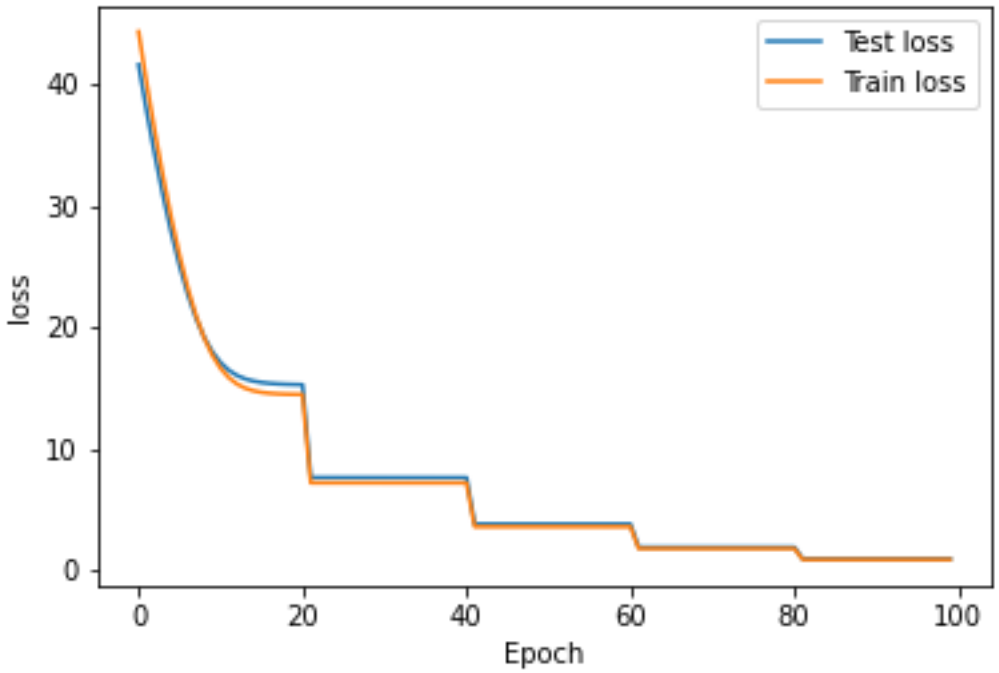
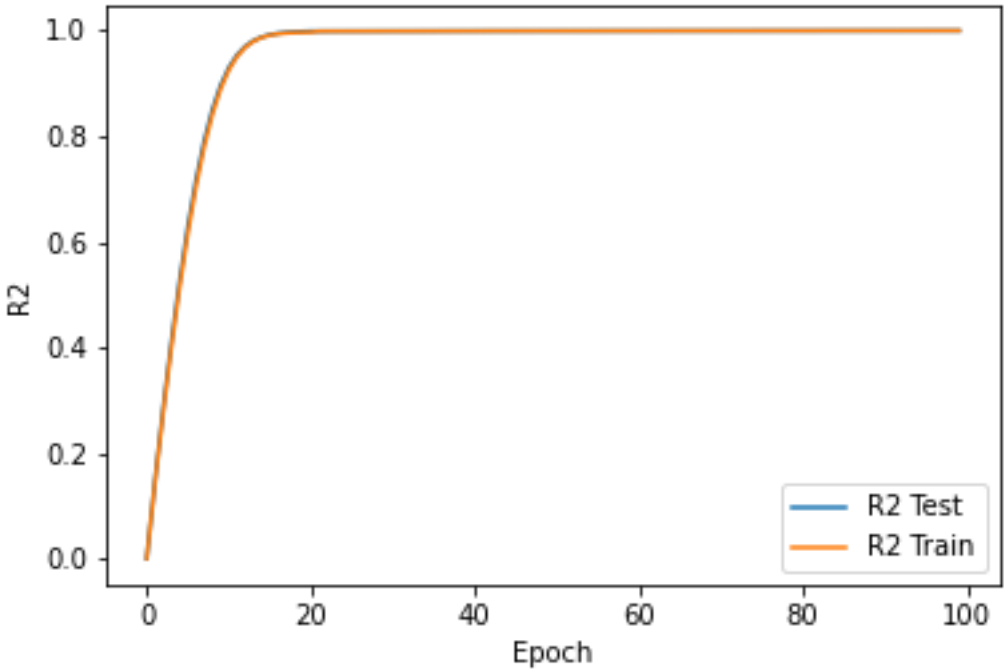
a) Minibatch starts at 64 and doubles every 20 epochs

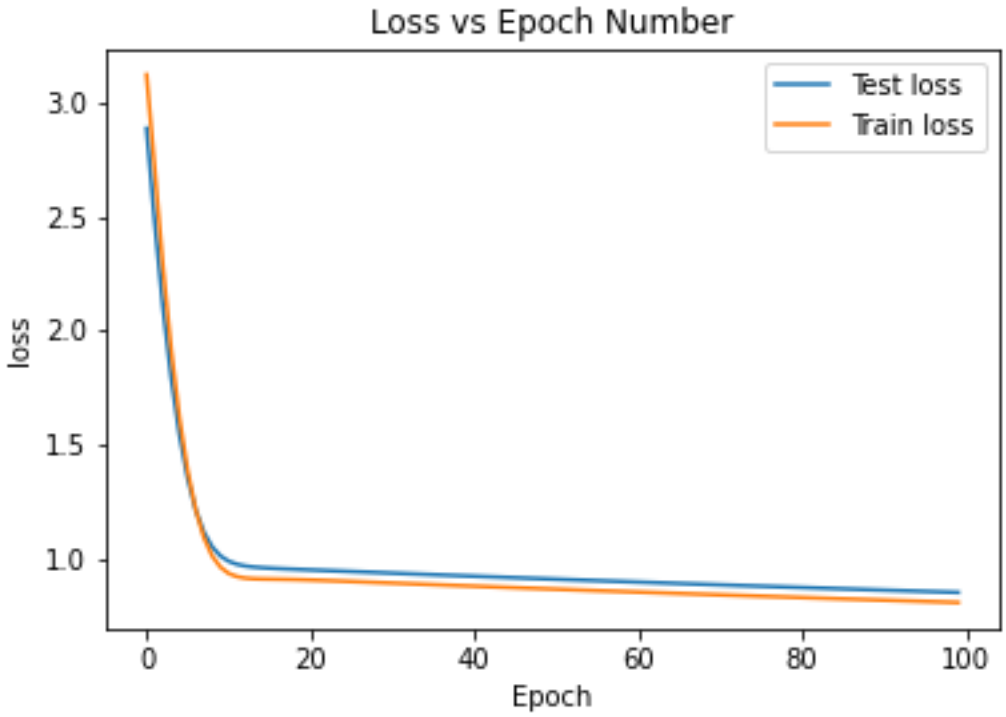
Loss function vs Epoch Number

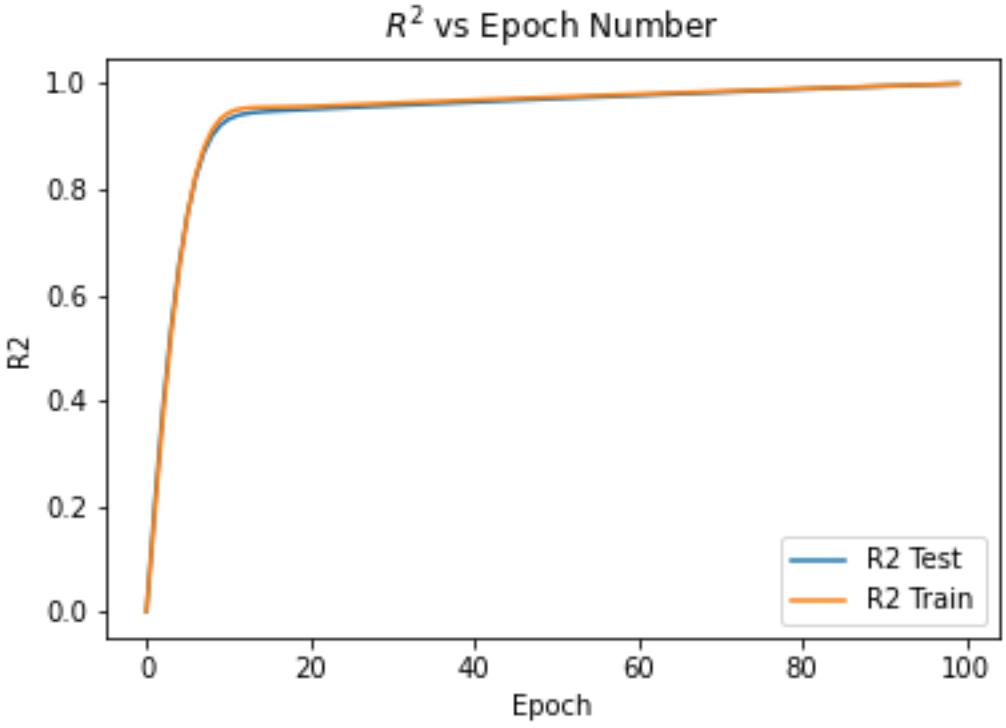


R^2 vs Epoch Number



b) Fixed minibatch = 1024

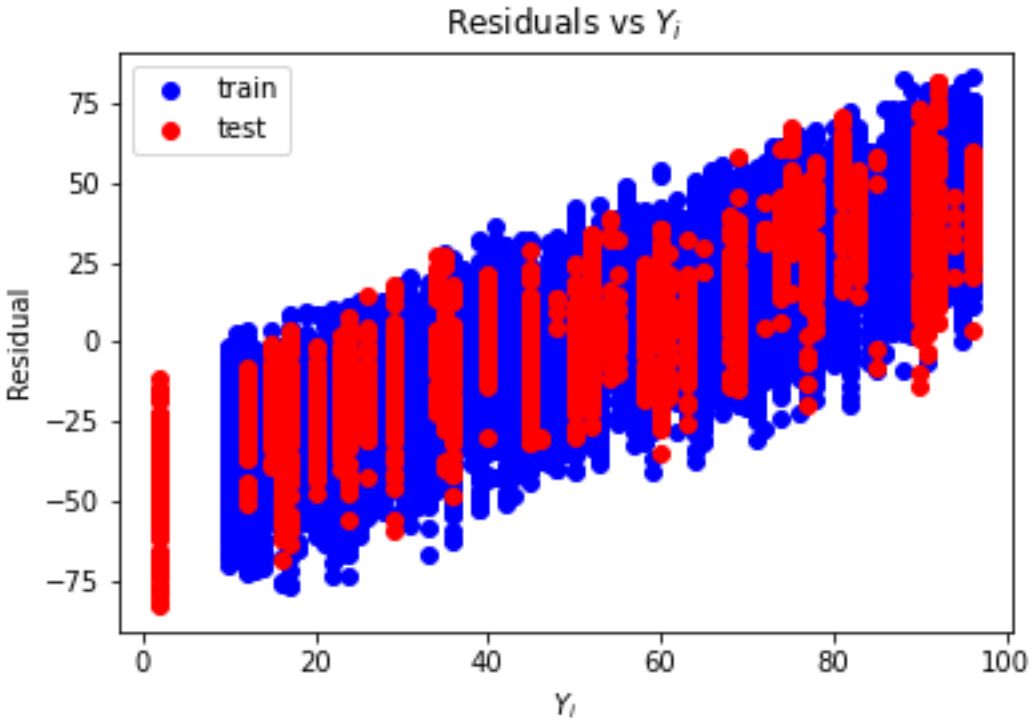




c) CNN Architecture

	Layer Description	Number of Filters	Kernel Size	Stride	Number of Parameters
Layer 1	Conv2d	16	5	1	1216
Layer 2	Conv2d + 2x2 max pooling	16	5	1	6416
Layer 3	Conv2d + 2x2 max pooling	32	5	1	12832
Layer 4	Conv2d + ReLU activation	32	5	1	25632
Layer 5	Linear	2048/1			2049
Total Number of Parameters	48145				

d) Plot of residuals



```
#!/usr/bin/env python
# coding: utf-8



import numpy as np
import matplotlib.pyplot as plt

import torch
from torch.autograd import Variable
import torch.optim as optim
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F

import os

#use gpu and cuda for speedup
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')



class MyDataset(torch.utils.data.Dataset):

    def __init__(self,folder,train):
        super(MyDataset, self).__init__()
        self.train = train
        self.folder = folder
        self.label_mean = 52.90
        if self.train:
            self.root = os.path.join(self.folder,"cnntrain")
        else:
            self.root = os.path.join(self.folder,"cnntest")
```

```
        self.files = os.listdir(self.root) # take all files in the root
directory




    def __len__(self):

        return len(self.files)


    def __getitem__(self, idx):

        #sample, label = torch.load(os.path.join(self.root, self.files[idx])) #
load the features of this sample

        sample = plt.imread(os.path.join(self.root, self.files[idx]))

        sample = np.moveaxis(sample,-1,0)

        #label  = np.array(float((self.files[idx][-20:-18]))-
self.label_mean).astype(float)

        label  = np.array(float((self.files[idx][-20:-18]))).astype(float)

        #print(label)

        sample,label =
torch.from_numpy(sample).type(torch.FloatTensor),torch.from_numpy(label).type(t
orch.FloatTensor)

        #label = label.view(-1,1)

        return sample, label




class CNN(nn.Module):

    def __init__(self, in_channels, out_channels):

        super(CNN, self).__init__()

        #Start the CNN architecture


        #First CONV layer (16-filters)

        self.conv1 = nn.Conv2d(in_channels = in_channels, out_channels = 16,
kernel_size = 5, stride = 1)

        #self.batch1 = nn.BatchNorm2d(16)
```

```
        #First CONV -> POOL layer (16-filters)
        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 16, kernel_size
= 5, stride = 1)
        self.pool = nn.MaxPool2d(kernel_size = 2, stride = 2)


        #Second CONV -> POOL layer (32-filters)
        self.conv3 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size
= 5, stride = 1)
        #self.batch2 = nn.BatchNorm2d(32)


        #First and only CONV -> ReLU layer (32-filters)
        self.conv4 = nn.Conv2d(in_channels = 32, out_channels = 32, kernel_size
= 5, stride = 1)
        #self.relu = nn.ReLU(inplace=True)


        #full connectivity layers
        self.fc1 = nn.Linear(in_features= 2048, out_features = out_channels)


    def forward(self, x):
        x = self.conv1(x)
        x = self.pool(self.conv2(x))
        x = self.pool(self.conv3(x))
        x = self.conv4(x)
        #x = self.conv5(x)
        #print(x.shape)
        x = torch.flatten(x, 1)
        #print(x.shape)
        out = self.fc1(x)
        #out = self.fc2(x)
        return out



def main(fixed):
```

```
    source_folder = "../datasets/cnn"
    dataset_train = MyDataset(folder=source_folder, train=True)
    dataset_val   = MyDataset(folder=source_folder, train=False)
    N_train = dataset_train.__len__()
    N_test = dataset_val.__len__()

    # defining the model
    input_dim = (3, 64, 64) #3 channels in 64 x 64 images
    model = CNN(3, 1)

    # Define loss
    loss_fn = nn.MSELoss(reduction='sum')

    # Optimizer
    if fixed == False:
        learning_rate = 1e-8 / N_train
    elif fixed == True:
        learning_rate = 1e-8 / N_train
    optimizer = optim.SGD(model.parameters(), lr = learning_rate, momentum=0.9,
weight_decay = 0.0001)



    # Move to GPU
    model = model.to(device)
    loss_fn = loss_fn.to(device)



    test_loss  = []
    train_loss = []

    R2_test  = []
```

```
    R2_train = []



    for epoch in range(100):
        print('EPOCH {}:'.format(epoch + 1))
        batch_size = 0


        #set up for part a and b of hw
        if fixed == False:
            if epoch <= 20:
                batch_size = 64
            elif epoch > 20 and epoch <= 40:
                batch_size = 128
            elif epoch > 40 and epoch <= 60:
                batch_size = 256
            elif epoch > 60 and epoch <= 80:
                batch_size = 512
            else:
                batch_size = 1024
            if epoch in list([0, 21, 41, 61, 81]):
                loader_train  = DataLoader(dataset=dataset_train, batch_size =
batch_size)
                loader_test   = DataLoader(dataset=dataset_val, batch_size =
batch_size)
                optimizer = optim.SGD(model.parameters(), lr = learning_rate /
(epoch + 1), momentum=0.9, weight_decay = 0.0001)


        elif fixed == True:
            batch_size = 1024
            loader_train = DataLoader(dataset = dataset_train, batch_size =
batch_size)
            loader_test = DataLoader(dataset = dataset_val, batch_size =
batch_size)
```

```
running_train_loss = 0
running_test_loss  = 0
predictions_train, actuals_train = [], []

#make sure gradient tracking is on
model.train()

for data in loader_train:

    inputs, labels = data
    #inputs = (inputs - torch.mean(inputs)) / torch.std(inputs)
    #print(inputs.shape,labels.shape)

    # Zero gradients for every batch
    optimizer.zero_grad()

    #send to gpu
    inputs = inputs.to(device)
    labels = labels.to(device)

    #Make prediction for this batch
    outputs = model(inputs)

    labels = labels.unsqueeze(1)

    #Compute loss and its gradient
    loss = loss_fn(outputs, labels)
    loss.backward()
```

```
            #adjust learning weight
            optimizer.step()

            #gather loss
            running_train_loss += loss.item() / batch_size

            #need to get values outside of loop
            model.eval()
            with torch.no_grad():
                y_predicted_train_batch = model(inputs)
                y_predicted_train_batch = y_predicted_train_batch.cpu().numpy()
                actual = labels.cpu().numpy() #Copy the tensor to cpu
                actual = actual.reshape((len(actual), 1))

                # store y values from model
                for element in y_predicted_train_batch:
                    predictions_train.append(element)
                for element in actual:
                    actuals_train.append(element)


#print(np.array(predictions_train).shape,np.array(actuals_train).shape)
            model.train()



        train_loss.append(running_train_loss / N_train)


        actuals_train, predictions_train = np.array(actuals_train).flatten(),
np.array(predictions_train).flatten()
        #actuals_train, predictions_train = torch.flatten(actuals_train),
torch.flatten(predictions_train)


        #print(actuals_train.shape,predictions_train.shape)
```

```
        y_bar_train = np.mean(actuals_train)


        #assert actuals_train.shape ==  predictions_train.shape


        R2_train.append(1 - (np.sum(np.square(actuals_train -
predictions_train)) / np.sum(np.square(actuals_train -  y_bar_train))))


        predictions_test, actuals_test = [], []


        #Do not need gradients to do validation
        model.eval()
        with torch.no_grad():
            for vdata in loader_test:


                vinputs, vlabels = vdata
                #vinputs = (vinputs - torch.mean(vinputs)) / torch.std(vinputs)
                y_predicted_on_test_batch = model(vinputs.to(device))
                vlabels = vlabels.to(device)
                vlabels = vlabels.unsqueeze(1)


                #print(labels.size(),y_predicted_on_test_batch.size())


                batch_loss = loss_fn(y_predicted_on_test_batch, vlabels)
                running_test_loss += batch_loss.item() / batch_size


                y_predicted_on_test_batch =
y_predicted_on_test_batch.cpu().numpy()
                actual = vlabels.cpu().numpy() #Copy the tensor to cpu
                actual = actual.reshape((len(actual), 1))


                # store test values
                for element in y_predicted_on_test_batch:
```

```
                    predictions_test.append(element)
                for element in actual:
                    actuals_test.append(element)



        test_loss.append(running_test_loss / N_test)


        actuals_test, predictions_test = np.array(actuals_test).flatten(),
np.array(predictions_test).flatten()
        #print(actuals_test.shape,predictions_test.shape)
        #assert actuals_test.shape ==  predictions_test.shape


        y_bar_test  = np.mean(actuals_test)
        R2_test.append(1 - (np.sum(np.square(actuals_test -  predictions_test))
/ np.sum(np.square(actuals_test -  y_bar_test))))


        model.train()
        print(R2_test[-1], R2_train[-1], test_loss[-1], train_loss[-1])


    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad: continue
        params = parameter.numel()
        print(name, "\tnum params:", params)
        total_params+=params
    print(f"Total Trainable Params: {total_params}")


    torch.save(model.state_dict(),
os.path.join(source_folder,"model","CNN_A.pt"))
    return R2_test, R2_train, test_loss, train_loss
```

```
R2_test, R2_train, test_loss, train_loss = main(fixed = False)



plt.plot(R2_test,label ="R2 Test")
plt.plot(R2_train,label="R2 Train")
plt.xlabel('Epoch')
plt.ylabel('R2')
plt.legend()
plt.savefig("R2_varied")
plt.show()



plt.plot(test_loss,label ="Test loss")
plt.plot(train_loss,label="Train loss")
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.legend()
plt.savefig("Loss_varied")
plt.show()


R2_test_b, R2_train_b, test_loss_b, train_loss_b = main(fixed = True)



plt.plot(R2_test_b,label ="R2 Test")
plt.plot(R2_train_b,label="R2 Train")
plt.xlabel('Epoch')
plt.ylabel('R2')
```

```
plt.title("$R^2$ vs Epoch Number")
plt.legend()
plt.savefig("R2_fixedbatch")
plt.show()



plt.plot(test_loss_b,label ="Test loss")
plt.plot(train_loss_b,label="Train loss")
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.title("Loss vs Epoch Number")
plt.legend()
plt.savefig("Loss_fixedbatch")
plt.show()



import pandas as pd
num_param = 48145
results = {}
results['Layer 1'] = ['Conv2d', 16, 5, 1, 1216]
results['Layer 2'] = ['Conv2d 2x2 max pooling', 16, 5, 1, 6416]
results['Layer 3'] = ['Conv2d + 2x2 max pooling', 32, 5, 1, 12832]
results['Layer 4'] = ['Conv2d + ReLU activation', 32, 5, 1, 25632]
results['Layer 5'] = ['Linear', "2048/1",'','', 2049]
#results['Layer 6'] = ['Linear', "84/1",'','', 0]
results['Total Number of Parameters'] = [num_param, '','','', '']
parameter_list = pd.DataFrame(results, index = ['Layer Description' , 'Number
of Filters', 'Kernel_Size','Stride', 'Number of Parameters']).transpose()
parameter_list
```

```
def predict():

    model2 = CNN(3,1)
    model2.load_state_dict(torch.load('../datasets/cnn/model/CNN_A.pt'))
    model2.eval()

    source_folder = "../datasets/cnn"
    dataset_train = MyDataset(folder=source_folder,train=True)
    dataset_val   = MyDataset(folder=source_folder,train=False)
    loader_train  = DataLoader(dataset=dataset_train, batch_size = 1024)
    loader_test   = DataLoader(dataset=dataset_val,batch_size = 1024)



    predictions_train, actuals_train = [], []
    with torch.no_grad():
        for data in loader_train:
            inputs, labels = data
            y_predicted_train_batch = model2(inputs).cpu().numpy()
            actual = labels.cpu().numpy()
            actual = actual.reshape((len(actual), 1))
            # store

            for element in y_predicted_train_batch:
                predictions_train.append(element)
            for element in actual:
                actuals_train.append(element)



    predictions_test, actuals_test = [], []
```

```
    with torch.no_grad():
        for data in loader_test:
            inputs, labels = data
            y_predicted_on_current_batch = model2(inputs).cpu().numpy()
            actual = labels.cpu().numpy()
            actual = actual.reshape((len(actual), 1))
            # store
            for element in y_predicted_on_current_batch:
                predictions_test.append(element)
            for element in actual:
                actuals_test.append(element)


    return predictions_train, actuals_train, predictions_test, actuals_test


y_pred_train, y_train, y_pred_test, y_test = predict()



residuals_train = np.array(y_train) - np.array(y_pred_train)
residuals_test = np.array(y_test) - np.array(y_pred_test)
plt.scatter(y_train, residuals_train, color="blue", label="train")
plt.scatter(y_test, residuals_test, color="red", label="test")
plt.xlabel("$Y_i$")
plt.ylabel("Residual")
plt.title("Residuals vs $Y_i$")
plt.legend()
plt.savefig("Residuals.png")
plt.show()
```