Advanced Machine Learning - Exercise 03

In [27]:

```
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.optim as optim
from torch.utils.data import DataLoader

import torchvision.datasets as datasets
import torchvision.transforms as transforms

from torch.nn.functional import conv2d, max_pool2d, cross_entropy
```

In [28]:

```
plt.rc("figure", dpi=100)
```

1) Introduction

In [29]:

```
batch_size = 100

# transform images into normalized tensors
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])

train_dataset = datasets.MNIST(
    "./",
    download=True,
    train=True,
    transform=transform,
)

test_dataset = datasets.MNIST(
    "./",
    download=True,
    train=False,
    transform=transform,
)
```

In [30]:

```
train_dataloader = DataLoader(
    dataset=train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=1,
    pin_memory=True,
)

test_dataloader = DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=1,
    pin_memory=True,
)
```

In [31]:

```
def init_weights(shape):
    # Kaiming He initialization (a good initialization is important)
    # https://arxiv.org/abs/1502.01852
    std = np.sqrt(2. / shape[0])
    w = torch.randn(size=shape) * std
    w.requires_grad = True
    return w


def rectify(x):
    # Rectified Linear Unit (ReLU)
    return torch.max(torch.zeros_like(x), x)
```

In [32]:

```
class RMSprop(optim.Optimizer):
    """
    This is a reduced version of the PyTorch internal RMSprop optimizer
    It serves here as an example
    """
    def __init__(self, params, lr=1e-3, alpha=0.5, eps=1e-8):
        defaults = dict(lr=lr, alpha=alpha, eps=eps)
        super(RMSprop, self).__init__(params, defaults)

    def step(self):
        for group in self.param_groups:
            for p in group['params']:
                grad = p.grad.data
                state = self.state[p]

                # state initialization
                if len(state) == 0:
                    state['square_avg'] = torch.zeros_like(p.data)

                square_avg = state['square_avg']
                alpha = group['alpha']

                # update running averages
                square_avg.mul_(alpha).addcmul_(grad, grad, value=1 - alpha)
                avg = square_avg.sqrt().add_(group['eps'])

                # gradient update
                p.data.addcdiv_(grad, avg, value=-group['lr'])
```

In [33]:

```
# define the neural network
def model(x, w_h, w_h2, w_o):
    h = rectify(x @ w_h)
    h2 = rectify(h @ w_h2)
    pre_softmax = h2 @ w_o
    return pre_softmax
```

In [34]:

```
# initialize weights

# input shape is (B, 784)
w_h = init_weights((784, 625))
# hidden layer with 625 neurons
w_h2 = init_weights((625, 625))
# hidden layer with 625 neurons
w_o = init_weights((625, 10))
# output shape is (B, 10)

optimizer = RMSprop(params=[w_h, w_h2, w_o])
```

In [35]:

```
n_epochs = 100

train_loss = []
test_loss = []

# put this into a training loop over 100 epochs
for epoch in range(n_epochs + 1):
    train_loss_this_epoch = []
    for idx, batch in enumerate(train_dataloader):
        x, y = batch

        # our model requires flattened input
        x = x.reshape(batch_size, 784)
        # feed input through model
        noise_py_x = model(x, w_h, w_h2, w_o)

        # reset the gradient
        optimizer.zero_grad()

        # the cross-entropy loss function already contains the softmax
        loss = cross_entropy(noise_py_x, y, reduction="mean")

        train_loss_this_epoch.append(float(loss))

        # compute the gradient
        loss.backward()
        # update weights
        optimizer.step()

    train_loss.append(np.mean(train_loss_this_epoch))

    # test periodically
    if epoch % 10 == 0:
        print(f"Epoch: {epoch}")
        print(f"Mean Train Loss: {train_loss[-1]:.2e}")
        test_loss_this_epoch = []

        # no need to compute gradients for validation
        with torch.no_grad():
            for idx, batch in enumerate(test_dataloader):
                x, y = batch
                x = x.reshape(batch_size, 784)
                noise_py_x = model(x, w_h, w_h2, w_o)

                loss = cross_entropy(noise_py_x, y, reduction="mean")
                test_loss_this_epoch.append(float(loss))

        test_loss.append(np.mean(test_loss_this_epoch))

        print(f"Mean Test Loss:  {test_loss[-1]:.2e}")
```

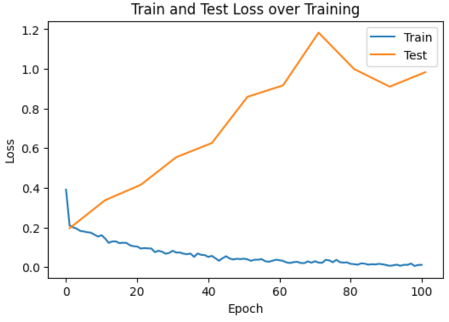
```
Epoch: 0
Mean Train Loss: 3.92e-01
Mean Test Loss:  1.96e-01
Epoch: 10
Mean Train Loss: 1.61e-01
Mean Test Loss:  3.38e-01
Epoch: 20
Mean Train Loss: 1.04e-01
Mean Test Loss:  4.15e-01
Epoch: 30
Mean Train Loss: 8.25e-02
Mean Test Loss:  5.55e-01
Epoch: 40
Mean Train Loss: 5.16e-02
Mean Test Loss:  6.26e-01
Epoch: 50
Mean Train Loss: 4.24e-02
Mean Test Loss:  8.59e-01
Epoch: 60
Mean Train Loss: 3.51e-02
Mean Test Loss:  9.17e-01
Epoch: 70
Mean Train Loss: 2.98e-02
Mean Test Loss:  1.18e+00
Epoch: 80
Mean Train Loss: 1.65e-02
Mean Test Loss:  9.99e-01
Epoch: 90
Mean Train Loss: 1.05e-02
Mean Test Loss:  9.11e-01
Epoch: 100
Mean Train Loss: 1.13e-02
Mean Test Loss:  9.83e-01
```

In [36]:

```
plt.plot(np.arange(n_epochs + 1), train_loss, label="Train")
plt.plot(np.arange(1, n_epochs + 2, 10), test_loss, label="Test")
plt.title("Train and Test Loss over Training")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[36]:

```
<matplotlib.legend.Legend at 0x7fb4b1cc3310>
```



2) Dropout

In [37]:

```
def dropout(x, p_drop=0.5):
    if not 0 <= p_drop < 1:
        raise ValueError(f"Dropout Probability must be between 0 and 1.")

    if p_drop > 0:
        phi = np.random.binomial(1, p_drop, x.shape).astype(np.float32)
        phi[phi == 0] = 1.0 / (1.0 - p_drop)
        phi[phi == 1] = 0
        phi = torch.from_numpy(phi)

        x = x * phi

    return x
```

or

In [38]:

```
def dropout(x, p_drop=0.5):
    if not 0 <= p_drop < 1:
        raise ValueError(f"Dropout Probability must be between 0 and 1.")

    if p_drop == 0.0:
        return x

    binomial_mask = torch.rand(x.size()) < p_drop
    x[binomial_mask] = 0

    return x / (1.0 - p_drop)
```

In [39]:

```
# define model with dropout
def dropout_model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(X @ w_h)
    h = dropout(h, p_drop_hidden)
    h2 = rectify(h @ w_h2)
    h2 = dropout(h2, p_drop_hidden)
    pre_softmax = h2 @ w_o
    return pre_softmax
```

In [40]:

```
# reinitialize weights
w_h = init_weights((784, 625))
w_h2 = init_weights((625, 625))
w_o = init_weights((625, 10))

# reinitialize optimizer
optimizer = RMSprop([w_h, w_h2, w_o])

# additional dropout parameters
p_drop_input = 0.2
p_drop_hidden = 0.2
```

In [41]:

```
n_epochs = 100

train_loss = []
test_loss = []

for epoch in range(n_epochs + 1):
    train_loss_this_epoch = []
    for idx, batch in enumerate(train_dataloader):
        x, y = batch

        x = x.reshape(batch_size, 784)
        noise_py_x = dropout_model(x, w_h, w_h2, w_o, p_drop_input, p_drop_hidden)

        optimizer.zero_grad()
        loss = cross_entropy(noise_py_x, y, reduction="mean")
        train_loss_this_epoch.append(float(loss))

        loss.backward()
        optimizer.step()

    train_loss.append(np.mean(train_loss_this_epoch))

    if epoch % 10 == 0:
        print(f"Epoch: {epoch}")
        print(f"Mean Train Loss: {train_loss[-1]:.2e}")
        test_loss_this_epoch = []

        with torch.no_grad():
            for idx, batch in enumerate(test_dataloader):
                x, y = batch
                x = x.reshape(batch_size, 784)

                # in testing, dropout is turned off
                noise_py_x = dropout_model(x, w_h, w_h2, w_o, 0.0, 0.0)

                loss = cross_entropy(noise_py_x, y, reduction="mean")
                test_loss_this_epoch.append(float(loss))

        test_loss.append(np.mean(test_loss_this_epoch))

        print(f"Mean Test Loss:  {test_loss[-1]:.2e}")
```

```
Epoch: 0
Mean Train Loss: 5.20e-01
Mean Test Loss:  2.08e-01
Epoch: 10
Mean Train Loss: 3.62e-01
Mean Test Loss:  1.47e-01
Epoch: 20
Mean Train Loss: 3.96e-01
Mean Test Loss:  2.12e-01
Epoch: 30
Mean Train Loss: 4.40e-01
Mean Test Loss:  2.94e-01
Epoch: 40
Mean Train Loss: 4.71e-01
Mean Test Loss:  2.35e-01
Epoch: 50
Mean Train Loss: 4.90e-01
Mean Test Loss:  2.98e-01
Epoch: 60
Mean Train Loss: 5.43e-01
Mean Test Loss:  3.02e-01
Epoch: 70
Mean Train Loss: 5.64e-01
Mean Test Loss:  3.81e-01
Epoch: 80
Mean Train Loss: 6.07e-01
Mean Test Loss:  3.86e-01
Epoch: 90
Mean Train Loss: 6.28e-01
Mean Test Loss:  4.04e-01
Epoch: 100
Mean Train Loss: 6.61e-01
Mean Test Loss:  4.02e-01
```

In [42]:

```
plt.plot(np.arange(n_epochs + 1), train_loss, label="Train")
plt.plot(np.arange(1, n_epochs + 2, 10), test_loss, label="Test")
plt.title("Train and Test Loss over Training")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[42]:

```
<matplotlib.legend.Legend at 0x7fb493a4bc40>
```

Question:

Dropout means that at training stage individual neurons are dropped out of the net with probability p, such that only a reduced network is left. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. Dropout is a regularization approach in neural networks which helps reduce interdependent learning amongst the neurons. At test-time, the dropout needs to be switched off (i.e., p_drop = 0) in order to test the full model.

The model without dropout overfits badly. The train error goes down towards zero while the test error goes up to one. The model with dropout still overfits, but much less.

3) Parametric ReLU

In [43]:

```
def PRelu(x, a):
    return torch.where(x > 0, x, a * x)

def init_alpha(shape):
    a = torch.zeros(size=shape)
    a.requires_grad = True

    return a
```

In [44]:

```
def prelu_model(X, w_h, w_h2, w_o, a, a2, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = PRelu(X @ w_h, a)
    h = dropout(h, p_drop_hidden)
    h2 = PRelu(h @ w_h2, a2)
    h2 = dropout(h2, p_drop_hidden)
    pre_softmax = h2 @ w_o
    return pre_softmax
```

In [45]:

```python
w_h = init_weights((784, 625))
w_h2 = init_weights((625, 625))
w_o = init_weights((625, 10))
a = init_alpha((batch_size, 625))
a2 = init_alpha((batch_size, 625))

optimizer = RMSprop([w_h, w_h2, w_o, a, a2])

p_drop_input = 0.2
p_drop_hidden = 0.2
```

In [46]:

```python
n_epochs = 100

train_loss = []
test_loss = []

for epoch in range(n_epochs + 1):
    train_loss_this_epoch = []
    for idx, batch in enumerate(train_dataloader):
        x, y = batch

        x = x.reshape(batch_size, 784)
        noise_py_x = prelu_model(x, w_h, w_h2, w_o, a, a2, p_drop_input, p_drop_hidden)

        optimizer.zero_grad()
        loss = cross_entropy(noise_py_x, y, reduction="mean")
        train_loss_this_epoch.append(float(loss))

        loss.backward()
        optimizer.step()

    train_loss.append(np.mean(train_loss_this_epoch))

    if epoch % 10 == 0:
        print(f"Epoch: {epoch}")
        print(f"Mean Train Loss: {train_loss[-1]:.2e}")
        test_loss_this_epoch = []

        with torch.no_grad():
            for idx, batch in enumerate(test_dataloader):
                x, y = batch
                x = x.reshape(batch_size, 784)

                noise_py_x = prelu_model(x, w_h, w_h2, w_o, a, a2, 0.0, 0.0)

                loss = cross_entropy(noise_py_x, y, reduction="mean")
                test_loss_this_epoch.append(float(loss))

        test_loss.append(np.mean(test_loss_this_epoch))

        print(f"Mean Test Loss:  {test_loss[-1]:.2e}")
```

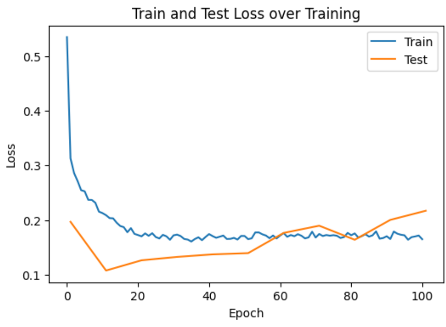
```
Epoch: 0
Mean Train Loss: 5.35e-01
Mean Test Loss:  1.97e-01
Epoch: 10
Mean Train Loss: 2.13e-01
Mean Test Loss:  1.07e-01
Epoch: 20
Mean Train Loss: 1.72e-01
Mean Test Loss:  1.26e-01
Epoch: 30
Mean Train Loss: 1.72e-01
Mean Test Loss:  1.32e-01
Epoch: 40
Mean Train Loss: 1.74e-01
Mean Test Loss:  1.37e-01
Epoch: 50
Mean Train Loss: 1.71e-01
Mean Test Loss:  1.39e-01
Epoch: 60
Mean Train Loss: 1.71e-01
Mean Test Loss:  1.76e-01
Epoch: 70
Mean Train Loss: 1.68e-01
Mean Test Loss:  1.89e-01
Epoch: 80
Mean Train Loss: 1.72e-01
Mean Test Loss:  1.64e-01
Epoch: 90
Mean Train Loss: 1.70e-01
Mean Test Loss:  2.00e-01
Epoch: 100
Mean Train Loss: 1.65e-01
Mean Test Loss:  2.17e-01
```

In [47]:

```python
plt.plot(np.arange(n_epochs + 1), train_loss, label="Train")
plt.plot(np.arange(1, n_epochs + 2, 10), test_loss, label="Test")
plt.title("Train and Test Loss over Training")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[47]:

```
<matplotlib.legend.Legend at 0x7fb493a4a080>
```



4) Convolutional Layers

4.1) Create a Convolutional Network

In [48]:

```python
def conv_model(X, conv_w1, conv_w2, conv_w3, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    conv1 = rectify(conv2d(X, conv_w1))
    pool1 = max_pool2d(conv1, (2, 2))
    drop1 = dropout(pool1, p_drop_hidden)

    conv2 = rectify(conv2d(drop1, conv_w2))
    pool2 = max_pool2d(conv2, (2, 2))
    drop2 = dropout(pool2, p_drop_hidden)

    conv3 = rectify(conv2d(drop2, conv_w3))
    pool3 = max_pool2d(conv3, (2, 2))
    drop3 = dropout(pool3, p_drop_hidden)

    h2 = rectify(torch.reshape(drop3, (batch_size, 128)) @ w_h2)
    pre_softmax = h2 @ w_o
    return pre_softmax
```

In [49]:

```python
conv_w1 = init_weights((32, 1, 5, 5))
conv_w2 = init_weights((64, 32, 5, 5))
conv_w3 = init_weights((128, 64, 2, 2))
w_h2 = init_weights((128, 625))
w_o = init_weights((625, 10))

optimizer = RMSprop([conv_w1, conv_w2, conv_w3, w_h2, w_o])

p_drop_input = 0.05
p_drop_hidden = 0.05
```

4.2) Application of Convolutional Network

In [50]:

```python
n_epochs = 20

train_loss = []
test_loss = []

for epoch in range(n_epochs + 1):
    train_loss_this_epoch = []
    for idx, batch in enumerate(train_dataloader):
        x, y = batch

        # convolutional model needs input (B, C, H, W)
        # B = Batch Size, C = Number of Channels, H, W = Image Dimensions
        x = x.reshape(-1, 1, 28, 28)
        noise_py_x = conv_model(x, conv_w1, conv_w2, conv_w3, w_h2, w_o, p_drop_input, p_drop_hidden)

        optimizer.zero_grad()
        loss = cross_entropy(noise_py_x, y, reduction="mean")
        train_loss_this_epoch.append(float(loss))

        loss.backward()
        optimizer.step()

    train_loss.append(np.mean(train_loss_this_epoch))

    if epoch % 2 == 0:
        print(f"Epoch: {epoch}")
        print(f"Mean Train Loss: {train_loss[-1]:.2e}")
        test_loss_this_epoch = []

        with torch.no_grad():
            for idx, batch in enumerate(test_dataloader):
                x, y = batch
                x = x.reshape(-1, 1, 28, 28)

                noise_py_x = conv_model(x, conv_w1, conv_w2, conv_w3, w_h2, w_o, 0.0, 0.0)

                loss = cross_entropy(noise_py_x, y, reduction="mean")
                test_loss_this_epoch.append(float(loss))

        test_loss.append(np.mean(test_loss_this_epoch))

        print(f"Mean Test Loss:  {test_loss[-1]:.2e}")
```

```
Epoch: 0
Mean Train Loss: 9.98e-01
Mean Test Loss:  1.39e-01
Epoch: 2
Mean Train Loss: 1.68e-01
Mean Test Loss:  8.85e-02
Epoch: 4
Mean Train Loss: 1.61e-01
Mean Test Loss:  8.45e-02
Epoch: 6
Mean Train Loss: 1.76e-01
Mean Test Loss:  7.43e-02
Epoch: 8
Mean Train Loss: 1.82e-01
Mean Test Loss:  8.31e-02
Epoch: 10
Mean Train Loss: 1.91e-01
Mean Test Loss:  9.02e-02
Epoch: 12
Mean Train Loss: 2.00e-01
Mean Test Loss:  1.03e-01
Epoch: 14
Mean Train Loss: 1.98e-01
Mean Test Loss:  9.12e-02
Epoch: 16
Mean Train Loss: 1.99e-01
Mean Test Loss:  1.15e-01
Epoch: 18
Mean Train Loss: 2.05e-01
Mean Test Loss:  8.99e-02
Epoch: 20
Mean Train Loss: 2.03e-01
```

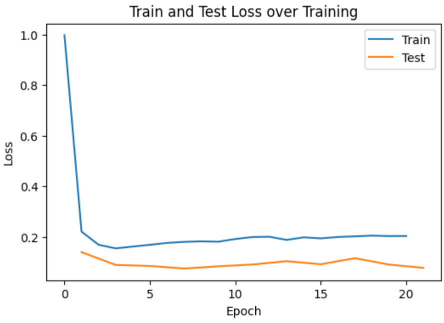
```
Mean Test Loss:  7.70e-02
```

In [51]:

```
plt.plot(np.arange(n_epochs + 1), train_loss, label="Train")
plt.plot(np.arange(1, n_epochs + 2, 2), test_loss, label="Test")
plt.title("Train and Test Loss over Training")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

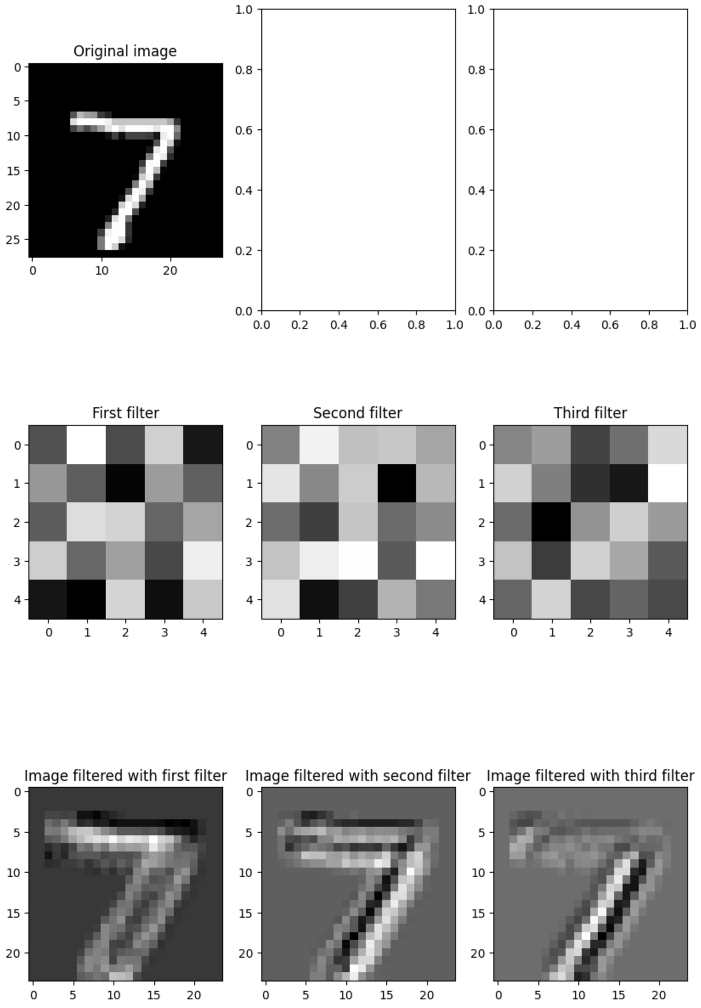
Out[51]:

```
<matplotlib.legend.Legend at 0x7fb493a88850>
```



In [57]:

```
# get next batch of images from testloader
(data, label) = next(iter(test_dataloader))

# take first image of batch and transform it to numpy
# for later visualization
img_numpy = data[0, 0, :, :].numpy()
# take three filters from first convolutional layer and transform
# them to numpy for later visualization
filter1_numpy = conv_w1[1, 0, :, :].detach().numpy()
filter2_numpy = conv_w1[2, 0, :, :].detach().numpy()
filter3_numpy = conv_w1[3, 0, :, :].detach().numpy()

# do convolution on batch with first layer
conv1 = conv2d(data, conv_w1)
# now take first image convoluted with the filters chosen earlier and
# transform them to numpy for visualiztaion
im_filter1 = conv1[0, 1, :, :].detach().numpy()
im_filter2 = conv1[0, 2, :, :].detach().numpy()
im_filter3 = conv1[0, 3, :, :].detach().numpy()

# do the visualization
fig, axes = plt.subplots(3, 3, figsize=(10, 16))
axes[0, 0].imshow(img_numpy, plt.cm.gray)
axes[0, 0].set_title("Original image")
axes[1, 0].imshow(filter1_numpy, plt.cm.gray)
axes[1, 0].set_title("First filter")
axes[1, 1].imshow(filter2_numpy, plt.cm.gray)
axes[1, 1].set_title("Second filter")
axes[1, 2].imshow(filter3_numpy, plt.cm.gray)
axes[1, 2].set_title("Third filter")
axes[2, 0].imshow(im_filter1, plt.cm.gray)
axes[2, 0].set_title("Image filtered with first filter")
axes[2, 1].imshow(im_filter2, plt.cm.gray)
axes[2, 1].set_title("Image filtered with second filter")
axes[2, 2].imshow(im_filter3, plt.cm.gray)
axes[2, 2].set_title("Image filtered with third filter")
plt.show()
```



In [53]:

```
def weird_conv_model(X, conv_w1, conv_w2, conv_w3, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    conv1 = rectify(conv2d(X, conv_w1))
    pool1 = max_pool2d(conv1, (2, 2))
    drop1 = dropout(pool1, p_drop_hidden)
    conv2 = rectify(conv2d(drop1, conv_w2))
    pool2 = max_pool2d(conv2, (2, 2))
    drop2 = dropout(pool2, p_drop_hidden)
    conv3 = rectify(conv2d(drop2, conv_w3))
    pool3 = max_pool2d(conv3, (2, 2))
    drop3 = dropout(pool3, p_drop_hidden)
    h2 = rectify(torch.reshape(drop3, (batch_size, 128)) @ w_h2)
    pre_softmax = h2 @ w_o
    return pre_softmax
```

In [54]:

```python
conv_w1_uniso = init_weights((32, 1, 5, 6))
conv_w2_uniso = init_weights((64, 32, 4, 5))
conv_w3_uniso = init_weights((128, 64, 2, 2))
w_h2_uniso = init_weights((128, 625))
w_o_uniso = init_weights((625, 10))

optimizer = RMSprop([conv_w1_uniso, conv_w2_uniso, conv_w3_uniso, w_h2_uniso, w_o_uniso])

p_drop_input = 0.05
p_drop_hidden = 0.05
```

In [55]:

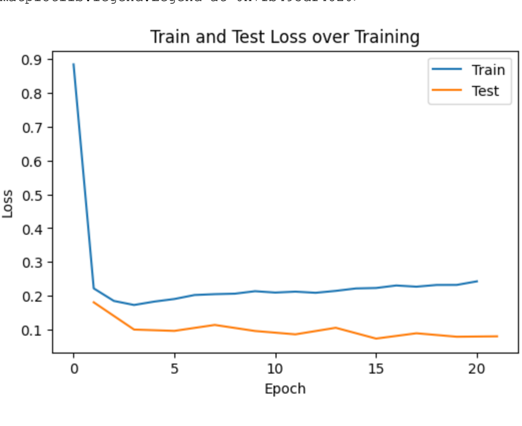
```python
n_epochs = 20

train_loss = []
test_loss = []

for epoch in range(n_epochs + 1):
    train_loss_this_epoch = []
    for idx, batch in enumerate(train_dataloader):
        x, y = batch

        # convolutional model needs input (B, C, H, W)
        # B = Batch Size, C = Number of Channels, H, W = Image Dimensions
        x = x.reshape(-1, 1, 28, 28)
        noise_py_x = weird_conv_model(x, conv_w1_uniso, conv_w2_uniso, conv_w3_uniso, w_h2_uniso, w_o_u
niso, p_drop_input, p_drop_hidden)

        optimizer.zero_grad()
        loss = cross_entropy(noise_py_x, y, reduction="mean")
        train_loss_this_epoch.append(float(loss))

        loss.backward()
        optimizer.step()

    train_loss.append(np.mean(train_loss_this_epoch))

    if epoch % 2 == 0:
        print(f"Epoch: {epoch}")
        print(f"Mean Train Loss: {train_loss[-1]:.2e}")
        test_loss_this_epoch = []

        with torch.no_grad():
            for idx, batch in enumerate(test_dataloader):
                x, y = batch
                x = x.reshape(-1, 1, 28, 28)

                noise_py_x = weird_conv_model(x, conv_w1_uniso, conv_w2_uniso, conv_w3_uniso, w_h2_unis
o, w_o_uniso, 0.0, 0.0)

                loss = cross_entropy(noise_py_x, y, reduction="mean")
                test_loss_this_epoch.append(float(loss))

        test_loss.append(np.mean(test_loss_this_epoch))

        print(f"Mean Test Loss:  {test_loss[-1]:.2e}")
```

```
Epoch: 0
Mean Train Loss: 8.85e-01
Mean Test Loss:  1.81e-01
Epoch: 2
Mean Train Loss: 1.85e-01
Mean Test Loss:  1.00e-01
Epoch: 4
Mean Train Loss: 1.83e-01
Mean Test Loss:  9.64e-02
Epoch: 6
Mean Train Loss: 2.03e-01
Mean Test Loss:  1.14e-01
Epoch: 8
Mean Train Loss: 2.06e-01
Mean Test Loss:  9.61e-02
Epoch: 10
Mean Train Loss: 2.10e-01
Mean Test Loss:  8.62e-02
Epoch: 12
Mean Train Loss: 2.09e-01
Mean Test Loss:  1.06e-01
Epoch: 14
Mean Train Loss: 2.22e-01
Mean Test Loss:  7.36e-02
Epoch: 16
Mean Train Loss: 2.31e-01
Mean Test Loss:  8.94e-02
Epoch: 18
Mean Train Loss: 2.32e-01
Mean Test Loss:  7.91e-02
Epoch: 20
Mean Train Loss: 2.43e-01
Mean Test Loss:  8.03e-02
```

In [56]:

```python
plt.plot(np.arange(n_epochs + 1), train_loss, label="Train")
plt.plot(np.arange(1, n_epochs + 2, 2), test_loss, label="Test")
plt.title("Train and Test Loss over Training")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[56]: `<matplotlib.legend.Legend at 0x7fb493df4820>`



The test loss of the convolutional network with the unisotropic filters is only slightly worse than the convoltional network with the isotropic filters. Since the unisotropic filters are not that weird in size it makes sense that the network is only slightly worse.