In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import torch
import torch.nn as nn
import torchvision

import matplotlib.pyplot as plt
import numpy as np

from typing import Union, Literal
```

### (a) Implement the 90-degree rotation group (cyclic group) over a plane including:

```
(i) group element
(ii) group product (__mul__)
(iii) group inverse (__invert__)
(iv) group action (__call__)
```

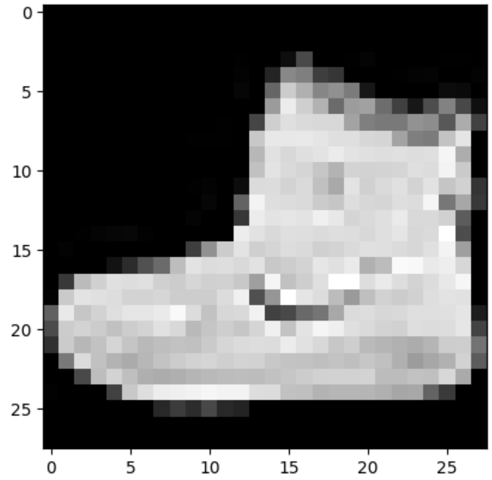
In [3]:
```
class RotationGroupElement():
    """
    Implementation of the cyclic rotation group via implemention of a group element.
    """
    def __init__(self, idx: int) -> None:
        """
        Parameters:
            idx: Index of group element.
                 Corresponds with number of 90 degree rotations performed.
        """
        assert(idx in [0,1,2,3])
        self.idx = idx

    def __call__(self, img: torch.tensor) -> torch.tensor:
        """
        Group action. Rotates img by (idx * 90) degrees.

        Note: rotates last two dimensions by default so that we can apply the
              group action to any kernel/signal of shape (..., H,W) or (..., W,H)
        """
        return torch.rot90(img, k=self.idx, dims=[-1,-2])

    def __mul__(self, other):
        """
        Group product: (g * g')(x) = g(g'(x))
        """
        new_idx = (self.idx + other.idx) % 4
        return RotationGroupElement(new_idx)

    def __invert__(self):
        """
        Group inverse: (~g * g) = identity
        """
        new_idx = (4 - self.idx) % 4
        return RotationGroupElement(new_idx)
```

### Apply the group actions on an image and print (imshow) them.

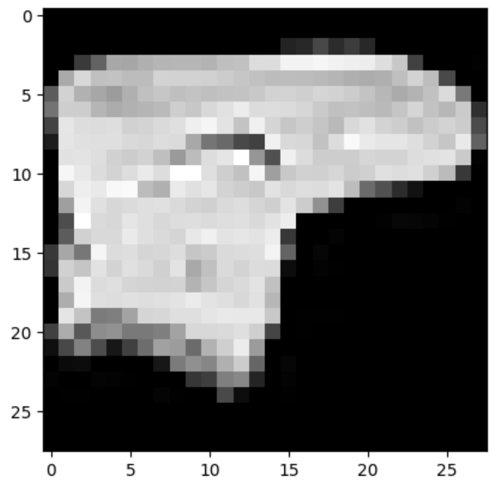
In [4]:
```
def plot_img(img_tensor):
    plt.imshow(img_tensor.squeeze(), cmap='gray')
    plt.show()
```

In [5]:
```
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transforms.Compose([transforms.ToTensor()]) )

num_workers=16
batch_size=128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
```

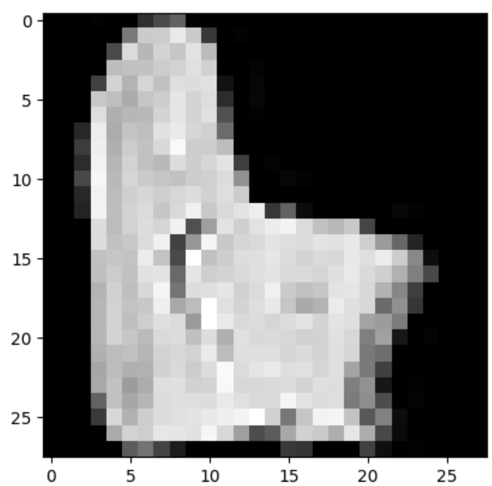
In [6]:
```
img = train_dataset[0][0]
plot_img(img)
```



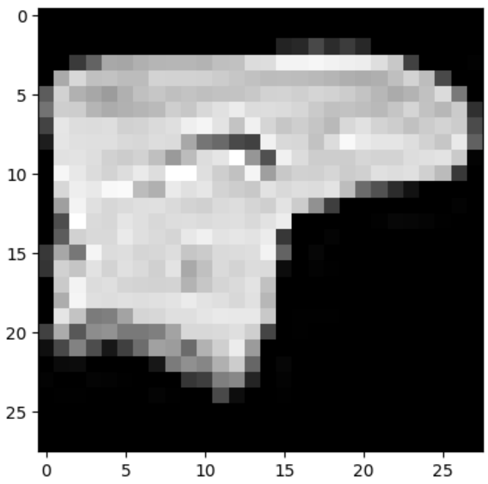
In [7]:
```
g = RotationGroupElement(2) # 180 degree rotation
plot_img(g(img))
```



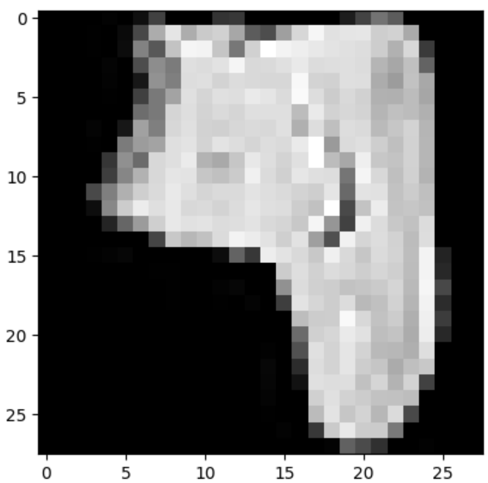
In [8]:
```
g = RotationGroupElement(1) # 90 degree rotation clockwise
plot_img(g(img))
```



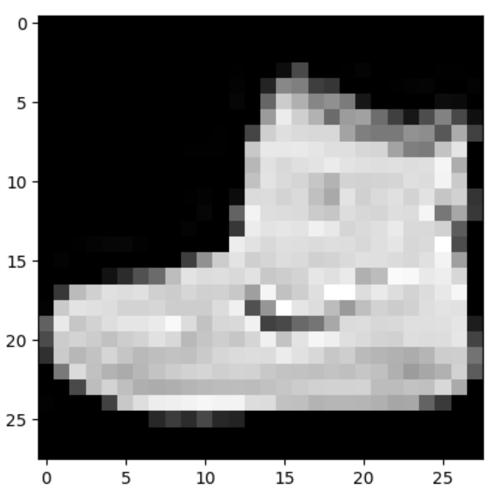
In [9]:
```
plot_img((g*g)(img)) # group product (composition) -> 180 degrees
```



In [10]:
```
plot_img((~g)(img)) # inverse -> rotate 90 degrees counterclockwise
```



In [11]:
```
plot_img((~g*g)(img)) # inverse times original -> identity
```



### (b) Implement the lifting convolution, including:

```
(i) lifting kernel with:
    input parameters size of the kernel
    input and output channel
    etc.
(ii) forward function.
```

In [12]:
```
from torch import nn, optim
from torch import Tensor
import torch.nn.functional as F
```

In [13]:

```python
class LiftingConv(nn.Module):
    def __init__(self,
        in_channels: int,
        out_channels: int,
        kernel_size: int,
        stride: int = 1,
        padding: float = 1
        ):
        """
        input shape:
            (N, Ci, Hi, Wi )
            N = batch size
            Ci = input channels
            Hi = input height
            Wi = input Width

        output shape:
            (N, Co, G, Ho, Wo )
            N = batch size
            Co = output channels
            G = group order
            Ho = output height
            Wo = output Width
        """
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

        self.weight = torch.nn.Parameter(torch.randn((out_channels, in_channels, kernel_size, kernel_size)) )


    def forward(self, input: Tensor):

        # convolve kernel with input under all group transformations:
        output = []
        for g in [RotationGroupElement(i) for i in [0,1,2,3]]:
            output.append(
                F.conv2d(input=input,
                         weight=g(self.weight),
                         stride=self.stride,
                         padding=self.padding))

        output = torch.stack(output)

        # reshape:
        # (group_order, batch_size, out_channels, H, W) ->
        # (batch_size, out_channels, group_order, H, W)
        output = output.permute(1,2,0,3,4)

        return output
```

## Applying the lifting convolution using a sobel filter to visualize the filter rotations:

In [14]:

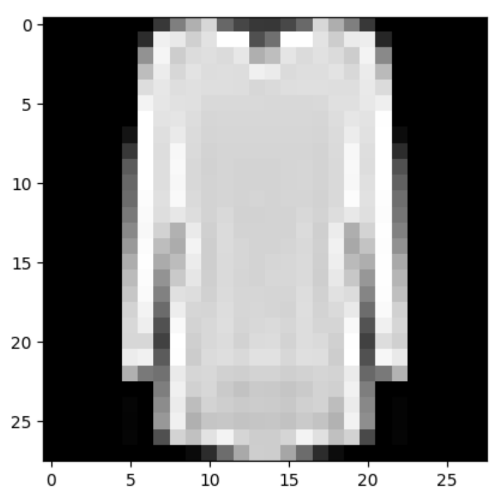
```python
lc = LiftingConv(1, 1, 3, 1, 1)
lc.weight = torch.nn.Parameter(torch.tensor(
    [[[[1,2,1],
       [0,0,0],
       [-1,-2,-1]
    ]]], dtype=torch.float32
))
```

In [15]:

```python
x,y = next(iter(train_loader))
```

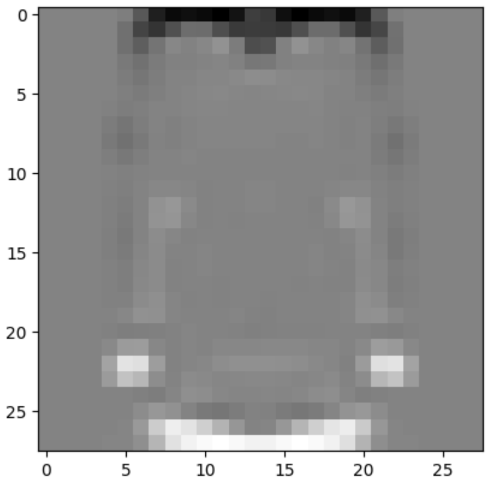
In [16]:

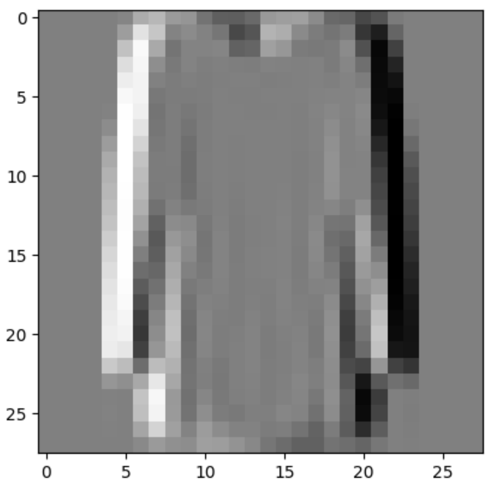
```python
plot_img(x[0,0,:,:])
```

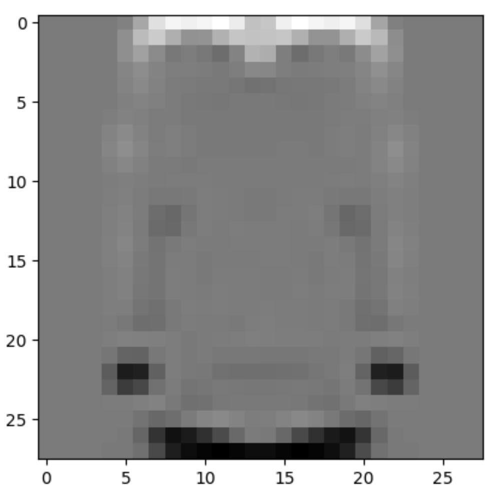


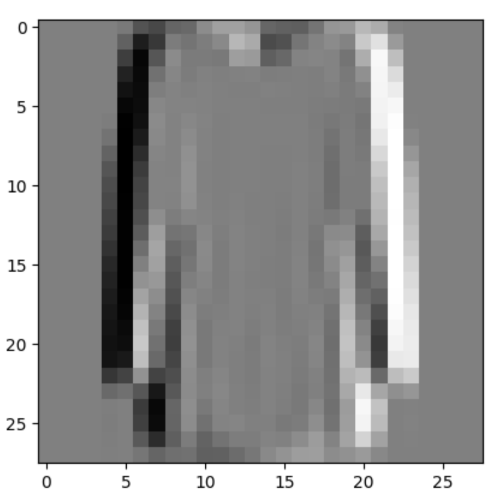
In [17]:

```
plot_img(lc(x)[0,0,0,:,:].detach())
plot_img(lc(x)[0,0,1,:,:].detach())
plot_img(lc(x)[0,0,2,:,:].detach())
plot_img(lc(x)[0,0,3,:,:].detach())
```


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25

### (c) Implement the group convolution, including:

```
(i) group convolution kernel (with input parameters size of the kernel, input and output channel, etc. )
(ii) the transformation of the group action
(iii) bilinear interpolation (sampling)
(iv) forward function.
(Note that other weight initializations and interpolations are welcome to test/include.)
```

In [18]:
```python
class GroupConv(nn.Module):
    def __init__(self,
        in_channels: int,
        out_channels: int,
        kernel_size: int = 3,
        stride: int = 1,
        padding: float = 1,
        group_order: int = 4
        ):
        """
        input shape:
            (N, Ci, Hi, Wi )
            N = batch size
            Ci = input channels
            G = group_order
            Hi = input height
            Wi = input Width

        output shape:
            (N, Co, G, Ho, Wo )
            N = batch size
            Co = output channels
            G = group order
            Ho = output height
            Wo = output Width
        """
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

        self.weight = torch.nn.Parameter(torch.randn((out_channels, in_channels * group_order, kernel_size, kernel_size)) )


    def forward(self, input: Tensor):
        # swap axes so all channels of the same group index appear consecutively
        # this allows easy shifting of the convolved filter
        # (batch_size, in_channels, group_order, H, W) ->
        # (batch_size, group_order, in_channels, H, W)
        input = input.swapaxes(1,2)

        # reshape input:
        batch_size, group_order, in_channels, H, W = input.shape
        flattened_input = input.reshape(batch_size, in_channels * group_order, H, W)

        # convolve kernel
        output = []
        for i, g in [(i,RotationGroupElement(i)) for i in [0,1,2,3]]:
            output.append(
                F.conv2d(input=flattened_input,
                         weight=torch.roll(g(self.weight), shifts=i*in_channels, dims=1),
                         stride=self.stride,
                         padding=self.padding))

        output = torch.stack(output)

        # (group_order, batch_size, out_channels, H, W) ->
        # (batch_size, out_channels, group_order, H, W)
        output = output.permute(1,2,0,3,4)

        # return output
        return output
```

In [19]:
```python
lc_out_channels = 8
gc_in_channels = lc_out_channels

lc = LiftingConv(1, 8, 3, 1, 1)
lc(x).shape
```

Out[19]: `torch.Size([128, 8, 4, 28, 28])`

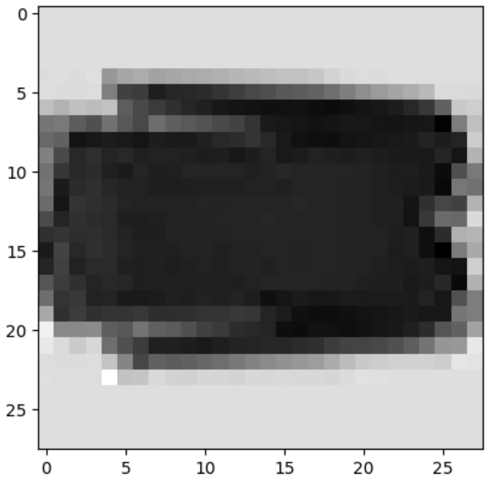
In [20]:
```python
gc = GroupConv(in_channels=gc_in_channels, out_channels=13)
gc.forward(lc(x)).shape
```

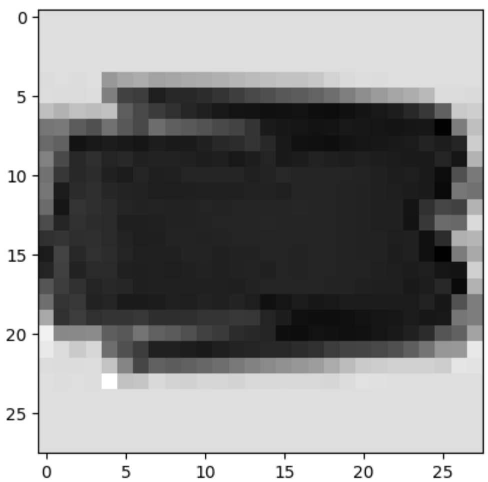
Out[20]: `torch.Size([128, 13, 4, 28, 28])`

### Lifting Convolution is Rotation Equivariant

In [21]:
```python
g = RotationGroupElement(1)
```

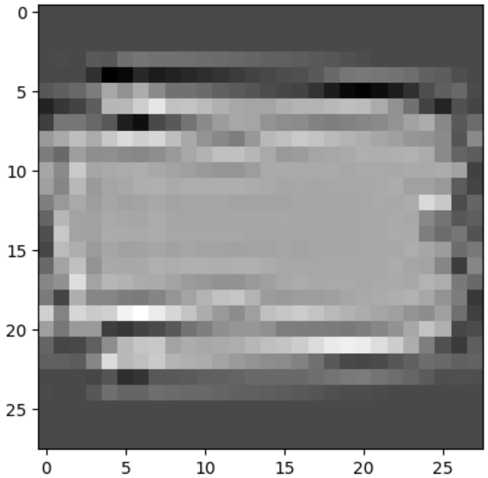
In [22]:
```
img1 = lc(g(x)).detach() # apply filter to rotated image
img2 = torch.roll(g(lc(x)).detach(), shifts=1, dims=2) # apply filter to original image, and rotate output (i.e rotate and shift by 1)


random_idx = np.random.choice([0,1,2,3])
plot_img(img1[0,0,random_idx,:,:])
plot_img(img2[0,0,random_idx,:,:])
```

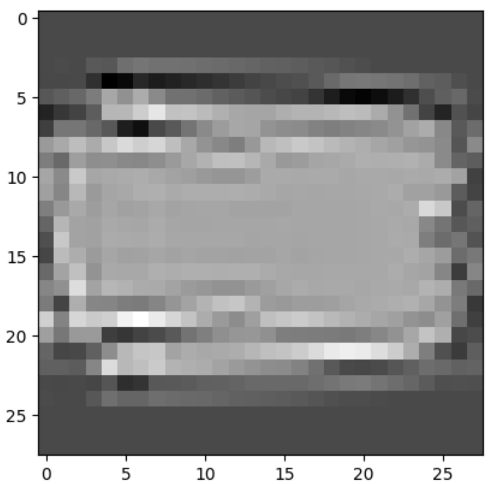



## Group Convolution is rotation equivariant:

In [23]:
```
img1 = gc(lc(g(x))).detach() # apply filter to rotated image
img2 = torch.roll(g(gc(lc(x))).detach(), shifts=1, dims=2) # apply filter to original image, and rotate output (i.e rotate and shift by 1)

random_idx = np.random.choice([0,1,2,3])
plot_img(img1[0,0,random_idx,:,:])
plot_img(img2[0,0,random_idx,:,:])
```




## (d) Implement a group convolution neural net

```
With:
    1. lifting convolution
    2. group convolution
    3. projection operation (using average-pooling or max-pooling over the group and spatial domain)

input parameters:
    1. size of the kernel
    2. input and output channel
    3. the number of hidden layers
    4. the hidden channel, etc.
```

In [24]:
```python
from functools import partial

class GroupConvNet(nn.Module):
    def __init__(
            self,
            in_channels: int = 1,
            hidden_channels: int = 15,
            out_channels: int = 10, # number of classes
            kernel_size: int = 3,
            n_hidden_layers: int = 5,
            pooling_op: Union[Literal['avg'], Literal['max']] = 'avg',
            kernel_padding: int = 1
    ) -> None:
        super().__init__()

        # lifting convolution:
        self.lc = LiftingConv(in_channels, hidden_channels, kernel_size, 1, kernel_padding)

        # group convolutions:
        self.gcs = torch.nn.ModuleList()
        for _ in range(n_hidden_layers):
            self.gcs.append(GroupConv(
                in_channels=hidden_channels,
                out_channels=hidden_channels,
                kernel_size=kernel_size,
                padding=kernel_padding
            ))

        self.fc = nn.Linear(hidden_channels, out_channels)

        if pooling_op == 'avg':
            self.pool = partial(torch.mean, dim=(-3, -2, -1))
        elif pooling_op == 'max':
            self.pool = partial(torch.amax, dim=(-3, -2, -1))
        elif pooling_op == 'sum':
            self.pool = partial(torch.sum, dim=(-3, -2, -1))
        else:
            raise ValueError(f'Pooling operation {pooling_op} not supported')


    def forward(self, x):

        x = self.lc(x)
        x = F.layer_norm(x, x.shape[-3:])
        x = F.relu(x)

        for gc in self.gcs:
            x = gc(x)
            x = F.layer_norm(x, x.shape[-3:])
            x = F.relu(x)

        x = self.pool(x)
        x = self.fc(x)

        return x
```

## (e) Experiment on the Fashion-MNIST dataset [1].

iv. Repeat steps (i)-(iii) but this time applying a random rotation [0, 2π] over the testing images. Compare the results and analyze how to improve t
he model for a random ro- tation [0, 2π].

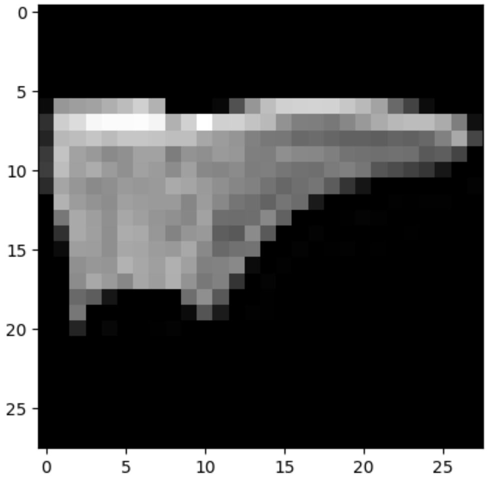
In [25]:
```python
import lightning.pytorch as pl
from torchmetrics import Accuracy

class LitGroupConvNet(pl.LightningModule):
    def __init__(self, net):
        super().__init__()
        self.net = net

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.net(x)

        loss = F.cross_entropy(y_hat, y)
        acc = torch.mean(y_hat.argmax(axis=1) == y, dtype=float)

        self.log_dict({
            "train_loss": loss,
            "train_accuracy": acc})

        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.net(x)

        loss = F.cross_entropy(y_hat, y)
        acc = torch.mean(y_hat.argmax(axis=1) == y, dtype=float)

        self.log_dict(
            {
                "val_loss": loss,
                "val_accuracy": acc
            },
            on_step=False,
            on_epoch=True)

        return loss

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(), lr=1e-2, weight_decay=1e-5)
        return optimizer
```

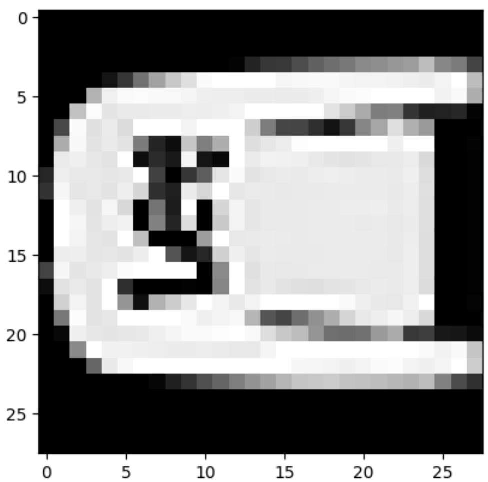
In [26]:
```python
from torch import utils
from lightning.pytorch.loggers import TensorBoardLogger

if False:
    net = GroupConvNet(hidden_channels=15, kernel_padding=0, pooling_op='max')
    pl_model = LitGroupConvNet(net)

    logger = TensorBoardLogger(
        "tb_logs",
        name="overfit_batch",
    )

    trainer = pl.Trainer(
        max_epochs=30,
        accelerator='gpu',
        logger=logger,
        log_every_n_steps=1,
        overfit_batches=1
    )

    trainer.fit(
        model=pl_model,
        train_dataloaders=train_loader
    )
```

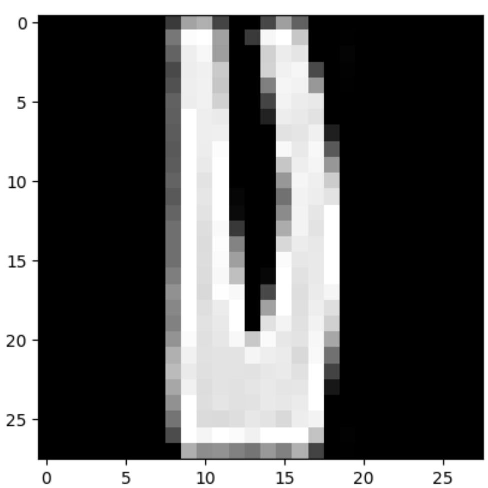
i. Splitting the dataset into training and testing sets(usetorchvision.datasets.FashionMNIST) with a batch size of 128.
For testing sets, apply a random rotation of the degree [0, 90, 180, 270] over the images.
Print a few examples from the testing set.

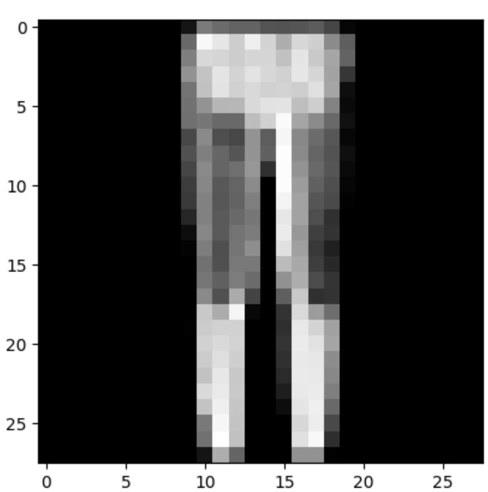
In [83]:
```python
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transforms.Compose([transforms.ToTensor
()]) )

# apply random 90 degree rotation to test images:
transform = transforms.Compose([
    transforms.ToTensor(),
    lambda x: RotationGroupElement(np.random.choice([0,1,2,3]))(x)
])
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train_size = int(0.95 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(train_dataset, [train_size, val_size])

num_workers=16
batch_size=128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)
```

In [84]:
```
x,y = next(iter(test_loader))

for i in range(4):
    plot_img(x[i,0,:,:])
```


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25

ii. Train the group convolution neural net with the following hyperparameters:

```
- input channel = 1,
- output channel = 10,
- kernel size = 3,
- hidden layer number = 5,
- hidden channel number = 16.
- For the optimizer, using Adam with the learning rate 10-2 and the weight decay 10-5.
```

In [28]:
```
if False:
    net = GroupConvNet(
        in_channels = 1,
        out_channels = 10,
        kernel_size = 3,
        n_hidden_layers = 5,
        hidden_channels=15,
        kernel_padding=0,
        pooling_op='sum'
    )

    pl_model = LitGroupConvNet(net)

    logger = TensorBoardLogger(
        "tb_logs",
        name="full_training",
    )

    trainer = pl.Trainer(
        max_epochs=150,
        accelerator='gpu',
        logger=logger,
        default_root_dir="./checkpoints/",
        use_distributed_sampler=False,
        devices=1
    )

    trainer.fit(
        model=pl_model,
        train_dataloaders = train_loader,
        val_dataloaders = val_loader
    )
```

iiii. Report the classification accuracy

In [31]:
```
CKPT_PATH = './tb_logs/full_training/version_4/checkpoints/epoch=149-step=66900.ckpt'

net = GroupConvNet(
    in_channels = 1,
    out_channels = 10,
    kernel_size = 3,
    n_hidden_layers = 5,
    hidden_channels=15,
    kernel_padding=0,
    pooling_op='sum'
)

model = LitGroupConvNet.load_from_checkpoint(CKPT_PATH, net=net)
model.eval()

pl.Trainer(accelerator='cpu').validate(dataloaders=test_loader, model=model)
```

```
GPU available: True (cuda), used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
Missing logger folder: /home/som/Geometric_DL_HW2/lightning_logs
```

| Validate metric | DataLoader 0 |
|---|---|
| val_accuracy | 0.8961 |
| val_loss | 0.9755975604057312 |

Out[31]:
```
[{'val_loss': 0.9755975604057312, 'val_accuracy': 0.8961}]
```

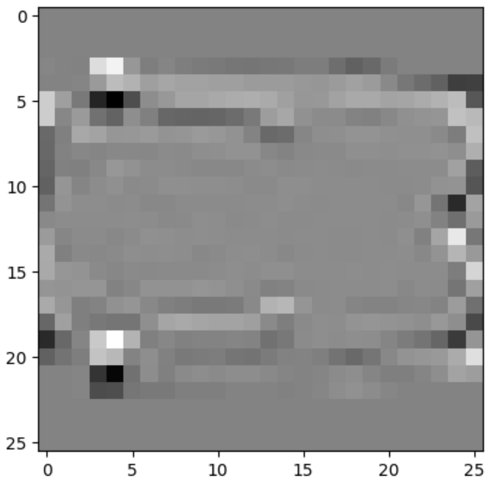
print the obtained features after each operation (lifting convolution, group convolution, and projection). Show that the final representations are indeed equivariant to 90-degree rotations.

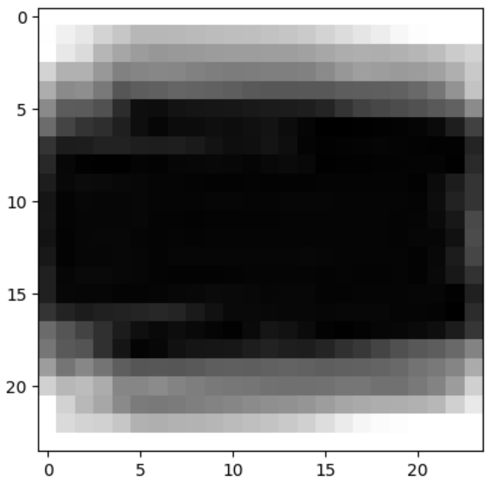
In [36]:
```
activation = {}
def get_activation(name):
    def hook(model, input, output):
        activation[name] = output.detach()
    return hook

net.lc.register_forward_hook(get_activation('lc'))
net.gcs[0].register_forward_hook(get_activation('gc1')) # first group conv
net.gcs[4].register_forward_hook(get_activation('gc5')) # 5th group conv
```

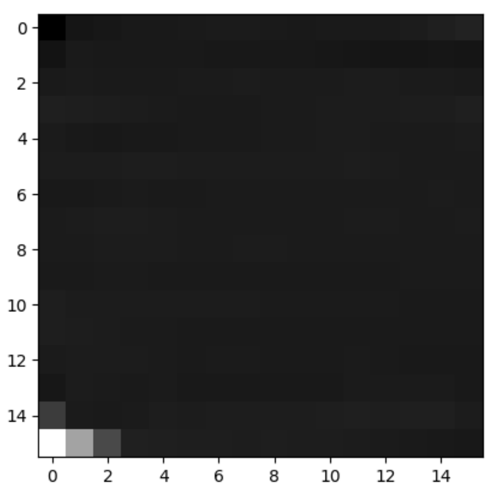
Out[36]:
```
<torch.utils.hooks.RemovableHandle at 0x7f991ad4efa0>
```

In [ ]:
```
g = RotationGroupElement(1)
```

In [56]:
```
output = net(g(x))
lc_output = activation['lc']
gc1_output = activation['gc1']
gc5_output = activation['gc5']

plot_img(lc_output[0,0,0,:,:])
plot_img(gc1_output[0,0,0,:,:])
plot_img(gc5_output[0,0,0,:,:])
```



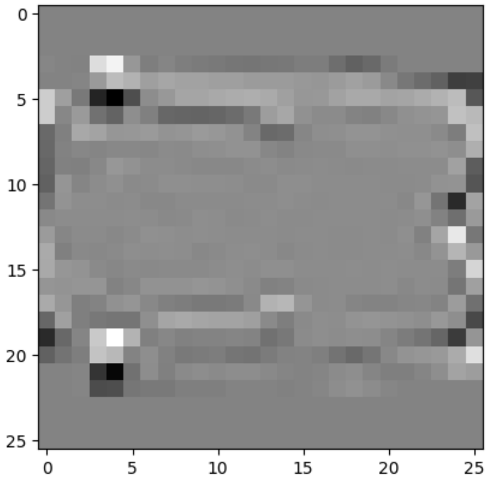



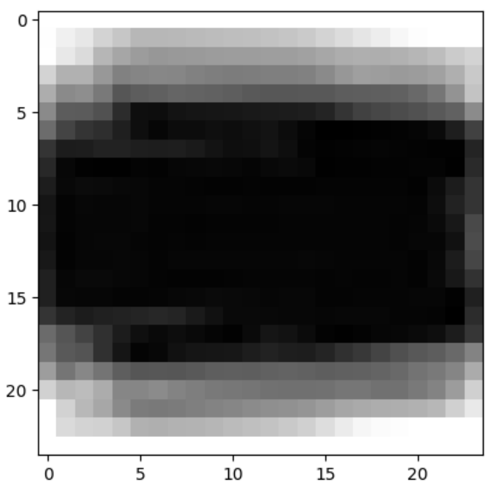



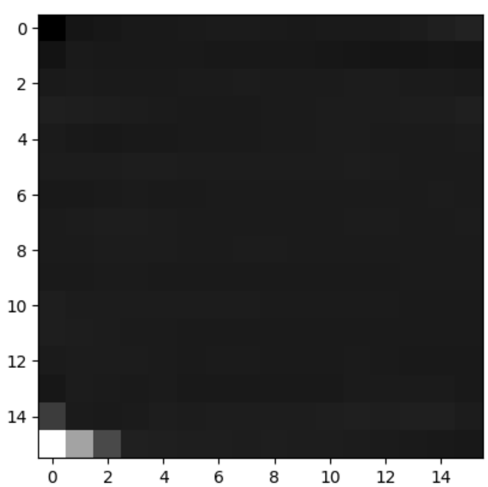

In [47]:
```
activation['lc'].shape
```

Out[47]:
```
torch.Size([128, 15, 4, 26, 26])
```

In [58]:
```
output = net(x)

# apply filter to original image, and rotate output (i.e rotate and shift by 1)
lc_output = torch.roll(g(activation['lc']), shifts=1, dims=2)
gc1_output = torch.roll(g(activation['gc1']), shifts=1, dims=2)
gc5_output = torch.roll(g(activation['gc5']), shifts=1, dims=2)

plot_img(lc_output[0,0,0,:,:])
plot_img(gc1_output[0,0,0,:,:])
plot_img(gc5_output[0,0,0,:,:])
```



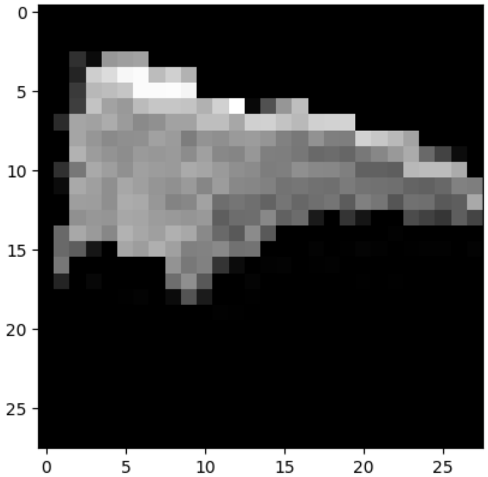


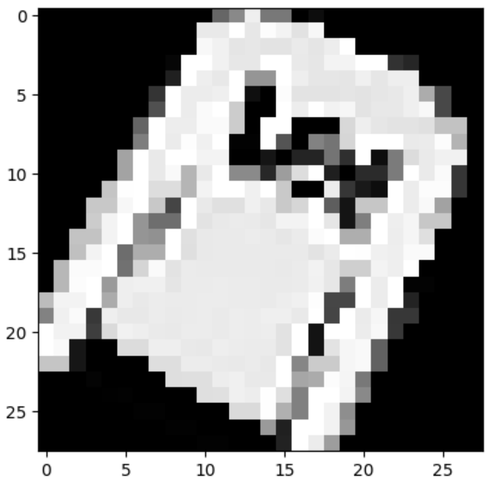


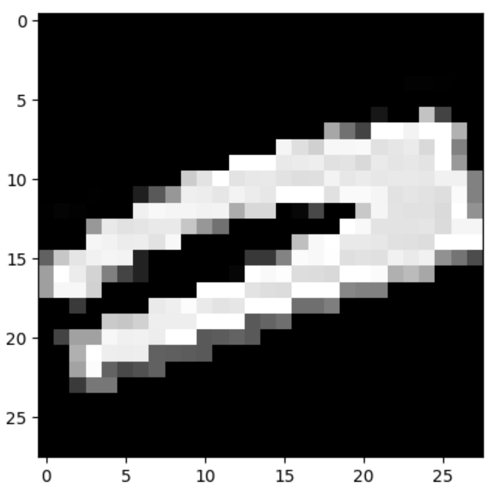
iv. Repeat steps (i)-(iii) but this time applying a random rotation [0, 2π] over the testing images. Compare the results and analyze how to improve the model for a random ro- tation [0, 2π].

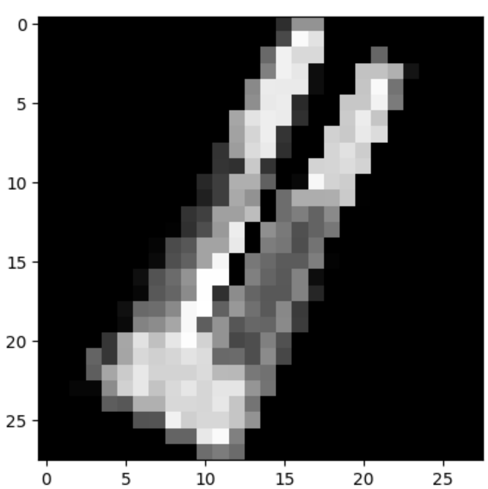
In [86]:
```
# apply random x degree rotation to test images:
transform = transforms.Compose([
    transforms.ToTensor(),
    torchvision.transforms.RandomRotation(180)
])
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)
```

In [87]:
```
x,y = next(iter(test_loader))

for i in range(4):
    plot_img(x[i,0,:,:])
```


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25

In [88]:
```
pl.Trainer(accelerator='cpu').validate(dataloaders=test_loader, model=model)
```

```
GPU available: True (cuda), used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
/home/som/Geometric_DL_HW2/.gdl/lib/python3.9/site-packages/lightning/pytorch/trainer/setup.py:176: PossibleUserWarning: GPU available but not used. Set `accelerator` and `devices` using `Trainer(accelerator='gpu', devices=4)`.
  rank_zero_warn(
```

| Validate metric | DataLoader 0 |
|---|---|
| val_accuracy | 0.4683 |
| val_loss | 17.98821449279785 |

Out[88]:
```
[{'val_loss': 17.98821449279785, 'val_accuracy': 0.4683}]
```

Strategies for dealing with random rotations:

1. Implementing a rotation group with a wider range of rotations, such as 10 degree rotations.
2. Augmenting the dataset with random rotations