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

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from torchvision import datasets
import torchvision.transforms as transforms
import torch.nn.functional as F
```

In [ ]:
```
# Processing Numerical Data

rawdata = pd.read_csv('testHouses.csv')

Xraw = np.column_stack((rawdata['Id'].values,
                        rawdata['City'].values,
                        rawdata['Bed'].values,
                        rawdata['Bath'].values,
                        rawdata['Sqft'].values,
                        rawdata['Price'].values))

Xraw_train = Xraw[0:2000,:]
city_data_train = Xraw_train[:,1]
bdrm_data_train = Xraw_train[:,2]
bath_data_train = Xraw_train[:,3]
sqft_data_train = Xraw_train[:,4]
price_data_train = Xraw_train[:,5]

city_train_norm = city_data_train/np.max(city_data_train)
bdrm_train_norm = bdrm_data_train/np.max(bdrm_data_train)
bath_train_norm = bath_data_train/np.max(bath_data_train)
sqft_train_norm = sqft_data_train/np.max(sqft_data_train)
price_train_norm = price_data_train/np.max(price_data_train)

city_train = torch.from_numpy(city_train_norm).float()
bdrm_train = torch.from_numpy(bdrm_train_norm).float()
bath_train = torch.from_numpy(bath_train_norm).float()
sqft_train = torch.from_numpy(sqft_train_norm).float()
price_train = torch.from_numpy(price_train_norm).float()

meta_train = torch.stack((city_train, bdrm_train, bath_train, sqft_train),dim=1)

Xraw_val = Xraw[2000:2500,:]
city_data_val = Xraw_val[:,1]
bdrm_data_val = Xraw_val[:,2]
bath_data_val = Xraw_val[:,3]
sqft_data_val = Xraw_val[:,4]
price_data_val = Xraw_val[:,5]

city_val_norm = city_data_val/np.max(city_data_val)
bdrm_val_norm = bdrm_data_val/np.max(bdrm_data_val)
bath_val_norm = bath_data_val/np.max(bath_data_val)
sqft_val_norm = sqft_data_val/np.max(sqft_data_val)
price_val_norm = price_data_val/np.max(price_data_val)

city_val = torch.from_numpy(city_val_norm).float()
bdrm_val = torch.from_numpy(bdrm_val_norm).float()
bath_val = torch.from_numpy(bath_val_norm).float()
sqft_val = torch.from_numpy(sqft_val_norm).float()
price_val = torch.from_numpy(price_val_norm).float()

meta_val = torch.stack((city_val, bdrm_val, bath_val, sqft_val),dim=1)

Xraw_test = Xraw[2500:3000,:]
city_data_test = Xraw_test[:,1]
bdrm_data_test = Xraw_test[:,2]
bath_data_test = Xraw_test[:,3]
sqft_data_test = Xraw_test[:,4]
price_data_test = Xraw_test[:,5]

city_test_norm = city_data_test/np.max(city_data_test)
bdrm_test_norm = bdrm_data_test/np.max(bdrm_data_test)
bath_test_norm = bath_data_test/np.max(bath_data_test)
sqft_test_norm = sqft_data_test/np.max(sqft_data_test)
price_test_norm = price_data_test/np.max(price_data_test)

city_test = torch.from_numpy(city_test_norm).float()
bdrm_test = torch.from_numpy(bdrm_test_norm).float()
bath_test = torch.from_numpy(bath_test_norm).float()
sqft_test = torch.from_numpy(sqft_test_norm).float()
price_test = torch.from_numpy(price_test_norm).float()

meta_test = torch.stack((city_test, bdrm_test, bath_test, sqft_test),dim=1)
```

In [ ]:
```
# Processing Images and Data

batch_size = 256

transformation = transforms.Compose([
        transforms.Resize([100, 157]),
        transforms.CenterCrop(100),
        transforms.ToTensor()])

train_images = datasets.ImageFolder('/content/drive/My Drive/socal_pics/train',
                    transform=transformation)
train_load = torch.utils.data.DataLoader(train_images, shuffle=False,
                         batch_size=batch_size)

val_images = datasets.ImageFolder('/content/drive/My Drive/socal_pics/val',
                    transform=transformation)
val_load = torch.utils.data.DataLoader(val_images, shuffle=False,
                         batch_size=batch_size)

test_images = datasets.ImageFolder('/content/drive/My Drive/socal_pics/test',
                    transform=transformation)
test_load = torch.utils.data.DataLoader(test_images, shuffle=False,
                         batch_size=batch_size)

def get_batch_train(batch_size,which_batch,
                    array_len=len(price_train)):

    num_batches = int(np.floor(array_len/batch_size))

    batch_y = []
    batch_meta = []
    for i in range(num_batches+1):
        batch_y_train = price_train[i*batch_size:(i+1)*batch_size]
        batch_meta_train = meta_train[i*batch_size:(i+1)*batch_size,:]
        batch_y.append(batch_y_train)
        batch_meta.append(batch_meta_train)

    price_data_train = torch.FloatTensor(batch_y[which_batch])
    metadata_train = torch.FloatTensor(batch_meta[which_batch])
    return price_data_train,metadata_train

def get_batch_val(batch_size,which_batch,
                  array_len=len(price_val)):

    num_batches = int(np.floor(array_len/batch_size))

    batch_y = []
    batch_meta = []
    for i in range(num_batches+1):
        batch_y_val = price_val[i*batch_size:(i+1)*batch_size]
        batch_meta_val = meta_val[i*batch_size:(i+1)*batch_size,:]
        batch_y.append(batch_y_val)
        batch_meta.append(batch_meta_val)

    price_data_val = torch.FloatTensor(batch_y[which_batch])
    metadata_val = torch.FloatTensor(batch_meta[which_batch])
    return price_data_val,metadata_val

def get_batch_test(batch_size,which_batch,
                   array_len=len(price_test)):

    num_batches = int(np.floor(array_len/batch_size))

    batch_y = []
    batch_meta = []
    for i in range(num_batches+1):
        batch_y_test = price_test[i*batch_size:(i+1)*batch_size]
        batch_meta_test = meta_test[i*batch_size:(i+1)*batch_size,:]
        batch_y.append(batch_y_test)
        batch_meta.append(batch_meta_test)

    price_data_test = torch.FloatTensor(batch_y[which_batch])
    metadata_test = torch.FloatTensor(batch_meta[which_batch])
    return price_data_test,metadata_test
```

In [ ]:
```
# Architecture
class DualNet(torch.nn.Module):
    def __init__(self):
        super(DualNet, self).__init__()
        # Image CNN
        self.pool = torch.nn.MaxPool2d(2, 2)
        self.conv1 = torch.nn.Conv2d(3, 5, 5)
        self.conv2 = torch.nn.Conv2d(5, 10, 5)
        self.conv3 = torch.nn.Conv2d(10, 15, 5)
        self.fc1 = torch.nn.Linear(15 * 9 * 9, 140)
        self.fc2 = torch.nn.Linear(140, 60)

        # Number NN MLP
        # 4 inputs (meta data) to NN
        self.fc3 = torch.nn.Linear(4, 120)
        self.fc4 = torch.nn.Linear(120, 60)
        self.fc5 = torch.nn.Linear(90, 60)

        # Combine outputs from CNN and NN
        self.fc6 = torch.nn.Linear(120, 120)
        self.fc7 = torch.nn.Linear(120, 1) # 1 output -> price [0,1]

    def forward(self, x1, x2, x3):
        x1 = self.pool(F.relu(self.conv1(x1)))
        x1 = self.pool(F.relu(self.conv2(x1)))
        x1 = self.pool(F.relu(self.conv3(x1)))
        x1 = x1.view(-1, 15 * 9 * 9)
        x1 = F.relu(self.fc1(x1))
        x1 = F.relu(self.fc2(x1))

        x2 = x2.view(-1, 4)
        x2 = F.relu(self.fc3(x2))
        x2 = F.relu(self.fc4(x2))
        x2 = F.relu(self.fc5(x2))

        x3 = torch.cat((x1, x2), dim=1) #Concatenate outputs from the two networks into one!
        x3 = F.relu(self.fc6(x3))
        x3 = self.fc7(x3)

        return x3
```

In [ ]:
```
# training loop

learn_rate = 1e-3
num_epochs = 25

torch.manual_seed(10)

model = DualNet().to('cuda') #using cuda instead of cpu

criterion = torch.nn.MSELoss(reduction='mean')

optimizer = torch.optim.Adam(model.parameters(),
                             lr=learn_rate)

result_vals = torch.zeros(num_epochs,4)
count_train = 0
count_val = 0

print(' ')
print('epoch  | loss_train  | loss_val  | err_train  | err_val')
print('-------------------------------------------------------')

error_train = torch.zeros(batch_size)
error_val = torch.zeros(batch_size)

for epoch in range(num_epochs):

    running_loss_train = 0
    running_loss_val = 0
    running_error_train = 0
    running_error_val = 0
    num_batches_train = 0
    num_batches_val = 0
    count_train = 0
    count_val = 0
    j = 0
    k = 0

    model.train() # Set torch to train
    for X_train, _ in train_load:
        X_train = X_train.to(dev)
        # (X,y) is a mini-batch
        # X size NxIxHxW (N: batch_size, I: three ch )
        # y size N

        # Get metadata in batches from function
        price_data_train,metadata_train = get_batch_train(batch_size,j)
        price_data_train,metadata_train = price_data_train.to('cuda'),metadata_train.to('cuda')

        optimizer.zero_grad()
        N,C,nX,nY = X_train.size()
        print('IMAGES' + str(X_train.shape))
        print('META' + str(metadata_train.shape))
        y_pred_train = model(X_train.view(N,C,nX,nY),metadata_train)
        loss_train = criterion(y_pred_train.squeeze(), price_data_train)
        y_pred_total.append(y_pred_train)

        # Back propagation
        loss_train.backward()

        # Update the parameters
        optimizer.step()

        # Compute and update loss for entire training set
        running_loss_train += loss_train.cpu().detach().numpy()
        num_batches_train += 1

        # CALCULATING PERCENT ERROR
        for i in range(len(price_data_train)):
            error_train[i] = abs(price_data_train[i].item() - y_pred_train.squeeze()[i].item())/price_data_train[i].item()

        error_train_sum = sum(error_train)
        running_error_train = running_error_train + error_train_sum

        j = j+1


    k = 0
    model.eval()  # Set torch for evaluation
    for X_val, _ in val_load:
        X_val = X_val.to(dev)
        # (X,y) is a mini-batch
        # X size NxIxHxW (N: batch_size, I: three ch )
        # y size N

        # Get metadata in batches from function
        price_data_val,metadata_val = get_batch_val(batch_size,k)
        price_data_val,metadata_val = price_data_val.to('cuda'),metadata_val.to('cuda')

        # run model and compute loss
        N,C,nX,nY = X_val.size()
        y_pred_val = model(X_val.view(N,C,nX,nY), metadata_val)
        loss_val = criterion(y_pred_val.squeeze(),price_data_val)
        val_pred_total.append(y_pred_val)

        # Compute and update loss for entire val set
        running_loss_val += loss_val.cpu().detach().numpy()
        num_batches_val += 1

        for i in range(len(price_data_val)):
            error_val[i] = abs(price_data_val[i].item() - y_pred_val.squeeze()[i].item())/price_data_val[i].item()


        error_val_sum = sum(error_val)
        running_error_val = running_error_val + error_val_sum

        k = k+1  # Step batch counter

    ave_loss_train = running_loss_train/num_batches_train
    ave_loss_val = running_loss_val/num_batches_val
    ave_error_train = (running_error_train.item()/len(y_train))*100
    ave_error_val = (running_error_val.item()/len(y_val))*100


    # Store loss to tensor for plotting
    result_vals[epoch, 0] = ave_loss_train
    result_vals[epoch, 1] = ave_loss_val
    result_vals[epoch, 2] = ave_error_train
    result_vals[epoch, 3] = ave_error_val

    # Print loss every 9 epochs
    #if epoch % 2 == 1:
    print(epoch, '      ', round(ave_loss_train.item(),5),
          '     ', round(ave_loss_val.item(),5),
          '    ', round(ave_error_train,5),'   ',
          round(ave_error_val,5))
```

In [ ]:
```
# Some sample results/predictions
print(price_train[2]) #2 and 3
print(price_train[3]) #2 and 3
print(y_pred_total[0][2])
print(y_pred_total[0][3])
```

In [ ]:
```
# Plot Loss and Accuracy for train and val sets

xvals = torch.linspace(0, num_epochs, num_epochs+1)
plt.plot(xvals[0:num_epochs].cpu().numpy(),
         result_vals[:,0].cpu().detach().numpy())
plt.plot(xvals[0:num_epochs].cpu().numpy(),
         result_vals[:,1].cpu().detach().numpy())
plt.legend(['loss_train', 'loss_val'],
           loc='upper right')
#plt.xticks(xvals[0:num_epochs])
plt.title('Combined Model Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.tick_params(right=True, labelright=True)
# plt.savefig('loss.pdf', bbox_inches='tight', dpi=2400)
plt.show()
```

```
#
# For plotting percent error (which needs to be added above)
plt.plot(xvals[0:num_epochs].cpu().numpy(),
         result_vals[:,2].cpu().detach().numpy())
plt.plot(xvals[0:num_epochs].cpu().numpy(),
         result_vals[:,3].cpu().detach().numpy())
plt.legend(['error_train', 'error_val'],
           loc='upper right')
#plt.xticks(xvals[0:num_epochs])
plt.title('Combined Model Error')
plt.xlabel('epochs')
plt.ylabel('error')
plt.tick_params(right=True, labelright=True)
# plt.ylim(-0.15, 1.0)
plt.show()
```