I trained 1,000 MNIST samples using 9 different augmentation methods (torchvision.transforms). The methods used and the sample images produced by the transformations are shown below.

```
loss_func = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

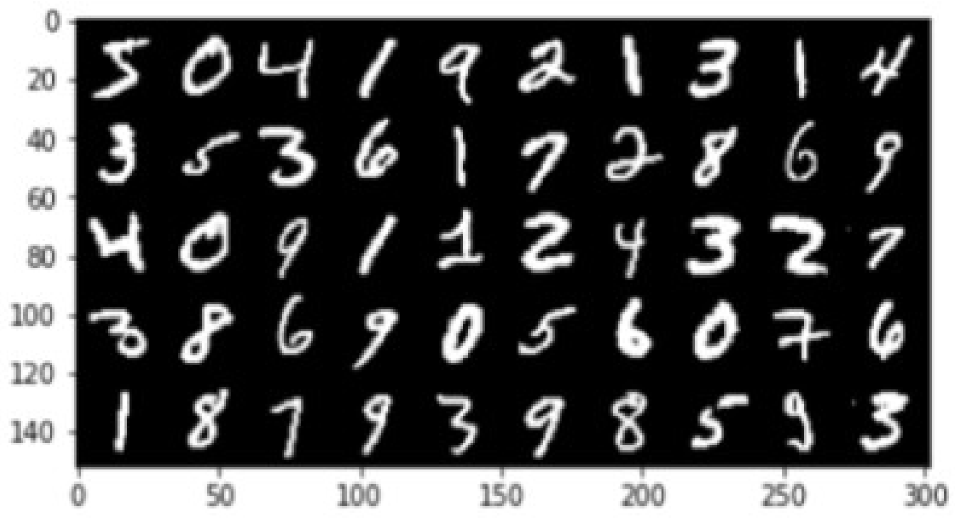
LeNet (with BatchNorm2d)

```
(conv1): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
(bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv2): Conv2d(20, 50, kernel_size=(5, 5), stride=(1, 1))
(bn2): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(fc1): Linear(in_features=800, out_features=500, bias=True)
(fc2): Linear(in_features=500, out_features=10, bias=True)
(relu): ReLU()
```

1. **Basic**

Basic transformation is applied. Images are converted into float tensors, and the inputs' range is changed to [0, 1]. Then it is standardized by mean and std as specified.

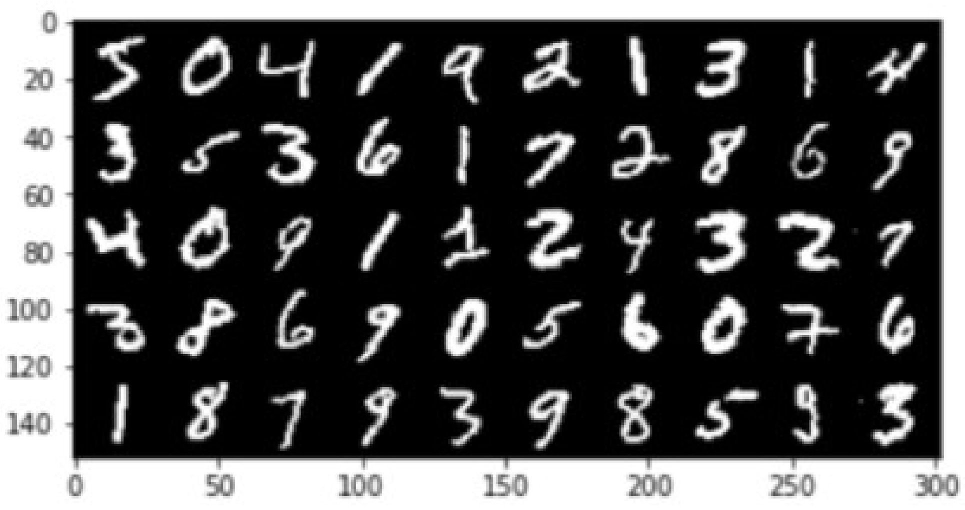
```
transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



2. **RandomAffine**

Random affine transform is applied where the range of degrees is set as specified.

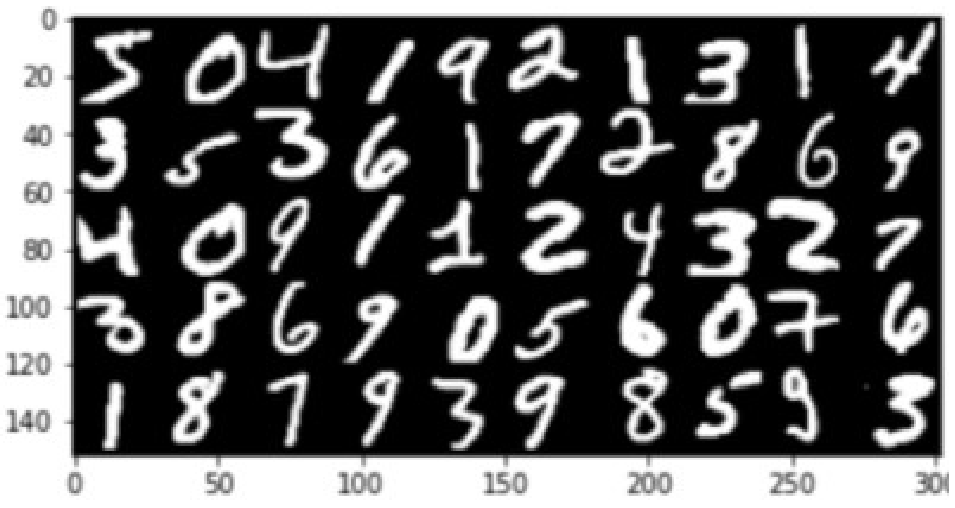
```
transforms.Compose([transforms.RandomAffine((-10.0, 10.0)),
transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



3. **RandomCrop**

Images are cropped to a specified size at random locations from magnified images.

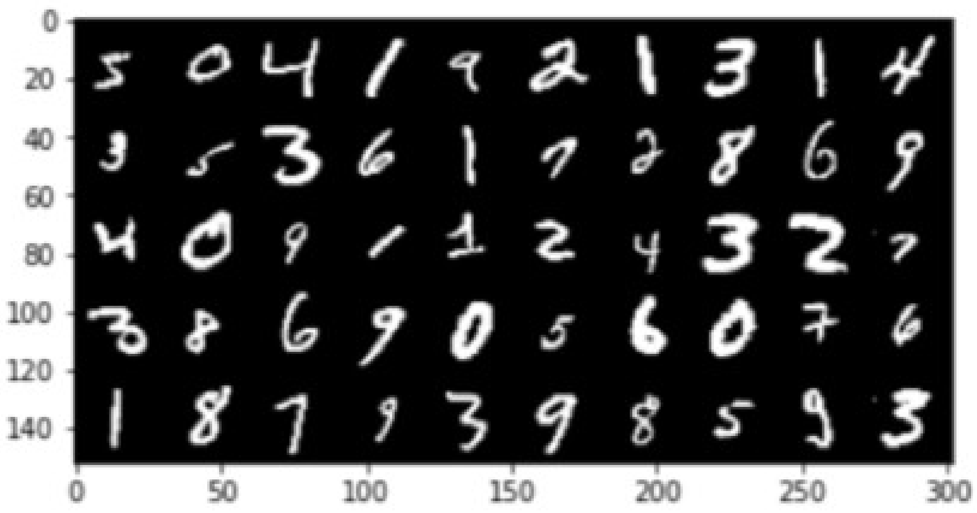
```
transforms.Compose([transforms.Resize((35,35)), transforms.RandomCrop(2
8), transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



4. **RandomPerspective**

Random perspective transform is applied where the distortion scale and interpolation are set as specified.

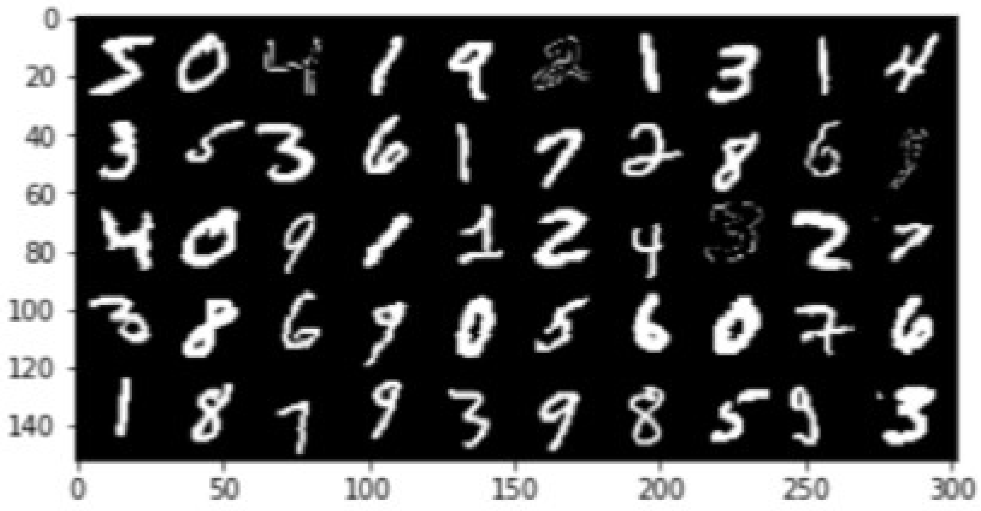
```
transforms.Compose([transforms.RandomPerspective(distortion_scale=0.5,
p=0.5, interpolation=3), transforms.ToTensor(), transforms.Normalize
((0.1307,), (0.3081,))])
```



5. **RandAugment**

The picture data is automatically augmented (twice).

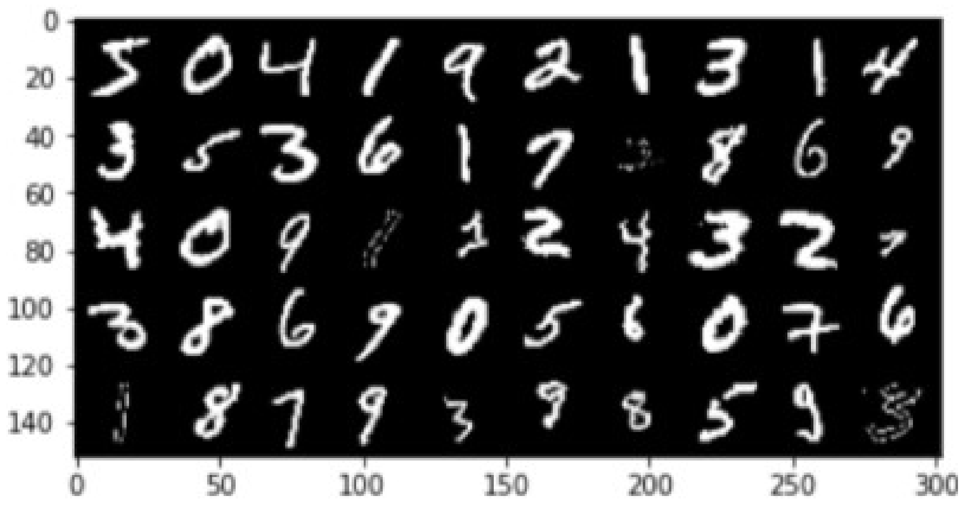
```
transforms.Compose([transforms.RandAugment(num_ops=2), transforms.ToTen
sor(), transforms.Normalize((0.1307,), (0.3081,))])
```



6. **RandomApply**

A list of transformations (RandomAffine, RandomPerspective, RandAugment) is rand
omly applied.

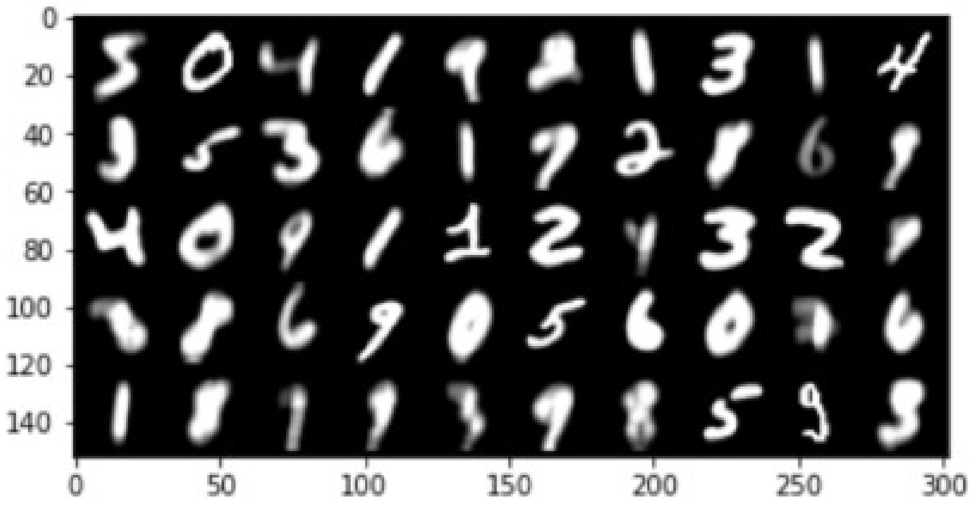
```
transforms.Compose([transforms.RandomApply(transforms=[transforms.Rando
mAffine((-10.0, 10.0)),
transforms.RandomPerspective(distortion_scale=0.5, p=0.5, interpolation
=3), transforms.RandAugment(num_ops=2)], p=0.5),
transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



7. **GaussianBlur**

Gaussian Blur transformation is applied where kernel size and sigma are set as spec ified.

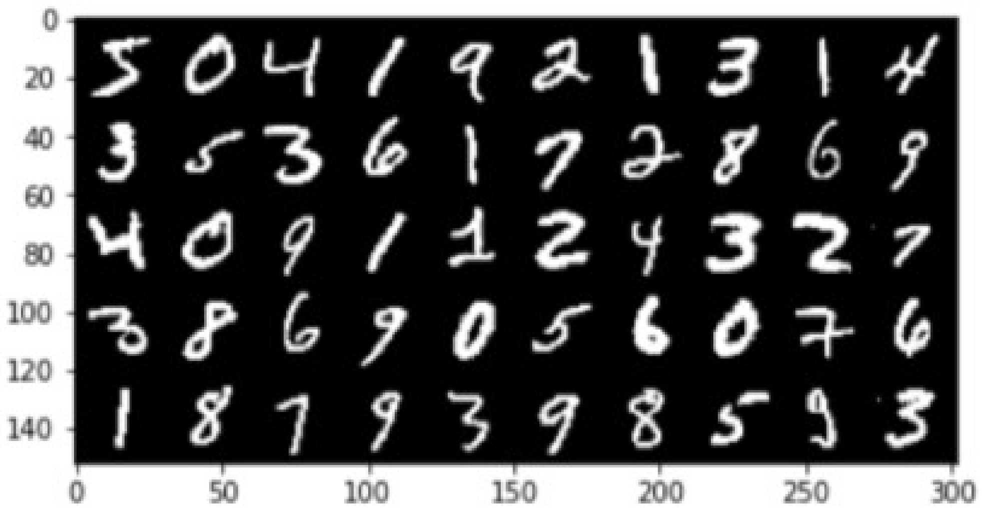
```
transforms.Compose([transforms.GaussianBlur(kernel_size=(5, 9),
sigma=(0.1, 5)), transforms.ToTensor(), transforms.Normalize((0.1307,),
(0.3081,))])
```



8. **RandomAdjustSharpness**

The sharpness of the images are randomly adjusted by specified sharpness factor.

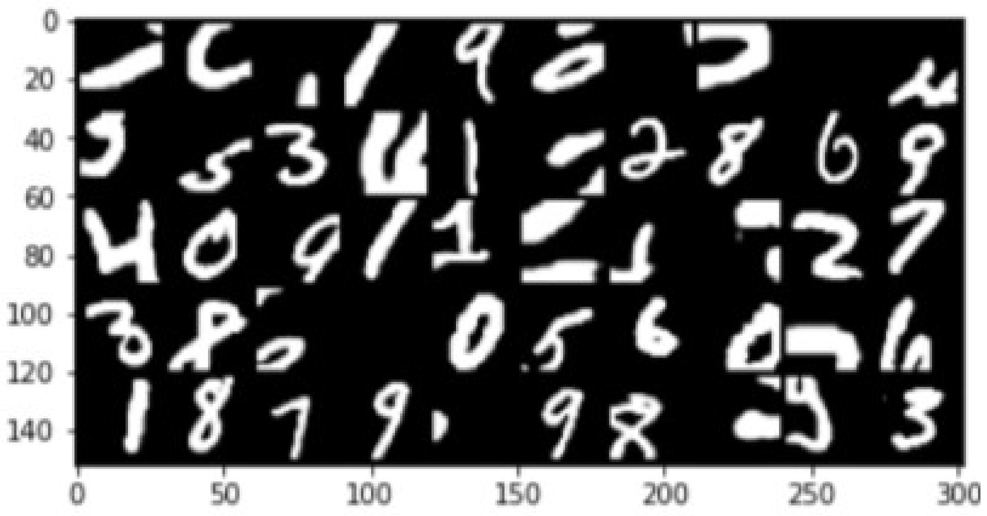
```
transforms.Compose([transforms.RandomAdjustSharpness(sharpness_factor=
2),
transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



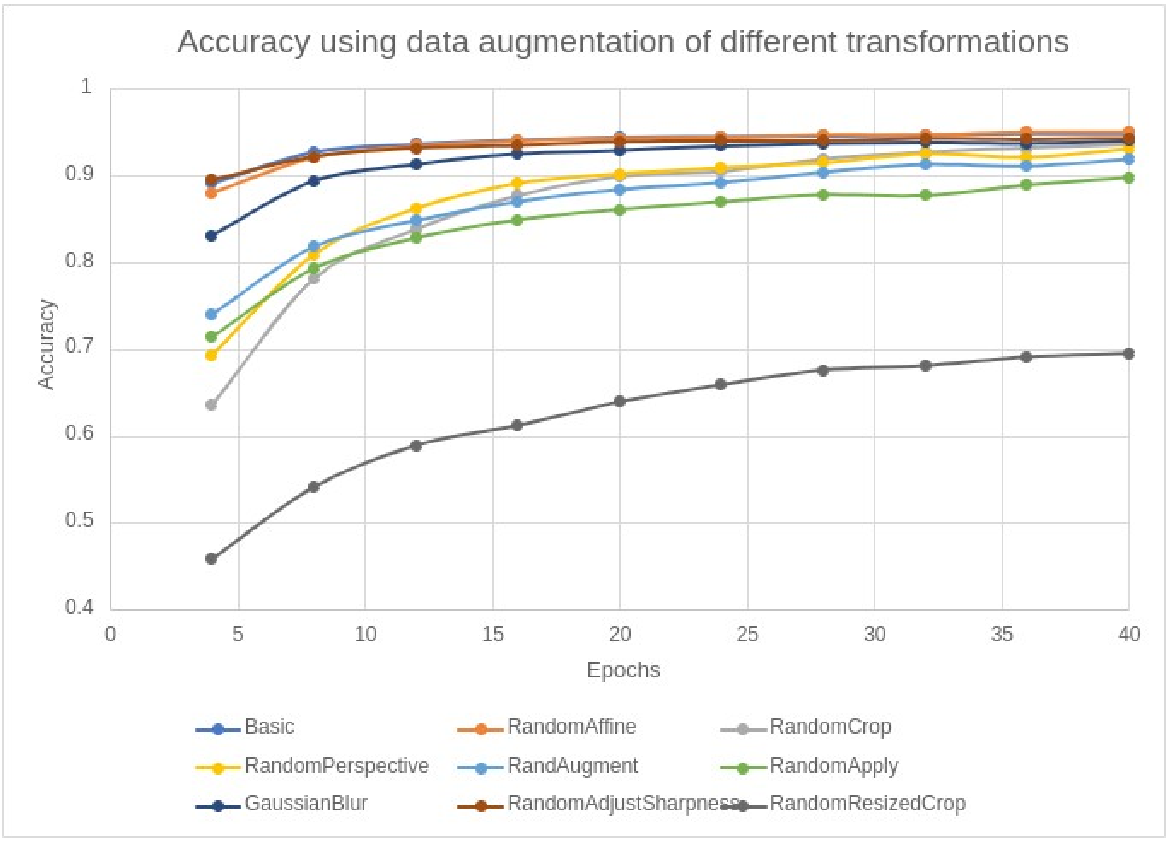
9. **RandomResizedCrop**

The images are cropped at a random location, then are resized to specified size.

```
transforms.Compose([transforms.RandomResizedCrop(size=(28,28)),
transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
```



The models are tested for every 4 epochs until 40 epochs. The prediction accuracy f or each method is shown in the graph below.



RandomResizedCrop clearly has the worst accuracy. This is because the images used lose t he important information from the real images (as can be seen from the sample images). Ev en humans may face difficulty to guess numbers from these images. Thus, RandomResized Crop may not be the best transformation for number image data.

The other 8 transformations perform well. Basic, RandomAffine, and RandomAdjustSharpne ss have the best accuracy from all the transformations from the beginning until the end. This is probably because all these three have similar images which contain complete data.

GaussianBlur initially does not have accuracy as good as these three, but after 40 epochs, it has similar accuracy as these three. This is probably because blurred images may be difficul t to recognize at the beginning. But since they do not lose significant information, the final ac curacy is still high.

RandomCrop initially does not have good accuracy, but it increases significantly compared t o other transformations and finally has similar accuracy as these three. RandomPerspective also has similar development as RandomCrop, but it has slightly better accuracy than Rando mCrop at the beginning. The argument for the reasoning of the result for these two is similar as for GaussianBlur.

RandAugment and RandomApply have a fair initial accuracy, but the final accuracy is not as good as the others.This is probably because some of the information is lost due to RandAug ment (note that RandomApply also contains RandAugment). As can be seen from the image s, some are not as clear as the real images.