c)

Neural Network Design:

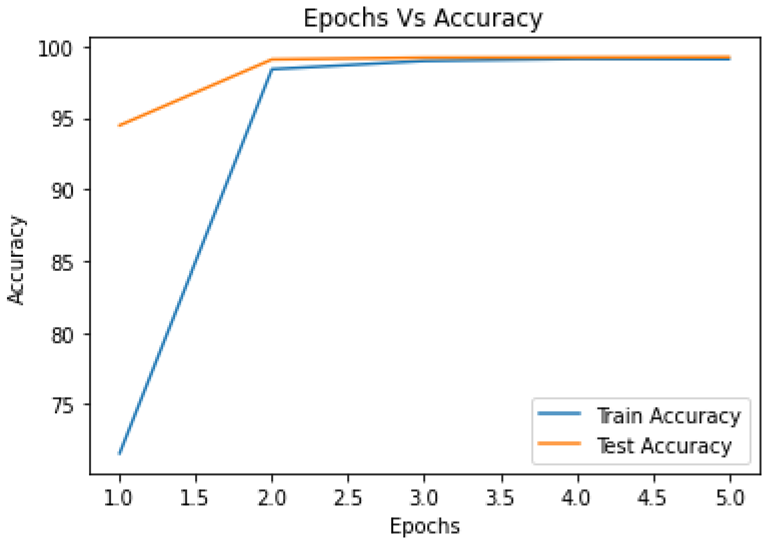
I followed the VGG11 architecture except for the last fully connected layers as the model was exceeding 50MB.

What did not work:
I tried a simpler architecture and was not getting high accuracy hence I used a complicated architecture, other than that I have tried different parameters. I used a learning rate of 0.001 but the loss was not decreasing, it was fluctuating.

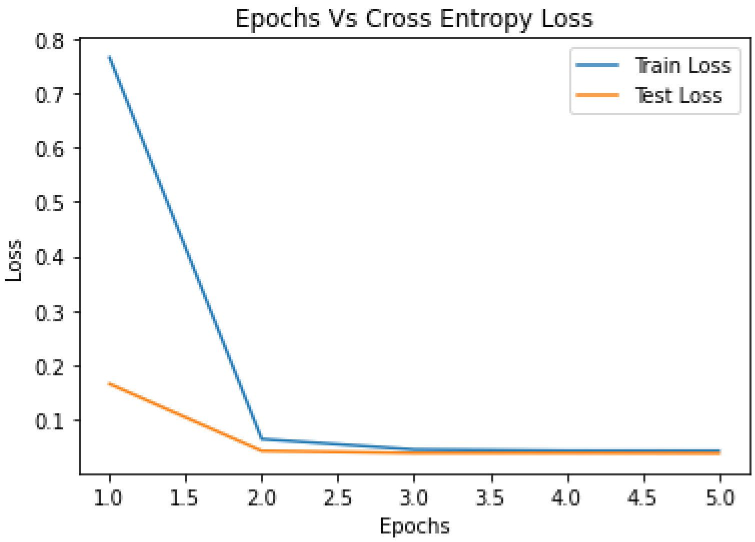
What worked:
Details of the Network:

Batch size = 64
Epochs = 5
Optimizer = Adam used amsgrad version
Learning rate = 0.0001
StepLr gamma = 0.1 (default value)

Graph of Epochs vs Accuracy



Graphs of Epochs vs Cross Entropy Loss



Code:

0601-659026651-Chintakunta.py:

```
# -*- coding: utf-8 -*-
"""HW5.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/166h0Omhz-At1oGYpAIb12yq5_NQfVWc7
"""

import os
import shutil
import glob
import random
import torch
from torchvision import transforms
from torch.utils.data import DataLoader
from torch import nn
from torchvision import datasets
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
```

```
import matplotlib.pyplot as plt
import math

# Code to split the dataset into training and test set
# the training and test folder are created as required by ImageFolder in the format
training_images/class/.*png

random.seed(42)
torch.manual_seed(42)
# path = os.pardir
# classes = ["Circle", "Square", "Octagon", "Heptagon", "Nonagon", "Star", "Hexagon",
"Pentagon", "Triangle"]
#
# training_files = glob.glob(os.path.join(path, "geometry_dataset/training_images/*.png"))
# test_files = glob.glob(os.path.join(path, "geometry_dataset/test_images/*.png"))
#
# for training_file, test_file in zip(training_files, test_files):
#     os.remove(training_file)
#     os.remove(test_file)
#
# os.mkdir(os.path.join(path, "geometry_dataset/training_images"))
# os.mkdir(os.path.join(path, "geometry_dataset/test_images"))
#
# for class_ in classes:
#     image_location = os.path.join(path, "geometry_dataset/output")
#     image_location = os.path.join(image_location, class_ + "*.png")
#     class_images = glob.glob(image_location)
#     random.shuffle(class_images)
#     train_directory = os.path.join(path, "geometry_dataset/training_images/"+str(class_))
#     test_directory = os.path.join(path, "geometry_dataset/test_images/"+str(class_))
#     os.mkdir(train_directory)
#     os.mkdir(test_directory)
#     for train_images in class_images[0:8000]:
#         shutil.copy(train_images, "geometry_dataset/training_images/"+str(class_))
#     for test_images in class_images[8000:]:
#         shutil.copy(test_images, "geometry_dataset/test_images/"+str(class_))

# Code to calculate the mean and standard deviation of the dataset
# transform = transforms.Compose([transforms.ToTensor()])
# train_dataset =
datasets.ImageFolder(os.path.join(os.getcwd(),"training_images"),transform=transform)
# train_loader = DataLoader(train_dataset,batch_size=64,shuffle=True)
# mean = 0.0
# for images, _ in train_loader:
```

```
#     batch_samples = images.size(0)
#     images = images.view(batch_samples, images.size(1), -1)
#     mean += images.mean(2).sum(0)
# mean = mean / len(train_loader.dataset)
#
# var = 0.0
# for images, _ in train_loader:
#     batch_samples = images.size(0)
#     images = images.view(batch_samples, images.size(1), -1)
#     var += ((images - mean.unsqueeze(1))**2).sum([0,2])
# std = torch.sqrt(var / (len(train_loader.dataset)*200*200))

# Neural Network Architecture used
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1,1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 1,1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(128, 256, 3, 1,1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, 1,1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(256, 512, 3, 1,1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1,1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(512, 512, 3, 1,1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1,1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(18432, 128),
            nn.ReLU()
        )
        self.fc2 = nn.Linear(128, 9)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.fc1(torch.flatten(x,start_dim=1))
        x = self.fc2(x)
        return x


def train(batch_size, model, device, train_loader, optimizer, epoch):
    model.train()
    tot_loss = 0
    correct = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = torch.nn.CrossEntropyLoss()(output, target)
        loss.backward()
        optimizer.step()

        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

        tot_loss = tot_loss + loss.item()
        if batch_idx % 50 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}, Accuracy: {:.2f}%'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), tot_loss/(batch_idx+1),
100.0*correct/((batch_idx+1)*batch_size)))

    loss = tot_loss / len(train_loader)
    acc = 100.0 * correct / (len(train_loader) * batch_size)
    print('End of Epoch: {}'.format(epoch))
    print('Training Loss: {:.6f}, Training Accuracy: {:.2f}%'.format(loss, acc))
    return loss,acc


def test(batch_size, model, device, test_loader):
    model.eval()
    tot_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            tot_loss += torch.nn.CrossEntropyLoss()(output, target).item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    loss = tot_loss / len(test_loader)
    acc = 100.0 * correct / (len(test_loader) * batch_size)
    print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(loss, acc))
    return loss, acc


batch_size = 64
epochs = 5
mean, std = (torch.tensor([0.4976, 0.4975, 0.4984]), torch.tensor([0.2877, 0.2891, 0.2883]))
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean,std)])
train_dataset =
datasets.ImageFolder(os.path.join(os.pardir,"geometry_dataset/training_images"),transform=transform)
test_dataset =
datasets.ImageFolder(os.path.join(os.pardir,"geometry_dataset/test_images"),transform=transform)
train_loader = DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
test_loader = DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Net().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.0001,amsgrad=True)

train_losses,train_accuracy = [],[]
test_losses,test_accuracy = [],[]
scheduler = StepLR(optimizer, step_size=1)
for epoch in range(1, epochs + 1):
    train_loss,train_acc = train(batch_size, model, device, train_loader, optimizer, epoch)
    test_loss, test_acc = test(batch_size,model, device, test_loader)
    train_losses.append(train_loss)
    train_accuracy.append(train_acc)
    test_losses.append(test_loss)
    test_accuracy.append(test_acc)
    scheduler.step()

    if test_loss < 1e-4 or math.isclose(test_acc, 100.0):
        break

torch.save(model.state_dict(), "0602-659026651-Chintakunta.pt")

epochs = list(range(1,epochs+1))
plt.plot(epochs,train_accuracy,label="Train Accuracy")
plt.plot(epochs,test_accuracy,label="Test Accuracy")
plt.title("Epochs Vs Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.plot(epochs,train_losses,label="Train Loss")
plt.plot(epochs,test_losses,label="Test Loss")
plt.title("Epochs Vs Cross Entropy Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

0603-659026651-Chintakunta.py:

```
# -*- coding: utf-8 -*-
"""HW5_1.ipynb
```

```
Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1NqUD33q3IfQjgwocMkjGkyVOV37Zo_w8
"""

import os
import torch
from torchvision import transforms
from torch import nn
from PIL import Image


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(128, 256, 3, 1, 1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, 1, 1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(256, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(512, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(18432, 128),
            nn.ReLU()
        )
        self.fc2 = nn.Linear(128, 9)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.fc1(torch.flatten(x, start_dim=1))
        x = self.fc2(x)
        return x


mean, std = (torch.tensor([0.4976, 0.4975, 0.4984]), torch.tensor([0.2877, 0.2891, 0.2883]))
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,),
(0.3081,))])
dataset_path = os.path.join(os.pardir, "geometry_dataset\sample_dataset") # change the path to
folder where the test file is located
test_dataset = sorted(os.listdir(dataset_path))
model = Net()
model_path = os.path.join(os.getcwd(), "0602-659026651-Chintakunta.pt")
model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
model.eval()

classes = {0: 'Circle', 1: 'Heptagon', 2: 'Hexagon', 3: 'Nonagon', 4: 'Octagon', 5: 'Pentagon', 6:
'Square', 7: 'Star', 8: 'Triangle'}

for X in test_dataset:
    image = Image.open(os.path.join(dataset_path, X))
    image = transform(image)
    pred = model(image.unsqueeze(dim=0))
    label = classes[pred.argmax().item()]
    print("the predicted output of " + str(X) + " is " + label)
```