Progress Report: Indentity Recognition With Masks

Group member: Chen Zhao, Ruihua Qiao, Zijian Chen

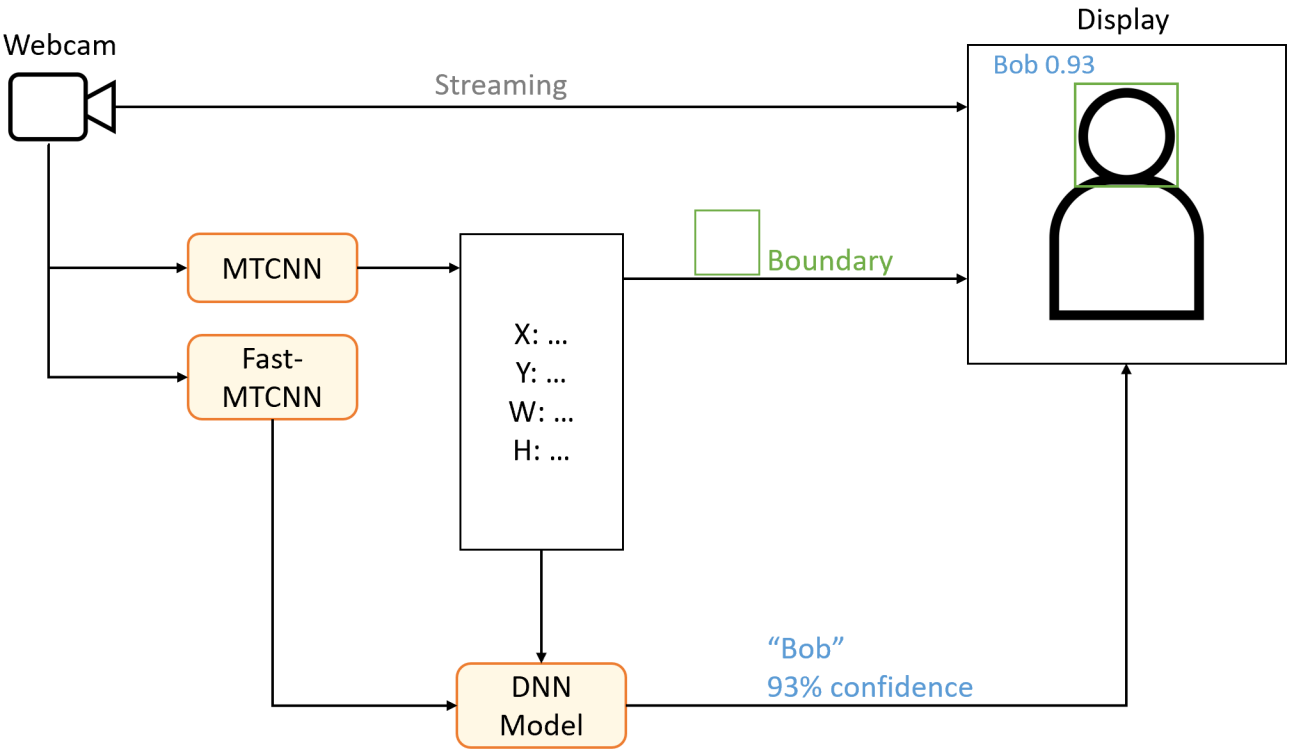
Human facial recognition is a classical task of computer vision. It has important applications in access control, attendance counting, facial security checks, etc. With the outbreak of the Covid-19, wearing masks has become mandatory for most public areas, which pose challenges for conventional facial recognition solutions. To solve this issue, our team plans to use a CNN-based deep learning model to achieve the masked face identity recognition task.

Our current progress is two-folds:

- We employed MTCNN (https://github.com/timesler/facenet-pytorch/blob/master/examples/infer.ipynb) -- a facial detection framework, to crop the target facial area. Facial detection in video streams are supported, which enables testing in real-time.
- We use Alexnet as a pretrained network followed by a simple CNN to perform the (unmasked) human facial recognition task. The dataset we use for now is the ORL dataset (https://www.kaggle.com/tavarez/the-orl-database-for-training-and-testing).

Our plan for the next stage:

- In the next stage, we plan to crop the eye and forehead area with MTCNN and use this to train our neural network to do the masked facial recognition task. The most important idea is to exclude mask in the input feature so as not to let the neural network to learn the shape of the mask instead of the user identity.
- We also notice there are some traditional computer vision algorithms that could preprocess (e.g. Autolevel) the image or perform facial recognition tasks (e.g. local binary pattern). Therefore, we are also interested in whether incorporating these methods into our DNN could improve the model performance.



Part 1: Extract facial pictures from webcam streaming for ID recognition

Install necessary packages

In [1]:
```
%%shell
pip install facenet-pytorch
pip install mmcv
```

```
Collecting facenet-pytorch
  Downloading facenet_pytorch-2.5.2-py3-none-any.whl (1.9 MB)
     |████████████████████████████████| 1.9 MB 11.2 MB/s
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packag
es (from facenet-pytorch) (7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
s (from facenet-pytorch) (1.21.5)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-p
ackages (from facenet-pytorch) (0.11.1+cu111)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
ages (from facenet-pytorch) (2.23.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
dist-packages (from requests->facenet-pytorch) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
7/dist-packages (from requests->facenet-pytorch) (2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
r/local/lib/python3.7/dist-packages (from requests->facenet-pytorch) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->facenet-pytorch) (2.10)
Requirement already satisfied: torch==1.10.0 in /usr/local/lib/python3.7/dist
-packages (from torchvision->facenet-pytorch) (1.10.0+cu111)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/
dist-packages (from torch==1.10.0->torchvision->facenet-pytorch) (3.10.0.2)
Installing collected packages: facenet-pytorch
Successfully installed facenet-pytorch-2.5.2
Collecting mmcv
  Downloading mmcv-1.4.6.tar.gz (438 kB)
     |████████████████████████████████| 438 kB 6.4 MB/s
Collecting addict
  Downloading addict-2.4.0-py3-none-any.whl (3.8 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
s (from mmcv) (1.21.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-pac
kages (from mmcv) (21.3)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packag
es (from mmcv) (7.1.2)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packag
es (from mmcv) (3.13)
Collecting yapf
  Downloading yapf-0.32.0-py2.py3-none-any.whl (190 kB)
     |████████████████████████████████| 190 kB 22.3 MB/s
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from packaging->mmcv) (3.0.7)
Building wheels for collected packages: mmcv
  Building wheel for mmcv (setup.py) ... done
  Created wheel for mmcv: filename=mmcv-1.4.6-py2.py3-none-any.whl size=65604
5 sha256=a0a1c1f7a98b3a97f923d46ea0959ded6a0fec77aabf3648e91f3afcb839bda0
  Stored in directory: /root/.cache/pip/wheels/4b/dc/28/f5fdb35b7e1a5f50de1a9
5a49e5f661e4ffb10461d35974240
Successfully built mmcv
Installing collected packages: yapf, addict, mmcv
Successfully installed addict-2.4.0 mmcv-1.4.6 yapf-0.32.0
```

Out[1]:

Import Dependencies

In [2]:
```
from IPython.display import Javascript, Image
from IPython import display as dis
from google.colab.output import eval_js
from base64 import b64decode, b64encode
import numpy as np
import io
import html
import time
from IPython.core.display import Video
from facenet_pytorch import MTCNN
import torch
import mmcv, cv2
import PIL
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
```

Several Helper methods to create video streaming

In [3]:
```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  image_bytes = b64decode(js_reply.split(',')[1]) # decode base64 image
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8) # convert bytes to nu
mpy array
  img = cv2.imdecode(jpg_as_np, flags=1) # decode numpy array into OpenCV BGR
 image

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte str
ing to be overlayed on video stream
def bbox_to_bytes(ipt):
  # ipt: Numpy array (pixels) containing rectangle to overlay on video stream.
  # bytes: Base64 image byte string
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(ipt, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = '{}'.format((str(b64encode(iobuf.getvalue
()), 'utf-8')))

  return bbox_bytes

# JavaScript to properly create our live video stream using our webcam as inpu
t
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;
    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
       imgElement = null;
       captureCanvas = null;
       labelElement = null;
    }

    function onAnimationFrame() {
      if (!shutdown) {
        window.requestAnimationFrame(onAnimationFrame);
      }
      if (pendingResolve) {
```

```
      var result = "";
      if (!shutdown) {
        captureCanvas.getContext('2d').drawImage(video, 0, 0, 720, 720);
        result = captureCanvas.toDataURL('image/jpeg', 0.8)
      }
      var lp = pendingResolve;
      pendingResolve = null;
      lp(result);
    }
  }

  async function createDom() {
    if (div !== null) {
      return stream;
    }

    div = document.createElement('div');
    div.style.border = '2px solid black';
    div.style.padding = '3px';
    div.style.width = '100%';
    div.style.maxWidth = '600px';
    document.body.appendChild(div);

    const modelOut = document.createElement('div');
    modelOut.innerHTML = "<span>Status:</span>";
    labelElement = document.createElement('span');
    labelElement.innerText = 'No data';
    labelElement.style.fontWeight = 'bold';
    modelOut.appendChild(labelElement);
    div.appendChild(modelOut);

    video = document.createElement('video');
    video.style.display = 'block';
    video.width = div.clientWidth - 6;
    video.setAttribute('playsinline', '');
    video.onclick = () => { shutdown = true; };
    stream = await navigator.mediaDevices.getUserMedia(
        {video: { facingMode: "environment"}});
    div.appendChild(video);

    imgElement = document.createElement('img');
    imgElement.style.position = 'absolute';
    imgElement.style.zIndex = 1;
    imgElement.onclick = () => { shutdown = true; };
    div.appendChild(imgElement);

    const instruction = document.createElement('div');
    instruction.innerHTML = '<span style="color: red; font-weight: bold;">'
+ 'When finished, click here or on the video to stop this demo</span>';
    div.appendChild(instruction);
    instruction.onclick = () => { shutdown = true; };

    video.srcObject = stream;
    await video.play();

    captureCanvas = document.createElement('canvas');
    captureCanvas.width = 720; //video width; 1280
```

```
      captureCanvas.height = 720; //video height; 720
      window.requestAnimationFrame(onAnimationFrame);

      return stream;
    }
    async function stream_frame(label, imgData) {
      if (shutdown) {
        removeDom();
        shutdown = false;
        return '';
      }

      var preCreate = Date.now();
      stream = await createDom();

      var preShow = Date.now();
      if (label != "") {
        labelElement.innerHTML = label;
      }

      if (imgData != "") {
        var videoRect = video.getClientRects()[0];
        imgElement.style.top = videoRect.top + "px";
        imgElement.style.left = videoRect.left + "px";
        imgElement.style.width = videoRect.width + "px";
        imgElement.style.height = videoRect.height + "px";
        imgElement.src = imgData;
      }

      var preCapture = Date.now();
      var result = await new Promise(function(resolve, reject) {
        pendingResolve = resolve;
      });
      shutdown = false;

      return {'create': preShow - preCreate,
              'show': preCapture - preShow,
              'capture': Date.now() - preCapture,
              'img': result};
    }
    ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data
```

Constructing a faster version of face detection model for potential higher throughput demand

In [4]:
```
class FastMTCNN(object):
    """Fast MTCNN implementation."""

    def __init__(self, stride, resize=1, *args, **kwargs):
        """Constructor for FastMTCNN class.

        Arguments:
            stride (int): The detection stride. Faces will be detected every `
stride` frames
                and remembered for `stride-1` frames.

        Keyword arguments:
            resize (float): Fractional frame scaling. [default: {1}]
            *args: Arguments to pass to the MTCNN constructor. See help(MTCN
N).
            **kwargs: Keyword arguments to pass to the MTCNN constructor. See
 help(MTCNN).
        """
        self.stride = stride
        self.resize = resize
        self.mtcnn = MTCNN(*args, **kwargs)

    def __call__(self, frames):
        """Detect faces in frames using strided MTCNN."""
        if self.resize != 1:
            frames = [
                cv2.resize(f, (int(f.shape[1] * self.resize), int(f.shape[0] *
self.resize)))
                    for f in frames
            ]

        boxes, probs = self.mtcnn.detect(frames[::self.stride])

        faces = []
        for i, frame in enumerate(frames):
            box_ind = int(i / self.stride)
            if boxes[box_ind] is None:
                continue
            for box in boxes[box_ind]:
                box = [int(b) for b in box]
                faces.append(frame[box[1]:box[3], box[0]:box[2]])

        return faces
```

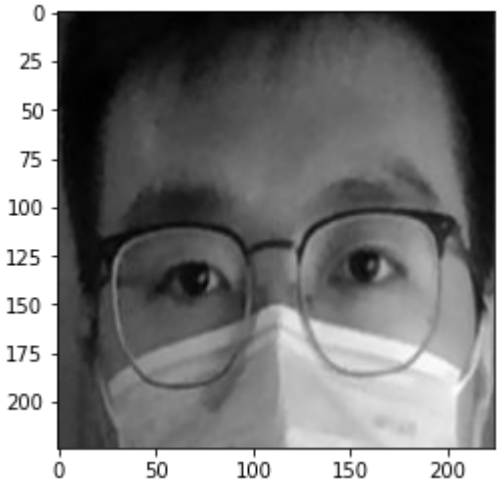
Face detection model implementations

In [5]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print('Running on device: {}'.format(device))
# currently using, for face detection
mtcnn = MTCNN(keep_all=True, min_face_size=224, device=device)
# A faster model
mtcnn_fast = FastMTCNN(keep_all=True, min_face_size=224, device=device,stride=
4)
```

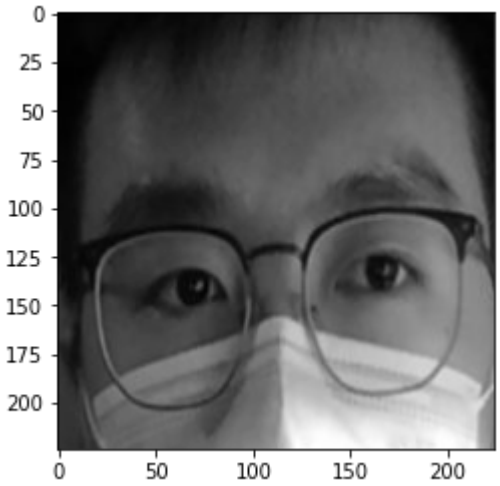
```
Running on device: cuda:0
```

In [14]:
```
from torch.functional import Tensor
video_stream()
label_html = 'Capturing...'
bbox = ''
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break
    # convert JS response to OpenCV Image
    img = js_to_image(js_reply["img"])

    rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #frame_fast = torch.from_numpy(rgb_img)
    #frame_fast = torch.stack([frame_fast], dim=0) #Prepared input frame for m
tcnn_fast model
    frame = Image.fromarray(rgb_img) #Input frame for mtcnn face detection mod
el



    faces, _ = mtcnn.detect(frame) #return coordinates of faces
    #faces_fast = mtcnn_fast(frame_fast) #higher throughput, return captured f
aces
    #plt.imshow(faces_fast[0].detach().numpy(), cmap = 'gray')
    #plt.show()

    bbox_array = np.zeros([720,720,4], dtype=np.uint8) #Transparent overlay fo
r drawing bounding box

    if faces is not None:
      for (x,y,w,h) in faces:
        x_w_diff = int(w-x)
        y_h_diff = int(h-y)
        if x_w_diff > 223. and y_h_diff >223.:
          x_w_mid = int(x+(w-x)/2)
          y_h_mid = int(y+(h-y)/2)
          selected_x = x_w_mid - 112
          selected_w = x_w_mid + 112
          selected_y = y_h_mid - 112 - 50
          selected_h = y_h_mid + 112 - 50

          #selected_img = img[int(y):int(y) + x_w_diff,int(x):int(x)+ x_w_dif
f,:]
          selected_img = img[selected_y:selected_h, selected_x:selected_w, :]
          selected_frame = cv2.cvtColor(selected_img, cv2.COLOR_BGR2GRAY)
          plt.imshow(selected_frame, cmap = 'gray')
          plt.show()
          print(selected_frame.shape)

          input_for_face_recognition = torch.from_numpy(selected_frame)
          input_for_face_recognition = torch.stack([input_for_face_recognition
], dim=0) #Prepared input in tensor format


          # Facial recognition process will be placed here
```

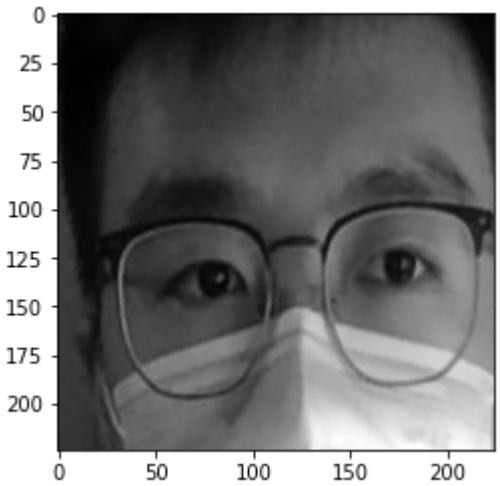
```
            #
            box_height = (w,int(h-(20)))
            bbox_array = cv2.rectangle(bbox_array,(x,y), box_height, (0,255,0),
2) #Bounding box size and colour
      bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
      bbox_bytes = bbox_to_bytes(bbox_array) # converting overlay of bbox into
bytes
      bbox =bbox_bytes # update bbox for the next frame
```



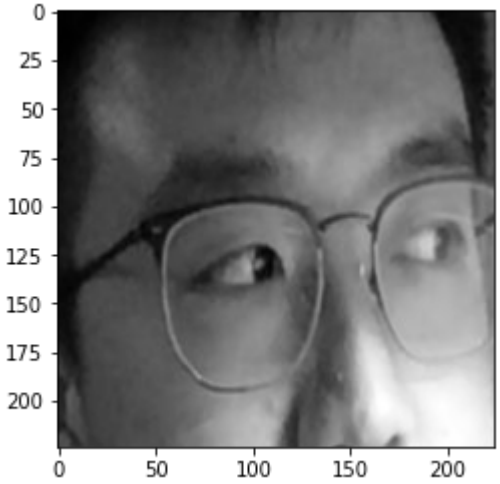

```
(224, 224)
```



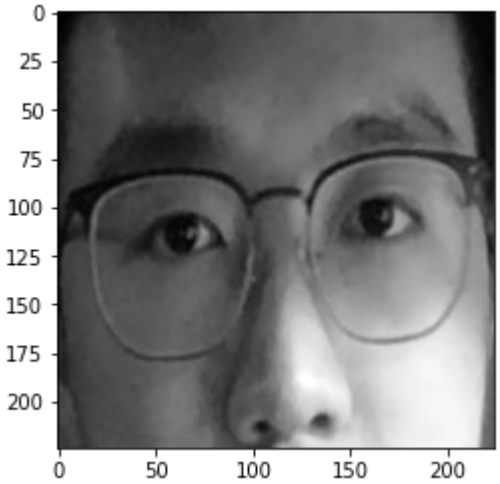

```
(224, 224)
```




```
(224, 224)
```



(224, 224)



(224, 224)

In [ ]:
```
%%shell
jupyter nbconvert --to html Project_Face_detection.ipynb
```

Part 2: Model Training

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [2]:
```
from os import listdir
from os.path import isfile, join
import os
import re
import shutil
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
from torchvision import datasets, transforms
import torch.optim as optim
import matplotlib.pyplot as plt
from tqdm import tqdm
from PIL import Image
import matplotlib.pyplot as plt

use_cuda = True
```

In [3]:
```
def split_data(dataset_path, batch_size = 64):
  torch.manual_seed(1)
  np.random.seed(1)

  transform = transforms.Compose([transforms.Resize((224,224)),
                  # transforms.ColorJitter(), # change image color
                  transforms.RandomHorizontalFlip(), # flip images
                  # transforms.RandomAffine(30), # Random affine transformatio
n of the image keeping center invariant.
                  # transforms.CenterCrop(224),
                  transforms.ToTensor()])

  training_dataset = datasets.ImageFolder(root = dataset_path, transform=trans
form)

  indices = list(range(len(training_dataset)))

  train_indices = []

  val_indices = []

# split data with ratio 8:2
  for i in indices:
    if i % 10 > 7:
      val_indices.append(i)
    else:
      train_indices.append(i)

  np.random.shuffle(train_indices)

  np.random.shuffle(val_indices)

  train_sampler = SubsetRandomSampler(train_indices)
  val_sampler = SubsetRandomSampler(val_indices)

  train_loader, val_loader = torch.utils.data.DataLoader(training_dataset, bat
ch_size=batch_size, sampler=train_sampler), \
                torch.utils.data.DataLoader(training_dataset, batch_size=batch
_size, sampler=val_sampler)

  return train_loader, val_loader
```

In [4]:
```
# set data path
ori_path = '/content/drive/MyDrive/ut/MIE1517_project_dataset/orl/images/'
group_folder_path = '/content/drive/MyDrive/ut/MIE1517_project_dataset/orl/gro
uped_images/'
files = [f for f in listdir(ori_path) if isfile(join(ori_path, f))]
```

In [5]:
```
# extract images and labels and store into target folder
for single_file in files:
  group = single_file.split("_")[1].split('.')[0]
  folder_path = group_folder_path + group

  if not os.path.exists(folder_path):
    print("Directory '{}' is created!".format(group))
    os.makedirs(folder_path)

  shutil.copyfile(ori_path + single_file, folder_path + '/' + single_file)
```

In [6]:
```
group_folder_path = '/content/drive/MyDrive/ut/MIE1517_project_dataset/orl/grouped_images/'

train_loader, val_loader = split_data(group_folder_path, batch_size = 1)
```

In [7]:
```
print('Training images we have:', len(train_loader))
print('Validation images we have:', len(val_loader))
```

```
Training images we have: 328
Validation images we have: 82
```

In [8]:
```
# define model to adapt Alex Model
class AlexNetModel(nn.Module):

    def __init__(self):
        super(AlexNetModel, self).__init__()
        self.name = 'AlexNetModel'
        self.conv1 = nn.Conv2d(256, 30, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(120, 64)
        self.fc2 = nn.Linear(64, 41)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 120)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

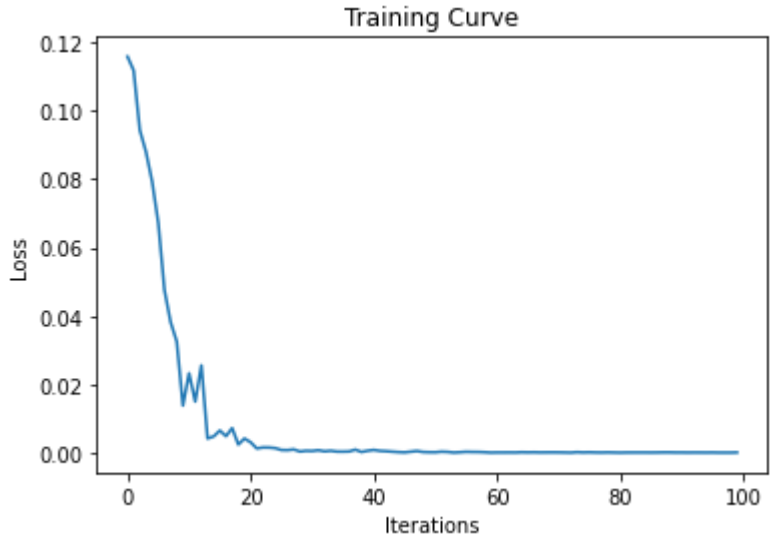
In [11]:
```
# function to save checkpoint
def save_checkpoint(model, batch_size, lr, epoch):
  model_path = "{}_{}_{}_{}".format(model.name, batch_size, lr, epoch)
  torch.save(model, model_path)
  print('Checkpoint of {} has been stored successfully!',format(model_path))

def get_accuracy(model, data_loader, grey_images_flag = False):
  correct = 0
  total = 0

  alexnet = torchvision.models.alexnet(pretrained=True)

  for imgs, labels in data_loader:
    ###############################################
    #To Enable GPU Usage

    if grey_images_flag:

      grey_images = torchvision.transforms.Grayscale()(imgs)

      imgs = torch.tensor(np.tile(grey_images, [1,3,1,1]))

    imgs = alexnet.features(imgs)

    if use_cuda and torch.cuda.is_available():
      imgs = imgs.cuda()
      labels = labels.cuda()
    ###############################################

    output = model(imgs)

    #select index with maximum prediction score
    pred = output.max(1, keepdim=True)[1]
    correct += pred.eq(labels.view_as(pred)).sum().item()
    total += imgs.shape[0]
  return correct / total

def train(model, dataset_path, opt, batch_size = 64, learning_rate=0.01, epoch
s=30, grey_images_flag = False):

  ###############################################
  # input: grey_images_flag: boolean if gray image conversion is needed
  ###############################################

  train_loader, val_loader = split_data(dataset_path, batch_size = batch_size)

  criterion = nn.CrossEntropyLoss()

  optimizer = opt(model.parameters(), lr=learning_rate)

  iters, losses, train_acc, val_acc = [], [], [], []

  alexnet = torchvision.models.alexnet(pretrained=True)

  n = 0 # the number of iterations
  for epoch in range(epochs):
```

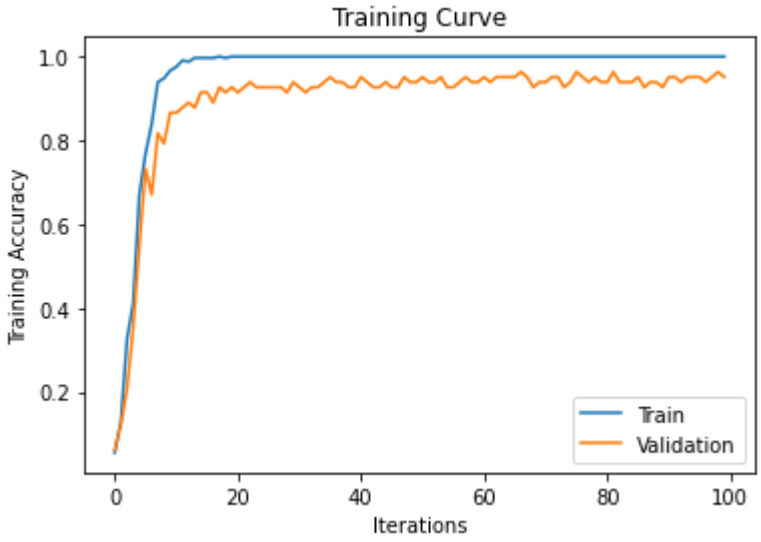
```
    for imgs, labels in tqdm(iter(train_loader)):

      if grey_images_flag:

        grey_images = torchvision.transforms.Grayscale()(imgs)

        imgs = torch.tensor(np.tile(grey_images, [1,3,1,1]))

      imgs = alexnet.features(imgs)

      ##############################################
      #To Enable GPU Usage
      if use_cuda and torch.cuda.is_available():
        imgs = imgs.cuda()
        labels = labels.cuda()
      ##############################################

      output = model(imgs)
      loss = criterion(output, labels)
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()

    # save the current training information
    iters.append(n)
    losses.append(float(loss)/batch_size)
    train_acc.append(get_accuracy(model, train_loader, grey_images_flag = grey
_images_flag))
    val_acc.append(get_accuracy(model, val_loader, grey_images_flag = grey_ima
ges_flag))
    n += 1

    if ((epoch+1) % 25 == 0):
      save_checkpoint(model, batch_size, learning_rate, epoch+1)

  # plotting
  plt.title("Training Curve")
  plt.plot(iters, losses, label="Train")
  plt.xlabel("Iterations")
  plt.ylabel("Loss")
  plt.show()

  plt.title("Training Curve")
  plt.plot(iters, train_acc, label="Train")
  plt.plot(iters, val_acc, label="Validation")
  plt.xlabel("Iterations")
  plt.ylabel("Training Accuracy")
  plt.legend(loc='best')
  plt.show()

  print("Final Training Accuracy: {}".format(train_acc[-1]))
  print("Final Validation Accuracy: {}".format(val_acc[-1]))
```

In [12]:
```
model = AlexNetModel()

if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

train(model, group_folder_path, opt = optim.Adam, batch_size = 32, learning_ra
te=0.001, epochs=100, grey_images_flag = False)
```

```
CUDA is not available.  Training on CPU ...

100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_25

100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_50
```

```
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_75

100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:23<00:00,  2.16s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.17s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_100
```






```
Final Training Accuracy: 1.0
Final Validation Accuracy: 0.9512195121951219
```

Future plan

As training and testing dataset is clean and standard, adding noise is not neccessary for current phase but could be benefit to real world data. Also, image processing method such as auto-level to scale the distribution of pixels for each channel could also improve our model performance for real world data.

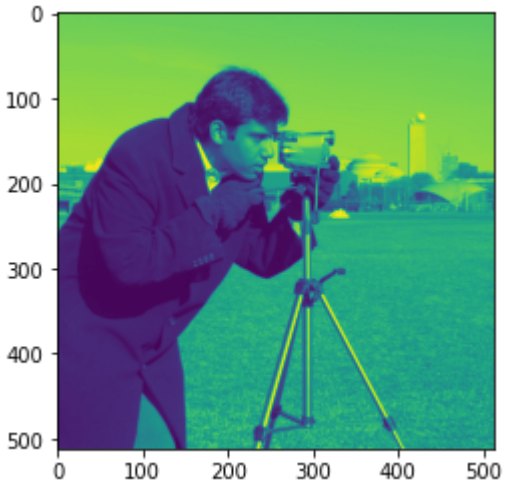
Example:

In [20]:
```
from skimage import data
from skimage.morphology import disk
from skimage.filters.rank import autolevel
from PIL import Image
```

In [16]:
```
img = data.camera()
auto = autolevel(img, disk(5))
```

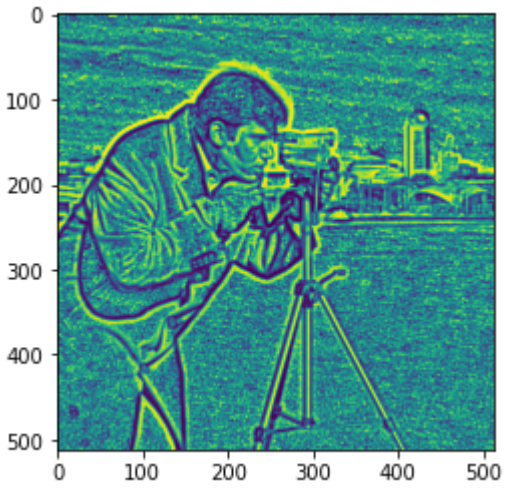
In [17]:
```
plt.imshow(img)
```

Out[17]: <matplotlib.image.AxesImage at 0x7f5059ed0950>



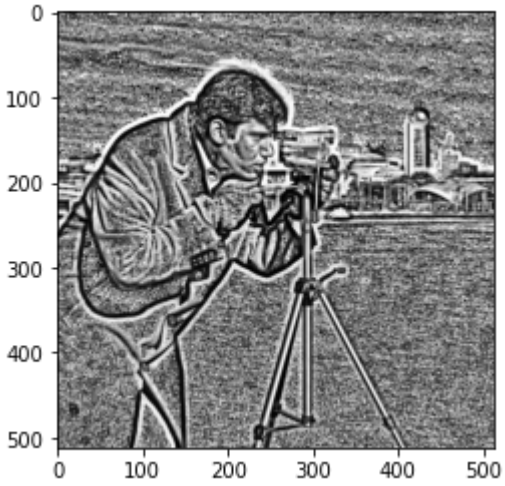
In [18]:
```
plt.imshow(auto)
```

Out[18]: <matplotlib.image.AxesImage at 0x7f50585b2a90>



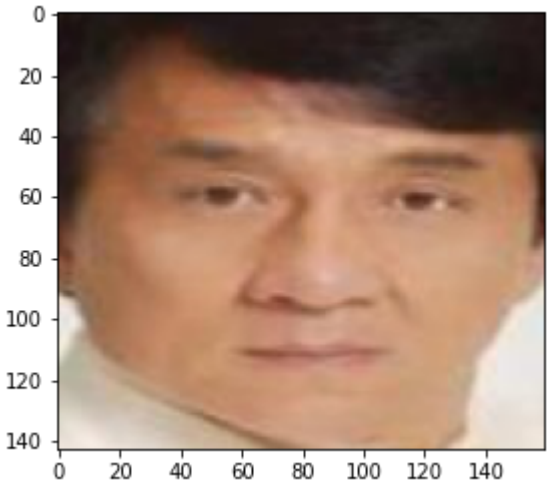
In [19]:
```
plt.imshow(auto, cmap = "gray")
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x7f505851ff10>
```



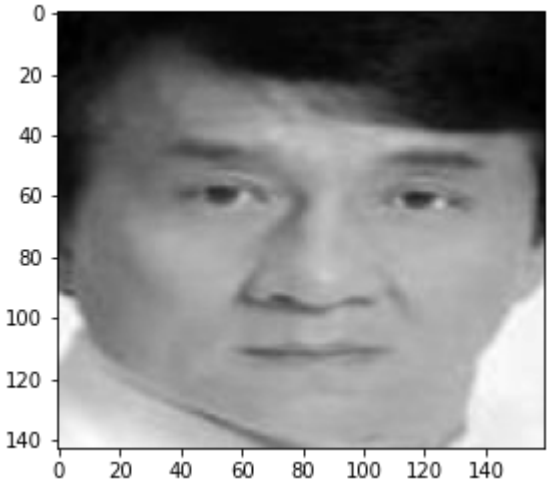
In [40]:
```
image = Image.open('img_1.jpg')
plt.imshow(image)
```

Out[40]:
```
<matplotlib.image.AxesImage at 0x7f5058021990>
```



In [46]:
```
image = torchvision.transforms.Grayscale()(image)
plt.imshow(image, cmap='gray')
```

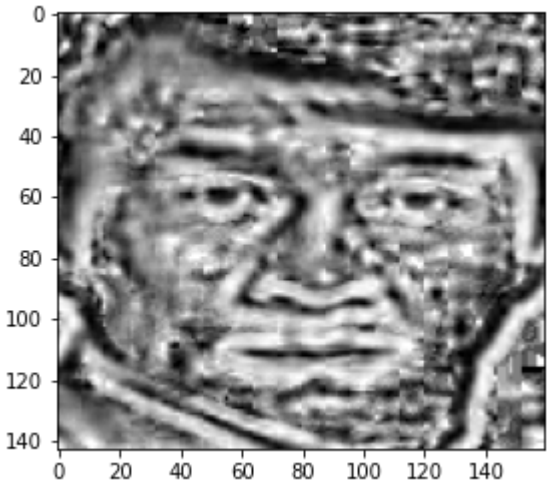
Out[46]: <matplotlib.image.AxesImage at 0x7f5057f14ad0>



In [59]:
```
image = np.asanyarray(image)
image_new = image.copy()
```

In [65]:
```
plt.imshow(autolevel(image_new, disk(5)), cmap='gray')
```

Out[65]: <matplotlib.image.AxesImage at 0x7f5057c1c1d0>



In [67]:
```
def auto_level_img(imgs):
  ret_imgs = []
  imgs = torchvision.transforms.Grayscale()(imgs)
  for i in imgs:
    ret_imgs.append(autolevel(image_new, disk(5)))
```

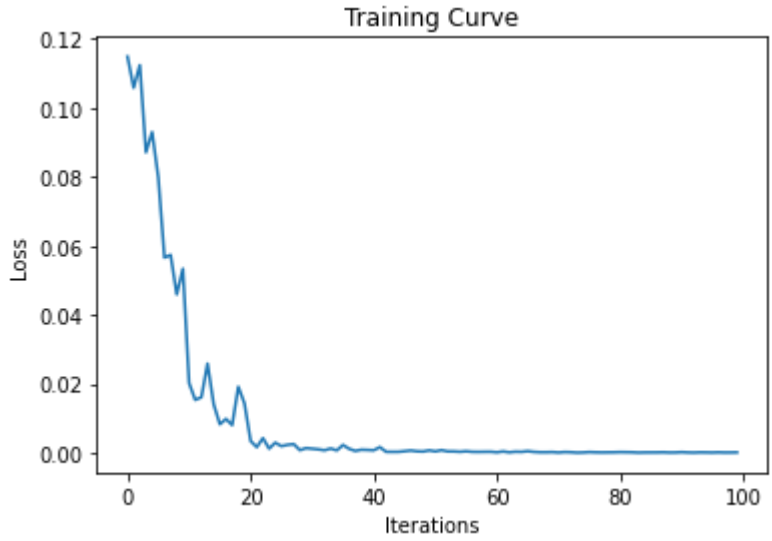
In [68]:
```
def train_autolevel(model, dataset_path, opt, batch_size = 64, learning_rate=
0.01, epochs=30, grey_images_flag = False):

  ##############################################
  # input: grey_images_flag: boolean if gray image conversion is needed
  ##############################################

  train_loader, val_loader = split_data(dataset_path, batch_size = batch_size)

  criterion = nn.CrossEntropyLoss()

  optimizer = opt(model.parameters(), lr=learning_rate)

  iters, losses, train_acc, val_acc = [], [], [], []

  alexnet = torchvision.models.alexnet(pretrained=True)

  n = 0 # the number of iterations
  for epoch in range(epochs):
    for imgs, labels in tqdm(iter(train_loader)):

      if grey_images_flag:

        grey_images = torchvision.transforms.Grayscale()(imgs)

        grey_images = auto_level_img(grey_images)

        imgs = torch.tensor(np.tile(grey_images, [1,3,1,1]))

      imgs = alexnet.features(imgs)

      ##############################################
      #To Enable GPU Usage
      if use_cuda and torch.cuda.is_available():
        imgs = imgs.cuda()
        labels = labels.cuda()
      ##############################################

      output = model(imgs)
      loss = criterion(output, labels)
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()

    # save the current training information
    iters.append(n)
    losses.append(float(loss)/batch_size)
    train_acc.append(get_accuracy(model, train_loader, grey_images_flag = grey
_images_flag))
    val_acc.append(get_accuracy(model, val_loader, grey_images_flag = grey_ima
ges_flag))
    n += 1

    if ((epoch+1) % 25 == 0):
      save_checkpoint(model, batch_size, learning_rate, epoch+1)
```

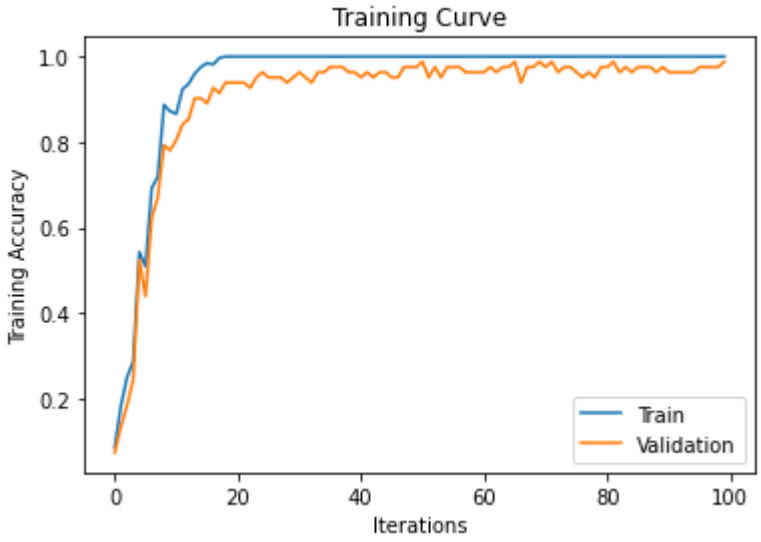
```
# plotting
plt.title("Training Curve")
plt.plot(iters, losses, label="Train")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

plt.title("Training Curve")
plt.plot(iters, train_acc, label="Train")
plt.plot(iters, val_acc, label="Validation")
plt.xlabel("Iterations")
plt.ylabel("Training Accuracy")
plt.legend(loc='best')
plt.show()

print("Final Training Accuracy: {}".format(train_acc[-1]))
print("Final Validation Accuracy: {}".format(val_acc[-1]))
```

In [69]:
```
model_auto_level = AlexNetModel()

if use_cuda and torch.cuda.is_available():
  model_auto_level.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

train(model_auto_level, group_folder_path, opt = optim.Adam, batch_size = 32,
learning_rate=0.001, epochs=100, grey_images_flag = True)
```

```
CUDA is not available.  Training on CPU ...

100%|██████████| 11/11 [00:25<00:00,  2.32s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_25

100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.25s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:23<00:00,  2.18s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.19s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_50
```

```
100%|██████████| 11/11 [00:24<00:00,  2.20s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_75

100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.21s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.24s/it]
100%|██████████| 11/11 [00:24<00:00,  2.23s/it]
100%|██████████| 11/11 [00:24<00:00,  2.24s/it]
100%|██████████| 11/11 [00:24<00:00,  2.22s/it]
100%|██████████| 11/11 [00:24<00:00,  2.24s/it]

Checkpoint of {} has been stored successfully! AlexNetModel_32_0.001_100
```






```
Final Training Accuracy: 1.0
Final Validation Accuracy: 0.9878048780487805
```

In [ ]: