```
In [2]: from solution import Trainer, NetworkConfiguration
```

```
In [1]: import torch
```

```
In [3]: activation = Trainer.create_activation_function('relu')
```

Question 2

```
In [6]: from typing import Tuple, List, NamedTuple
```

```
In [7]: class NewNetworkConfiguration(NamedTuple):
            n_channels: Tuple[int, ...] = (16,32,45)
            kernel_sizes: Tuple[int, ...] = (3, 3, 3)
            strides: Tuple[int, ...] = (1, 1, 1)
            dense_hiddens: Tuple[int, ...] = (128, 128)
```

```
In [8]: lrs = [0.01, 0.0001, 0.00000001]
        test_maes= []
```

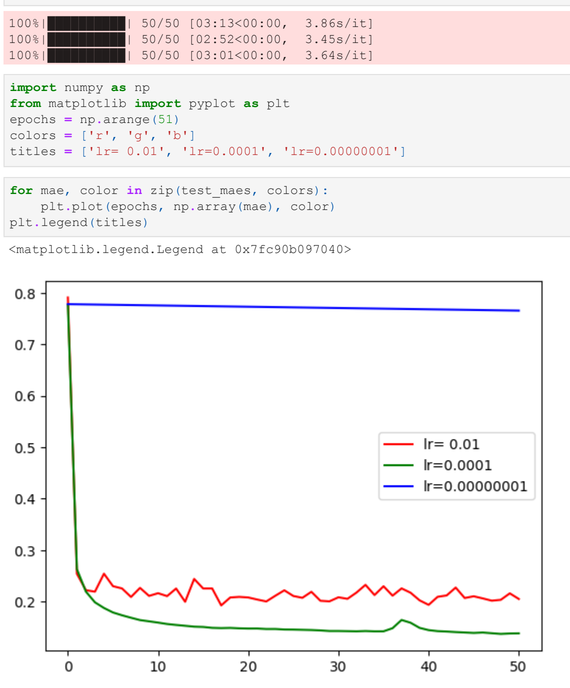
```
In [9]: for lr in lrs:
            mlp_trainer = Trainer(network_type='mlp', net_config=NewNetworkConfiguration(), batch_size=128, lr = lr)
            mlp_trainer.train_loop(50)
            test_maes.append(mlp_trainer.train_logs['test_mae'])
```

```
100%|██████████| 50/50 [03:13<00:00,  3.86s/it]
100%|██████████| 50/50 [02:52<00:00,  3.45s/it]
100%|██████████| 50/50 [03:01<00:00,  3.64s/it]
```

```
In [10]: import numpy as np
         from matplotlib import pyplot as plt
         epochs = np.arange(51)
         colors = ['r', 'g', 'b']
         titles = ['lr= 0.01', 'lr=0.0001', 'lr=0.00000001']
```

```
In [11]: for mae, color in zip(test_maes, colors):
             plt.plot(epochs, np.array(mae), color)
         plt.legend(titles)
```

```
Out[11]: <matplotlib.legend.Legend at 0x7fc90b097040>
```



Discussion

As a expected behvior, all the models with different learning rates start at the same place, learning rate 10^-2 and 10^-4 seem to drop significantly after a number of epochs but as learning rate increses(lr=10^-8) the model hardly trains: it is very slow. with small learning rate(lr=10^-2) it keeps jumping up and down between the optimum and takes longer time to converge. The optimal learning rate between these choices is 10^-4

```
In [12]: default_cnn_trainer = Trainer(batch_size=128, net_config=NewNetworkConfiguration())
```

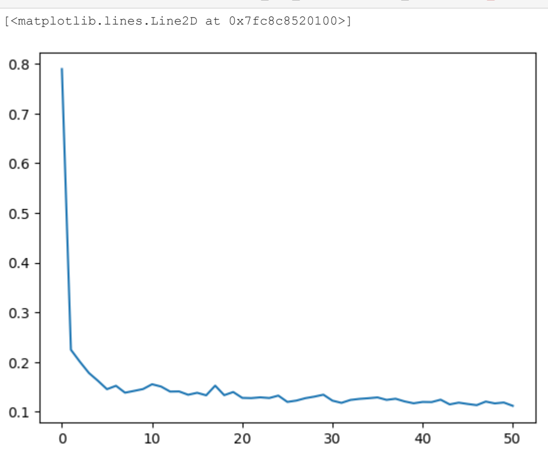
```
In [13]: default_cnn_trainer.train_loop(50)
```

```
100%|██████████| 50/50 [03:16<00:00,  3.92s/it]
```

```
Out[13]: {'train_loss': [1.0344717502593994,
  0.20082491636276245,
  0.1475037783384323,
  0.11355289816856384,
  0.09643035382032394,
  0.08125809580087662,
  0.0787515863776207,
  0.06960788369178772,
  0.0636700913310051,
  0.06224944442510605,
  0.06596673280000687,
  0.0606544129550457,
  0.04626021906733513,
  0.05104834586381912,
  0.042149618268013,
  0.05425605550408363,
  0.04190702736377716,
  0.052702706307172775,
  0.038915131241083145,
  0.04202105849981308,
  0.04329512268304825,
  0.040454085916280746,
  0.035262275487184525,
  0.041434261947870255,
  0.04398469626903534,
  0.030826522037386894,
  0.03273092955350876,
  0.0326799601316452,
  0.03261213004589081,
  0.03740614280104637,
  0.03302663564682007,
  0.02715657837688923,
  0.025991329923272133,
  0.03112666681408882,
  0.0310402270406846,
  0.027203848585486412,
  0.027775252237915993,
  0.02468753792345524,
  0.022772403433918953,
  0.021379858255386353,
  0.023108825087547302,
  0.020658385008573532,
  0.017890002578496933,
  0.017443383112549782,
  0.01905929669737816,
  0.016744181513786316,
  0.014183622784912586,
  0.019142545759677887,
  0.013565663248300552,
  0.013493961654603481,
  0.014370174147188663],
 'test_loss': [1.0114020109176636,
  0.18714900314807892,
  0.15247434377670288,
  0.12394973635673523,
  0.10677468776702881,
  0.09515918046236038,
  0.09681716561317444,
  0.08572312444448471,
  0.08623319864273071,
  0.09948132187128067,
  0.11413929611444473,
  0.11907105147838593,
  0.10136505216360092,
  0.10580488294363022,
  0.10278830677270889,
  0.10982845723628998,
  0.10323623567819595,
  0.1342983990907669,
  0.09738790988922119,
  0.1006079912185669,
  0.09758009016513824,
  0.09760189056396484,
  0.09666677564382553,
  0.09162579476833344,
  0.08721278607845306,
  0.07894019782543182,
  0.08317037671804428,
  0.09574374556541443,
  0.09486612677574158,
  0.11023630946874619,
  0.08982658386230469,
  0.08709407597780228,
  0.08767600357532501,
  0.09047281742095947,
  0.0966302827000618,
  0.08826736360788345,
  0.08780379593372345,
  0.10446401685476303,
  0.08671734482049942,
  0.08365285396575928,
  0.09070120006799698,
  0.08264794945716858,
  0.09915172308683395,
  0.0808945745229721,
  0.08936496078968048,
  0.09180083125829697,
  0.08841796964406967,
  0.08857586979866028,
  0.09348580986261368,
  0.09410692006349564,
  0.07688230276107788],
 'train_mae': [0.8021745681762695,
  0.23063777387142181,
  0.1963544636964798,
  0.1679833084344864,
  0.15024833381175995,
  0.1350274682044983,
  0.14206938445568085,
  0.12679605185985565,
  0.12739266455173492,
  0.13014188408851624,
  0.1376175731420517,
  0.13186237215995789,
  0.1207040399312973,
  0.11906635016202927,
  0.10975729674100876,
  0.11506084352731705,
  0.1074741408228742,
  0.12556034326553345,
  0.1084187924861908,
  0.11502372473478317,
  0.1041199266910553,
  0.10257494449615479,
  0.10114727914333344,
  0.10316569358110428,
  0.11259963363409042,
  0.09424735605716705,
  0.09887495636940002,
  0.10046600550413132,
  0.10349836945533752,
  0.10847415030002594,
  0.09547021239995956,
  0.0892011895775795,
  0.094544418156147,
  0.09812401980161667,
  0.09870248287916183,
  0.09922914206981659,
  0.09655298292636871,
  0.09320543706417084,
  0.08994447439908981,
  0.0878700539469719,
  0.08899024873971939,
  0.08633523434400558,
  0.08491679280996323,
  0.0831553041934967,
  0.08390764892101288,
  0.08440978080034256,
  0.0788940042257309,
  0.09058646112680435,
  0.0797191634774208,
  0.08035503327846527,
  0.07909205555915833],
 'test_mae': [0.7891135215759277,
  0.22473213076591492,
  0.20050089061260223,
  0.177947536110878,
  0.1623048186302185,
  0.14544036984443665,
  0.15229186415672302,
  0.1384153515100479,
  0.14199614524841309,
  0.14572939276695251,
  0.15538723766803741,
  0.15074869990348816,
  0.140671044588089,
  0.14093539118766785,
  0.13423654437065125,
  0.13821734488010406,
  0.1331654191017151,
  0.15255305171012878,
  0.13342028856277466,
  0.1397707164287567,
  0.12797656655311584,
  0.12752042710781097,
  0.12905250489711761,
  0.12770208716392517,
  0.13256056606769562,
  0.11986173689365387,
  0.1225336492061615,
  0.1273847073316574,
  0.13050898909568787,
  0.1344081610441208,
  0.12240338325500488,
  0.1179497018456459,
  0.12384666502475739,
  0.125972926616687,
  0.12729671597480774,
  0.12890750169754028,
  0.1239732876420021,
  0.12626290321350098,
  0.12088953703641891,
  0.11708945780992508,
  0.11981463432312012,
  0.11953610181808472,
  0.12434300035238266,
  0.11488685756921768,
  0.11842333525419235,
  0.11571648716926575,
  0.11337512731552124,
  0.12058897316455841,
  0.11690179258584976,
  0.11867371946573257,
  0.11197257786989212]}
```

```
In [14]: plt.plot(epochs, np.array(default_cnn_trainer.train_logs['test_mae']))
```

```
Out[14]: [<matplotlib.lines.Line2D at 0x7fc8c8520100>]
```

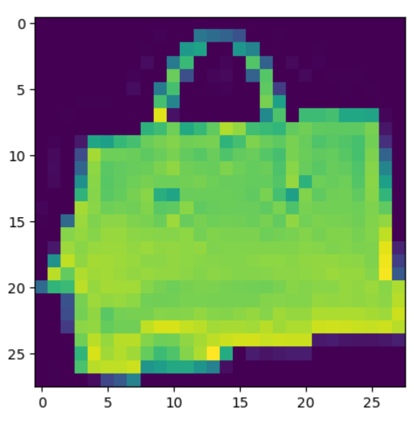


Question 3

part a) figure of original image

```
In [7]: test_img = (Trainer().train[0][0]+1)/2
        plt.imshow(test_img.permute(1, 2, 0))
        plt.title('Original Image')
```

```
Out[7]: <matplotlib.image.AxesImage at 0x7f9eb1ca3100>
```

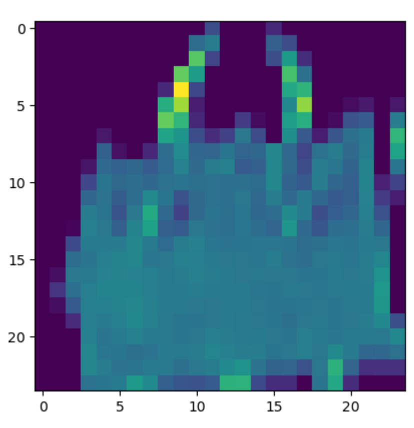


```
In [34]: conv = torch.nn.Conv2d(kernel_size=3, in_channels=1, out_channels=1, stride=1, padding=0)
         fullconv_model = lambda x: torch.relu(conv((torch.relu(conv((x))))))
         model = fullconv_model
```

part b) image of model prediction

```
In [38]: pred = model(test_img)
         plt.imshow(pred.detach().permute(1, 2, 0))
         plt.title('model prediction image')
```

```
Out[38]: <matplotlib.image.AxesImage at 0x7f9ec2252f20>
```



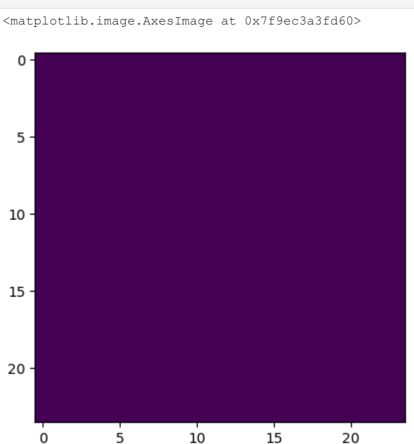
```
In [12]: from torchvision.transforms.functional import to_tensor, normalize, affine

         from functools import partial
         shift_amount = 5
         shift = partial(affine, angle=0, translate=(shift_amount, shift_amount), scale=1, shear=0)
         rotation = partial(affine, angle=90, translate=(0, 0), scale=1, shear=0)
```

part c) absolute difference of shifted prediction and prediction of shifted image

```
In [43]: pred_shift_img = model(shift(test_img))
         shift_pred_img = shift(model(test_img))
         difference = abs(pred_shift_img - shift_pred_img)
         plt.imshow(difference.detach().permute(1,2,0))
```

```
Out[43]: <matplotlib.image.AxesImage at 0x7f9ec3a3fd60>
```

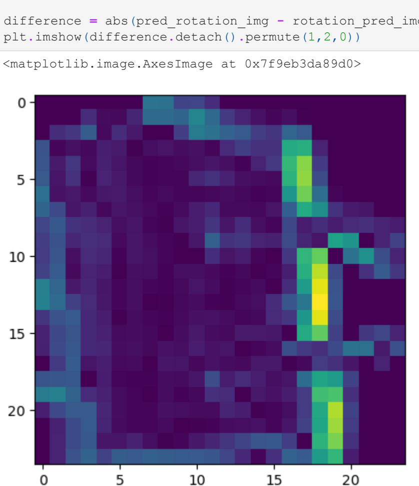


part d) absolute difference of rotated prediction and prediction of rotated image

```
In [44]: pred_rotation_img = model(rotation(test_img))
         rotation_pred_img = rotation(model(test_img))

         difference = abs(pred_rotation_img - rotation_pred_img)
         plt.imshow(difference.detach().permute(1,2,0))
```

```
Out[44]: <matplotlib.image.AxesImage at 0x7f9eb3da89d0>
```



discusstion

1)

As we can see, shifting is equivariant(the plot is completely purple with no pixel of difference) but is not equivariant to rotation(we can see pixels of difference in the plot)

2)

Some of them like translation is much more important than rotation. Translation give us the variance of data and frequency of occurrence but Rotating an image may not give us as much information as translation.

```
In [ ]:
```