 24/24
----------
train Loss: 0.0744 Acc: 0.9748
Epoch time taken: 81.75759553909302
val Loss: 0.7167 Acc: 0.8511
Epoch time taken: 89.1967556476593
```

### 4.2 SEResNet18

In [37]:

```
results_SEresnet18 = open("output_SE.txt", "r")
print(results_SEresnet18.read())
```

```
====================== SE ResNet =======================
Epoch 0/24
----------
train Loss: 1.7207 Acc: 0.3654
Epoch time taken: 86.21501898765564
val Loss: 1.4562 Acc: 0.4585
Epoch time taken: 93.75058031082153
Epoch 1/24
----------
train Loss: 1.1997 Acc: 0.5707
Epoch time taken: 86.72668242454529
val Loss: 0.9881 Acc: 0.6439
Epoch time taken: 94.23482990264893
Epoch 2/24
----------
```

```
train Loss: 0.8953 Acc: 0.6849
Epoch time taken: 86.90648603439331
val Loss: 0.7852 Acc: 0.7236
Epoch time taken: 94.48461151123047
Epoch 3/24
----------
train Loss: 0.7225 Acc: 0.7477
Epoch time taken: 87.01340055465698
val Loss: 0.6700 Acc: 0.7670
Epoch time taken: 94.60406541824341
Epoch 4/24
----------
train Loss: 0.6108 Acc: 0.7885
Epoch time taken: 86.98946404457092
val Loss: 0.5972 Acc: 0.7904
Epoch time taken: 94.5728771686554
Epoch 5/24
----------
train Loss: 0.5254 Acc: 0.8183
Epoch time taken: 87.02860951423645
val Loss: 0.5800 Acc: 0.8046
Epoch time taken: 94.61602449417114
Epoch 6/24
----------
train Loss: 0.4563 Acc: 0.8417
Epoch time taken: 86.98890829086304
val Loss: 0.5583 Acc: 0.8154
Epoch time taken: 94.55210256576538
Epoch 7/24
----------
train Loss: 0.4034 Acc: 0.8605
Epoch time taken: 87.24596619606018
val Loss: 0.5194 Acc: 0.8237
Epoch time taken: 94.829354763031
Epoch 8/24
----------
train Loss: 0.3445 Acc: 0.8811
Epoch time taken: 87.22963333129883
val Loss: 0.5709 Acc: 0.8134
Epoch time taken: 94.79582071304321
Epoch 9/24
----------
train Loss: 0.3070 Acc: 0.8935
Epoch time taken: 87.12807273864746
val Loss: 0.5009 Acc: 0.8342
Epoch time taken: 94.68710994720459
Epoch 10/24
----------
train Loss: 0.2661 Acc: 0.9075
Epoch time taken: 87.20602250099182
val Loss: 0.4962 Acc: 0.8437
Epoch time taken: 94.78709721565247
Epoch 11/24
----------
train Loss: 0.2353 Acc: 0.9189
Epoch time taken: 87.26906657218933
val Loss: 0.5175 Acc: 0.8398
Epoch time taken: 94.84312391281128
Epoch 12/24
----------
train Loss: 0.2068 Acc: 0.9278
Epoch time taken: 87.21528601646423
val Loss: 0.5810 Acc: 0.8256
Epoch time taken: 94.8557620048523
Epoch 13/24
----------
train Loss: 0.1875 Acc: 0.9360
Epoch time taken: 87.20333886146545
val Loss: 0.6358 Acc: 0.8197
```

```
Epoch time taken: 94.79728245735168
Epoch 14/24
----------
train Loss: 0.1667 Acc: 0.9418
Epoch time taken: 87.25335454940796
val Loss: 0.6255 Acc: 0.8254
Epoch time taken: 94.83267593383789
Epoch 15/24
----------
train Loss: 0.1504 Acc: 0.9485
Epoch time taken: 87.17038702964783
val Loss: 0.5749 Acc: 0.8463
Epoch time taken: 94.74958300590515
Epoch 16/24
----------
train Loss: 0.1417 Acc: 0.9518
Epoch time taken: 87.20997190475464
val Loss: 0.6886 Acc: 0.8243
Epoch time taken: 94.78706407546997
Epoch 17/24
----------
train Loss: 0.1273 Acc: 0.9566
Epoch time taken: 87.24172949790955
val Loss: 0.6476 Acc: 0.8377
Epoch time taken: 94.78197455406189
Epoch 18/24
----------
train Loss: 0.1245 Acc: 0.9577
Epoch time taken: 87.19077634811401
val Loss: 0.6463 Acc: 0.8440
Epoch time taken: 94.74433970451355
Epoch 19/24
----------
train Loss: 0.1138 Acc: 0.9605
Epoch time taken: 87.14714217185974
val Loss: 0.6356 Acc: 0.8439
Epoch time taken: 94.73013043403625
Epoch 20/24
----------
train Loss: 0.1070 Acc: 0.9619
Epoch time taken: 87.29391169548035
val Loss: 0.6377 Acc: 0.8359
Epoch time taken: 94.84334802627563
Epoch 21/24
----------
train Loss: 0.1044 Acc: 0.9650
Epoch time taken: 87.22932171821594
val Loss: 0.6333 Acc: 0.8432
Epoch time taken: 94.80563068389893
Epoch 22/24
----------
train Loss: 0.0947 Acc: 0.9684
Epoch time taken: 87.21725869178772
val Loss: 0.6990 Acc: 0.8314
Epoch time taken: 94.7738401889801
Epoch 23/24
----------
train Loss: 0.0911 Acc: 0.9682
Epoch time taken: 87.23177981376648
val Loss: 0.7036 Acc: 0.8416
Epoch time taken: 94.82511758804321
Epoch 24/24
----------
train Loss: 0.0908 Acc: 0.9689
Epoch time taken: 87.2577965259552
val Loss: 0.6878 Acc: 0.8463
Epoch time taken: 94.83099102973938
```

## 5. Evaluatation

In [38]:
```
#%% Check avilability of the GPU
from chosen_gpu import get_freer_gpu
device = torch.device(get_freer_gpu()) if torch.cuda.is_available() else torc
print("Configured device: ", device)
```

```
Configured device:  cuda:0
```

In [39]:
```
from modules import ResNet18, ResSENet18
model = ResNet18()
model_SE = ResSENet18()
criterion = nn.CrossEntropyLoss()
model = model.to(device)
model_SE = model_SE.to(device)
criterion = criterion.to(device)
```

In [40]:
```
model.load_state_dict(torch.load('resnet18_bestsofar.pth',))
model_SE.load_state_dict(torch.load('SEresnet18_bestsofar.pth'))
test_loss, test_acc, test_pred_label, test_true_label  = evaluate(model, test_
test_loss_SE, test_acc_SE, test_pred_label_SE, test_true_label_SE  = evaluate
print('================== ResNet18 ======================')
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc:.2f}%')
print('================== SEResNet18 ====================')
print(f'Test Loss: {test_loss_SE:.3f} | Test Acc: {test_acc_SE:.2f}%')
```

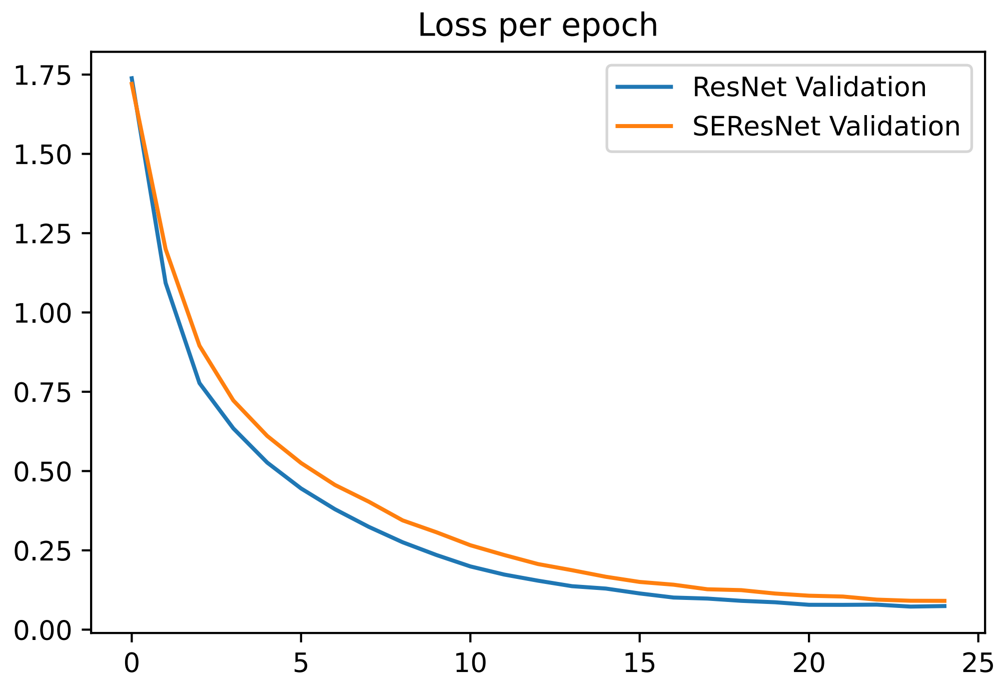
```
================== ResNet18 ======================
Test Loss: 0.680 | Test Acc: 85.37%
================== SEResNet18 ====================
Test Loss: 0.594 | Test Acc: 84.27%
```

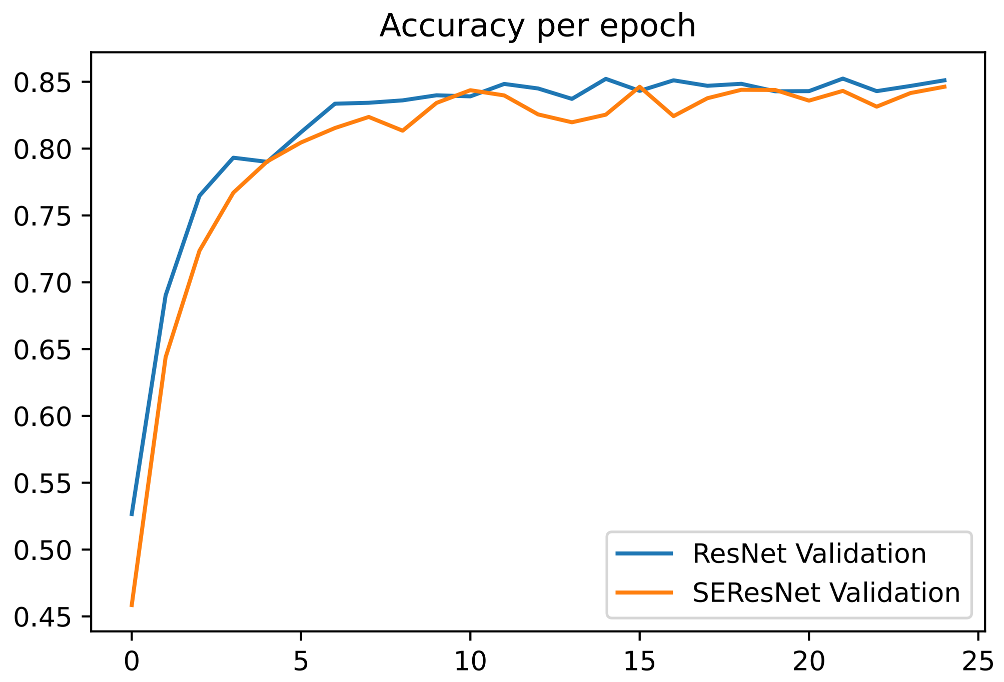
## 6. Plot

In [41]:
```
import matplotlib.pyplot as plt

def plot_data(val_acc_history, loss_acc_history, val_acc_history2, loss_acc_h
    plt.plot(loss_acc_history, label = 'ResNet Validation')
    plt.plot(loss_acc_history2, label = 'SEResNet Validation')
    plt.title('Loss per epoch')
    plt.legend()
    plt.show()
    plt.plot(val_acc_history, label = 'ResNet Validation')
    plt.plot(val_acc_history2, label = 'SEResNet Validation')
    plt.title('Accuracy per epoch')
    plt.legend()
    plt.show()
```

In [42]:
```
val_acc_history_resnet18 =  np.load('history/val_acc_history_resnet18_25.npy'
loss_acc_history_resnet18 = np.load('history/loss_acc_history_resnet18_25.npy
val_acc_history_SEresnet18 =  np.load('history/val_acc_history_SEresnet18_25.
loss_acc_history_SEresnet18 = np.load('history/loss_acc_history_SEresnet18_25
```

In [43]:
```
plot_data(val_acc_history_resnet18, loss_acc_history_resnet18, val_acc_histor
```




## Chihuahua Muffin

In this section of Lab03, taken the trained SE_ResNet model from the previous section, the model was fine-tuned by adjusting the hyper-paremeters during the 8-fold cross validation.

Since SE_ResNet was initially trained for a 10-class classification problem, The last layer of the output must be modified. In our case, the last layer was changed from 10 to 2.

The 8 variations (8-fold cross validation) were impleneted with Adam optimizer and crossentropy loss functions and they are as follows:

- criterion = nn.CrossEntropyLoss()
- optimizer = optim.Adam() which starts lr = 0.05 and increases by 0.05 every model.

The model was trained for 25 epochs with the batch size of 4 and their performace at the 25th epoch are:

- With optimizer0: Avg acc = 0.7999999999999999
- With optimizer1: Avg acc = 0.8474999999999999
- With optimizer2: Avg acc = 0.8925
- With optimizer3: Avg acc = 0.8625
- With optimizer4: Avg acc = 0.7300000000000001
- With optimizer5: Avg acc = 0.7300000000000001
- With optimizer6: Avg acc = 0.715
- With optimizer7: Avg acc = 0.8

** average accuracy of all epochs and folds

As for the evalution, optimizer2 was chosen due to its high accuracy as well as relatively less time taken to train which results in 87.50% on 4 different pictures of chihuahua and muffin taken from the internet

# 1. Import the dataset

In [71]:

```
import torchvision.datasets as dset
import torchvision.utils as vutils
from torch.utils.data import TensorDataset
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt

dataset = dset.ImageFolder(root='chihuahua_muffin',
                           transform=transforms.Compose([
                               transforms.Resize(256),
                               transforms.CenterCrop(224),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.
                           ]))
```

# 2. Define train function

In [72]:

```
def train_model(model, dataloaders, criterion, optimizer, num_epochs=25, weig
    print('====================== New Run =======================',file=open(

    since = time.time()

    val_acc_history = []
    loss_acc_history = []

    best_model_wts = deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        epoch_start = time.time()
        print('Epoch {}/{}'.format(epoch, num_epochs - 1), file=open(f"{weigh
        print('-' * 10, file=open(f"{weights_name}.txt", "a"))

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0
```

```
            for inputs, labels in dataloaders[phase]:

                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):

                    if is_inception and phase == 'train':
                        outputs, aux_outputs = model(inputs)

                        loss1 = criterion(outputs, labels)
                        loss2 = criterion(aux_outputs, labels)
                        loss = loss1 + 0.4*loss2
                    else:
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)

                    _, preds = torch.max(outputs, 1)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(dataloaders[phase].dataset)
            epoch_acc = running_corrects.double() / len(dataloaders[phase].da
            epoch_end = time.time()

            elapsed_epoch = epoch_end - epoch_start

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epo
            print(f"Epoch time taken: {elapsed_epoch}", file=open(f"{weights_

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = deepcopy(model.state_dict())
                torch.save(model.state_dict(), weights_name + ".pth")
            if phase == 'val':
                val_acc_history.append(epoch_acc)
            if phase == 'train':
                loss_acc_history.append(epoch_loss)

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, t
    print('Best val Acc: {:4f}'.format(best_acc), file=open(f"{weights_name}.

    return  val_acc_history, loss_acc_history
```

## 3. Check the availability of GPU

In [73]:

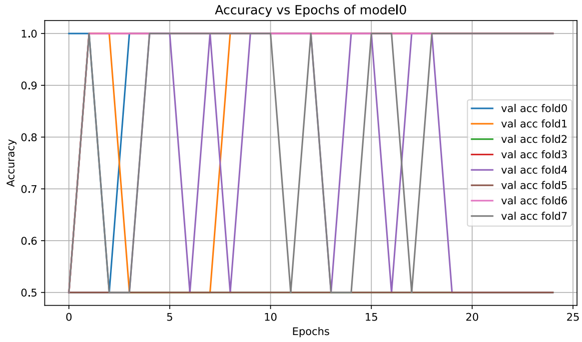
```
import torch
from chosen_gpu import get_freer_gpu
device = torch.device(get_freer_gpu()) if torch.cuda.is_available() else torc
print("Configured device: ", device)
```

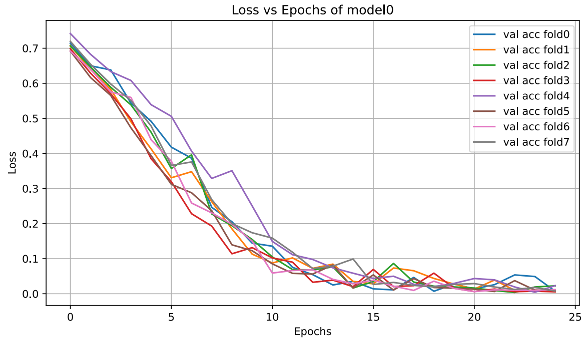
```
Configured device:  cuda:0
```

## 4. Perform 8-Fold Cross Validation

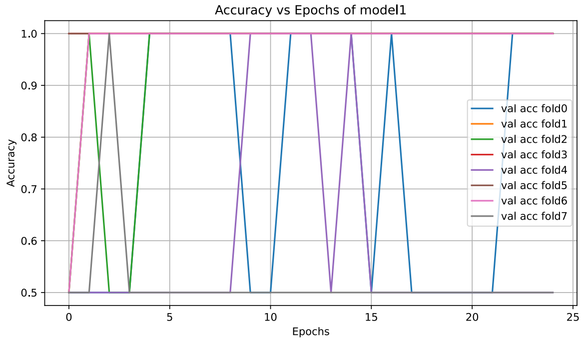
In [74]:

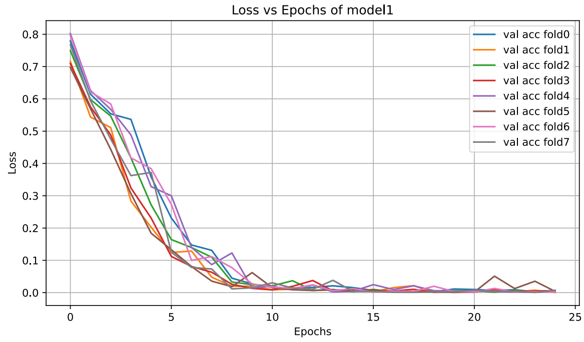
```
folds = 8
skf = StratifiedKFold(n_splits=folds, shuffle=True)


from modules import ResSENet18
models = []
def make_model(ResSENet18):
    model = ResSENet18()
    model.load_state_dict(torch.load('SEresnet18_bestsofar.pth'))
    model.linear = nn.Linear(512,2)
    model.eval()
    return model

n_models = 8

for i in np.arange(n_models):
    fig,ax = plt.subplots(1,2,sharex=True,figsize=(20,5))
    model_acc = 0
    for fold, (train_index, val_index) in enumerate(skf.split(dataset, datase
        print('******************** Fold {}/{} ******************** '.format
        batch_size = 4
        train = torch.utils.data.Subset(dataset, train_index)
        val = torch.utils.data.Subset(dataset, val_index)

        train_loader = torch.utils.data.DataLoader(train, batch_size=batch_si
        val_loader = torch.utils.data.DataLoader(val, batch_size=batch_size,

        dataloaders = {'train': train_loader, 'val': val_loader}

        model = make_model(ResSENet18)
        model.to(device)
        dataloaders = {'train': train_loader, 'val': val_loader}
        criterion = nn.CrossEntropyLoss().to(device)
        optimizer =  optim.Adam(model.parameters(), lr = 0.005 + 0.005*i)

        val_acc_history, loss_acc_history = train_model(model, dataloaders, c

        ax[0].plot(np.arange(25),np.array(val_acc_history),label = f"val acc
        ax[1].plot(np.arange(25),np.array(loss_acc_history),label = f"val acc
        ax[0].set_xlabel("Epochs")
        ax[1].set_xlabel("Epochs")
        ax[0].set_ylabel("Accuracy")
        ax[1].set_ylabel("Loss")
        ax[0].set_title(f"Accuracy vs Epochs of model{i}")
        ax[1].set_title(f"Loss vs Epochs of model{i}")
        ax[0].legend()
        ax[1].legend()
        ax[0].grid(True)
        ax[1].grid(True)
        #print(len(val_acc_history))
        #print(sum(val_acc_history))
        model_acc = model_acc + sum(val_acc_history)/len(val_acc_history)

    plt.show()
    print(f'Average accuracy of model{i}: {model_acc/8}')
```



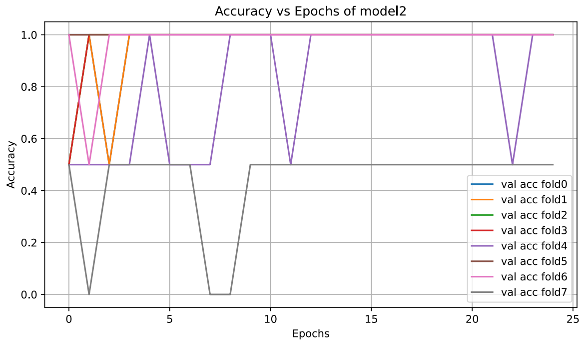



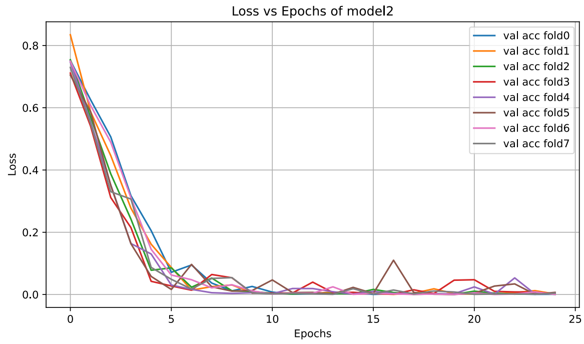

```
Average accuracy of model0: 0.7999999999999999
```



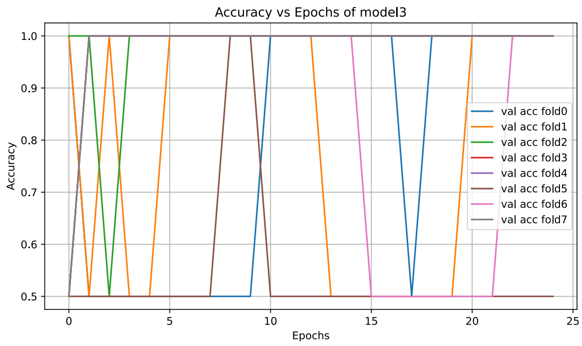



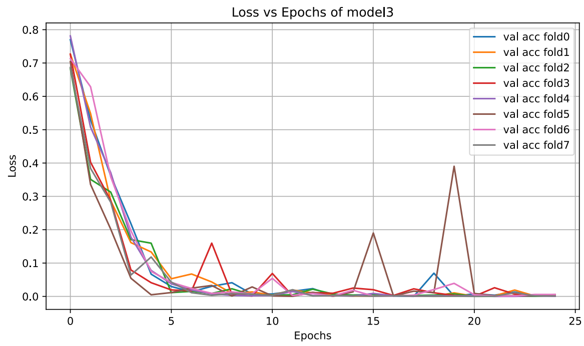

```
Average accuracy of model1: 0.8474999999999999
```



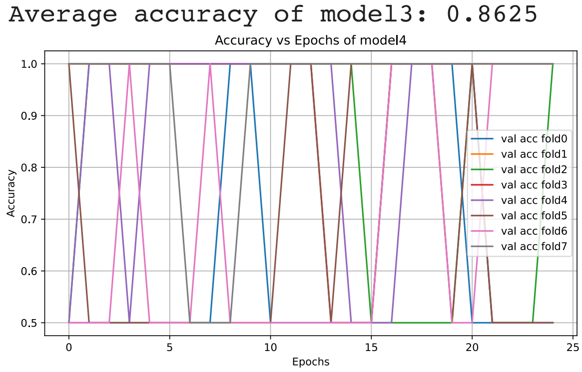



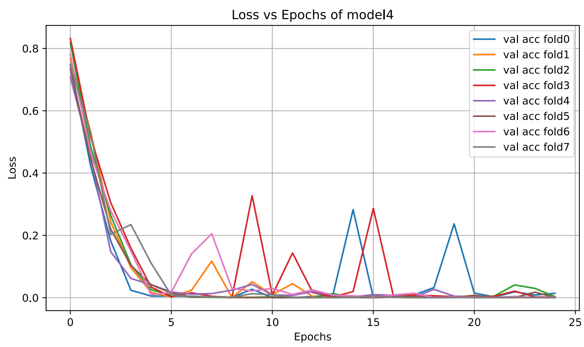

```
Average accuracy of model2: 0.8925
```



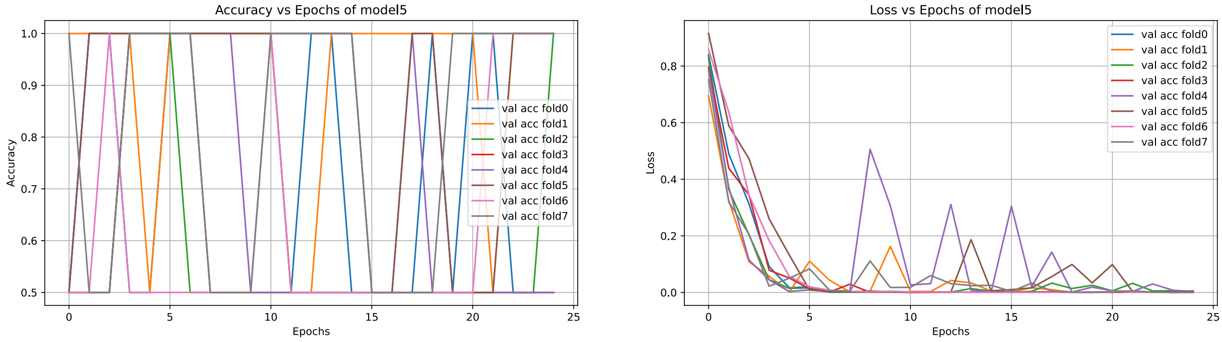




```
Average accuracy of model3: 0.8625
```

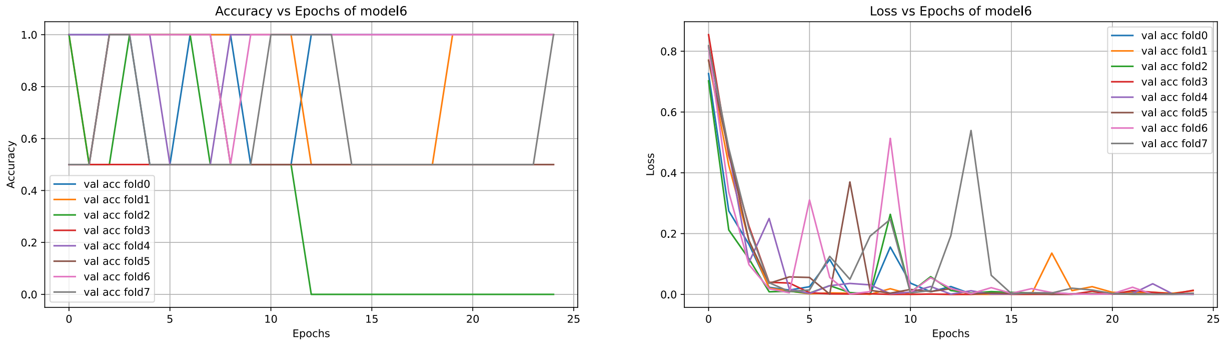


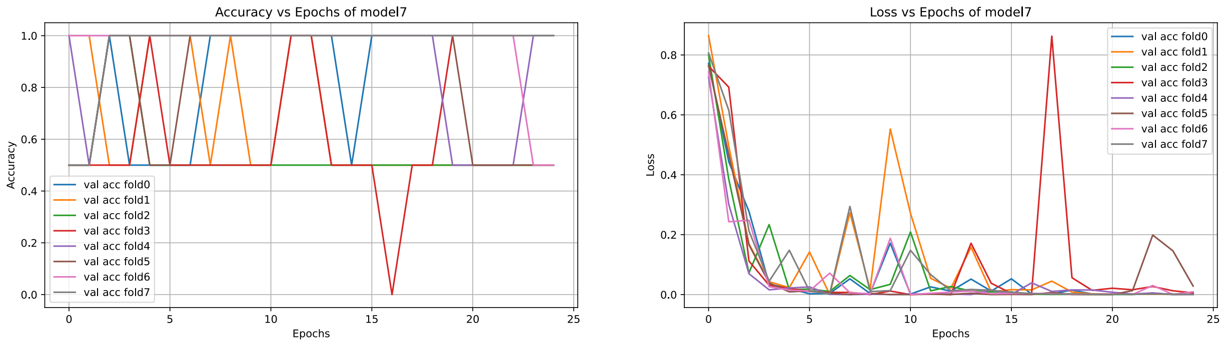

```
Average accuracy of model4: 0.8125
```




```
Average accuracy of model5: 0.7300000000000001
```




```
Average accuracy of model6: 0.715
```




```
Average accuracy of model7: 0.8
```

# 5. Train model with optimizer2

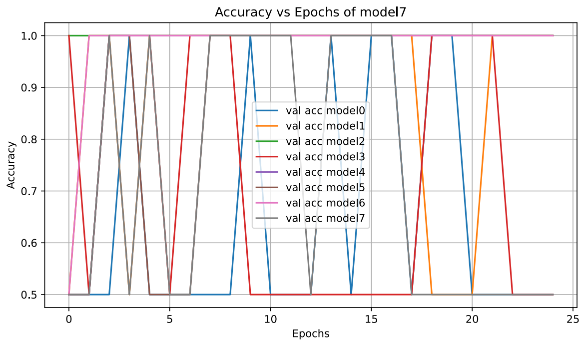
Optimizer2 was chosen due to its highest in average accuracy

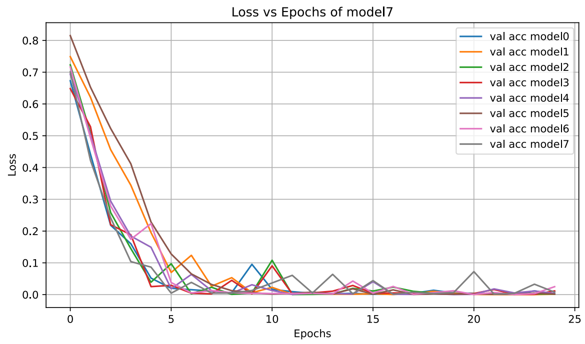
Optimizer2 = optim.Adam(model.parameters(), lr = 0.015)

In [77]:

```
fig,ax = plt.subplots(1,2,sharex=True,figsize=(20,5))
model_acc = 0
for fold, (train_index, val_index) in enumerate(skf.split(dataset, dataset.ta
        print('******************** Fold {}/{} ******************** '.format
        batch_size = 4
        train = torch.utils.data.Subset(dataset, train_index)
        val = torch.utils.data.Subset(dataset, val_index)

        train_loader = torch.utils.data.DataLoader(train, batch_size=batch_si
        val_loader = torch.utils.data.DataLoader(val, batch_size=batch_size,

        dataloaders = {'train': train_loader, 'val': val_loader}

        model = make_model(ResSENet18)
        model.to(device)
        dataloaders = {'train': train_loader, 'val': val_loader}
        criterion = nn.CrossEntropyLoss().to(device)
        optimizer =  optim.Adam(model.parameters(), lr = 0.005 + 0.005*2)

        val_acc_history, loss_acc_history = train_model(model, dataloaders, c
```

```
        ax[0].plot(np.arange(25),np.array(val_acc_history),label = f"val acc m
        ax[1].plot(np.arange(25),np.array(loss_acc_history),label = f"val acc
        ax[0].set_xlabel("Epochs")
        ax[1].set_xlabel("Epochs")
        ax[0].set_ylabel("Accuracy")
        ax[1].set_ylabel("Loss")
        ax[0].set_title(f"Accuracy vs Epochs of model{fold}")
        ax[1].set_title(f"Loss vs Epochs of model{fold}")
        ax[0].legend()
        ax[1].legend()
        ax[0].grid(True)
        ax[1].grid(True)
        #print(len(val_acc_history))
        #print(sum(val_acc_history))
        model_acc = model_acc + sum(val_acc_history)/len(val_acc_history)
plt.show()
print(f'Average accuracy of model{i}: {model_acc/8}')
```




```
Average accuracy of model7: 0.8524999999999999
```

## 6. Results

In [78]:

```
results_resnet18_muff = open("Train_SE_muff_chi_model.txt", "r")
print(results_resnet18_muff.read())
```

```
====================== Train_SE_muff_chi_model =======================
Epoch 0/24
----------
train Loss: 0.7008 Acc: 0.6429
Epoch time taken: 0.09666061401367188
val Loss: 0.6711 Acc: 0.5000
Epoch time taken: 0.10734295845031738
Epoch 1/24
----------
train Loss: 0.4225 Acc: 1.0000
Epoch time taken: 0.13723373413085938
val Loss: 0.6484 Acc: 0.5000
Epoch time taken: 0.1564638614654541
Epoch 2/24
----------
train Loss: 0.2417 Acc: 1.0000
Epoch time taken: 0.11687064170837402
val Loss: 0.6080 Acc: 1.0000
Epoch time taken: 0.12678956985473633
Epoch 3/24
----------
train Loss: 0.1039 Acc: 1.0000
Epoch time taken: 0.15042591094970703
val Loss: 0.5566 Acc: 0.5000
Epoch time taken: 0.17507648468017578
Epoch 4/24
----------
train Loss: 0.0865 Acc: 1.0000
Epoch time taken: 0.160980224609375
```

```
val Loss: 0.5449 Acc: 1.0000
Epoch time taken: 0.2153942584991455
Epoch 5/24
----------
train Loss: 0.0046 Acc: 1.0000
Epoch time taken: 0.18247509002685547
val Loss: 0.5445 Acc: 0.5000
Epoch time taken: 0.19686269760131836
Epoch 6/24
----------
train Loss: 0.0385 Acc: 1.0000
Epoch time taken: 0.13381004333496094
val Loss: 0.5572 Acc: 0.5000
Epoch time taken: 0.16004729270935059
Epoch 7/24
----------
train Loss: 0.0049 Acc: 1.0000
Epoch time taken: 0.14160966873168945
val Loss: 0.4764 Acc: 1.0000
Epoch time taken: 0.15662693977355957
Epoch 8/24
----------
train Loss: 0.0052 Acc: 1.0000
Epoch time taken: 0.1320176124572754
val Loss: 0.4878 Acc: 1.0000
Epoch time taken: 0.1465773582458496
Epoch 9/24
----------
train Loss: 0.0130 Acc: 1.0000
Epoch time taken: 0.20499539375305176
val Loss: 0.4406 Acc: 1.0000
Epoch time taken: 0.21779465675354004
Epoch 10/24
----------
train Loss: 0.0367 Acc: 1.0000
Epoch time taken: 0.08951091766357422
val Loss: 0.4659 Acc: 1.0000
Epoch time taken: 0.0996408462524414
Epoch 11/24
----------
train Loss: 0.0607 Acc: 0.9286
Epoch time taken: 0.08830952644348145
val Loss: 0.4135 Acc: 1.0000
Epoch time taken: 0.10071873664855957
Epoch 12/24
----------
train Loss: 0.0052 Acc: 1.0000
Epoch time taken: 0.08851456642150879
val Loss: 0.4132 Acc: 0.5000
Epoch time taken: 0.09856677055358887
Epoch 13/24
----------
train Loss: 0.0639 Acc: 1.0000
Epoch time taken: 0.0878450870513916
val Loss: 0.2954 Acc: 1.0000
Epoch time taken: 0.09878087043762207
Epoch 14/24
----------
train Loss: 0.0027 Acc: 1.0000
Epoch time taken: 0.08920454978942871
val Loss: 0.3085 Acc: 1.0000
Epoch time taken: 0.09890294075012207
Epoch 15/24
----------
train Loss: 0.0441 Acc: 1.0000
Epoch time taken: 0.08720564842224121
val Loss: 0.3045 Acc: 1.0000
Epoch time taken: 0.13122868537902832
Epoch 16/24
```

```
----------
train Loss: 0.0012 Acc: 1.0000
Epoch time taken: 0.18921375274658203
val Loss: 0.3581 Acc: 1.0000
Epoch time taken: 0.20605802536010742
Epoch 17/24
----------
train Loss: 0.0075 Acc: 1.0000
Epoch time taken: 0.16023540496826172
val Loss: 0.4485 Acc: 0.5000
Epoch time taken: 0.1779024600982666
Epoch 18/24
----------
train Loss: 0.0030 Acc: 1.0000
Epoch time taken: 0.16343235969543457
val Loss: 0.4798 Acc: 0.5000
Epoch time taken: 0.18760323524475098
Epoch 19/24
----------
train Loss: 0.0066 Acc: 1.0000
Epoch time taken: 0.17200970649719238
val Loss: 0.4792 Acc: 0.5000
Epoch time taken: 0.18838047981262207
Epoch 20/24
----------
train Loss: 0.0724 Acc: 0.9286
Epoch time taken: 0.11550569534301758
val Loss: 0.5652 Acc: 0.5000
Epoch time taken: 0.13154387474060059
Epoch 21/24
----------
train Loss: 0.0017 Acc: 1.0000
Epoch time taken: 0.11406397819519043
val Loss: 0.8340 Acc: 0.5000
Epoch time taken: 0.12906193733215332
Epoch 22/24
----------
train Loss: 0.0039 Acc: 1.0000
Epoch time taken: 0.10764861106872559
val Loss: 0.9420 Acc: 0.5000
Epoch time taken: 0.12497138977050781
Epoch 23/24
----------
train Loss: 0.0328 Acc: 1.0000
Epoch time taken: 0.11159157752990723
val Loss: 0.8898 Acc: 0.5000
Epoch time taken: 0.1236569881439209
Epoch 24/24
----------
train Loss: 0.0078 Acc: 1.0000
Epoch time taken: 0.10419845581054688
val Loss: 0.5948 Acc: 0.5000
Epoch time taken: 0.11961865425109863
Training complete in 0m 5s
Best val Acc: 1.000000
```

# 7. Evaluation

## 7.1 Import the dataset

In [79]:

```
dataset_test = dset.ImageFolder(root='chi_muff_test',
                           transform=transforms.Compose([
                               transforms.Resize(256),
                               transforms.CenterCrop(224),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.
```

```
                        ]))
test_dataloader = torch.utils.data.DataLoader(dataset_test,batch_size = 8,shu
```

### 7.2 Define and run the model

In [80]:

```
model_test = ResSENet18()
model_test.linear = nn.Linear(512,2)
model_test.eval()
model_test.to(device)
model_test.load_state_dict(torch.load(f'Train_SE_muff_chi_model.pth'))
criterion = nn.CrossEntropyLoss()
test_loss, test_acc, test_pred_label, test_true_label  = evaluate(model_test,
```

In [81]:

```
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc:.2f}%')
```

```
Test Loss: 0.485 | Test Acc: 87.50%
```