Task 1: Classification

1. define a Neural Network
2. define optimization procedure on FashionMNIST
3. train classifier on training set
4. evaluate model on test set
   - compute confusion matrix
   - compute Accuracy, Precision, Recall and F1 using the maximum response
   - visualize Precision-Recall curve for different classes
   - visualize example images with predicted classes

In [ ]:
```
import torch
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
from utils import NoisyFashionMNIST

%matplotlib inline
def show(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)), interpolation='nearest')
```

In [ ]:
```
from torch import nn
import torch.nn.functional as F
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_recall_curve
```

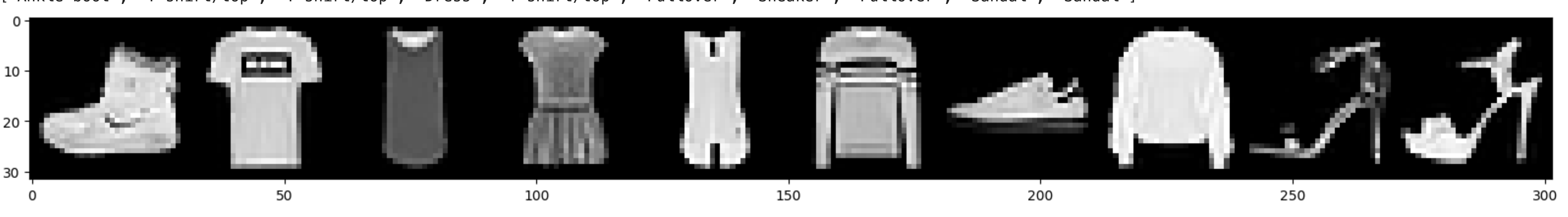
Dataset:

Downloads the FashionMNIST dataset in your local directory ./data
The following code shows how to access and visualize the data.

In [ ]:
```
transform=transforms.Compose([
        transforms.ToTensor()])

train_dataset = datasets.FashionMNIST("./data", train = True, download=True, transform=transform)
test_dataset = datasets.FashionMNIST("./data", train = False, download=True, transform=transform)
idx_to_class = {v: k for k, v in train_dataset.class_to_idx.items()} # .class_to_idx.items() is a dictionary that maps class labels to numerical indexes
```

In [ ]:
```
x = [train_dataset[i][0] for i in range(10)]
L = [idx_to_class[train_dataset[i][1]] for i in range(10)]
print(L)

plt.figure(figsize=(20,10))
show(torchvision.utils.make_grid(x, nrow=10))
plt.show()
```

```
['Ankle boot', 'T-shirt/top', 'T-shirt/top', 'Dress', 'T-shirt/top', 'Pullover', 'Sneaker', 'Pullover', 'Sandal', 'Sandal']
```



In [ ]:
```
# 0. trainloader, loading the data into batches of 64 images
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)
```

In [ ]:
```
# 1. Define a neural network
class FMNIST(nn.Module):
  def __init__(self):
    super().__init__()
    self.fc1 = nn.Linear(784, 128)
    self.fc2 = nn.Linear(128,64)
    self.fc3 = nn.Linear(64,10)

  def forward(self, x):
    x = x.view(x.shape[0], -1)

    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    x = F.log_softmax(x, dim=1)

    return x

model = FMNIST()
```

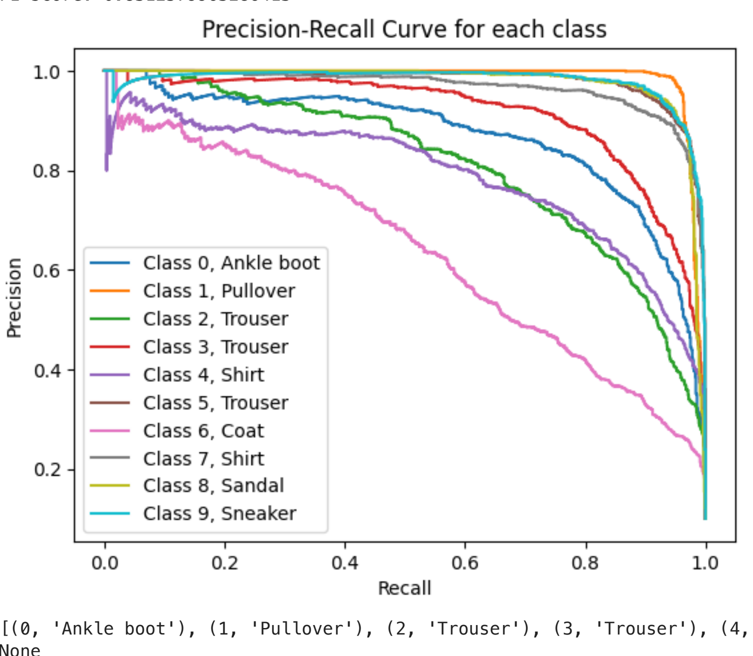
In [ ]:
```
# 2. Define an optimization procedure
from torch import optim

criterion = nn.NLLLoss()
# nn.NLLLoss() function expects the input to be logarithmized probabilities (logits) and the target to be class labels.
# It calculates the negative log-likelihood loss between the predicted probabilities and the true labels
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

In [ ]:
```
# 3. Train classifier on training set
num_epochs = 3

for i in range(num_epochs):
    cum_loss = 0

    for images, labels in train_loader :
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

        cum_loss += loss.item()

    print(f'Training Loss: {cum_loss/len(train_loader)}') # average loss per batch
```

```
Training Loss: 0.46412422120380503
Training Loss: 0.44977102495396315
Training Loss: 0.4382052550247229
```

In [ ]:
```
# 4. Evaluate the model on test set + add titles to cm

test_data = torch.utils.data.DataLoader(test_dataset, batch_size=10000, shuffle=False)
"""print(next(iter(test_data))[0])
print(next(iter(test_data))[1])"""

with torch.no_grad():
    data = next(iter(test_data))
    logps = model(data[0])
    output = torch.exp(logps)
    pred = torch.argmax(output,1)
    pred_array = pred.numpy()
    true_array = data[1].numpy()

    # 4.1 - compute confusion matrix

    cm = confusion_matrix(true_array, pred_array)
    print(cm)

    # 4.2 - compute Accuracy, Precision, Recall and F1 using the maximum response
    accuracy = accuracy_score(true_array, pred_array)
    precision = precision_score(true_array, pred_array, average='weighted') # for imbalanced class distributions. The weighted average takes into consideration the supp
    recall = recall_score(true_array, pred_array, average='weighted')
    f1 = f1_score(true_array, pred_array, average='weighted')
    print(f'Accuracy score: {accuracy}')
    print(f'Precision score: {precision}')
    print(f'Recall score: {recall}')
    print(f'F1 score: {f1}')

    # 4.3 - visualize Precision-Recall curve for different classes
    # A high area under the curve represents both high recall and high precision,
    # where high precision relates to a low false positive rate,
    # and high recall relates to a low false negative rate.
    true_labels = true_array
    predicted_probabilities = output.numpy()

    # Calculate precision and recall for each class
    precision = dict()
    recall = dict()
    for i in range(10):
        precision[i], recall[i], _ = precision_recall_curve(
            (true_labels == i).astype(int), predicted_probabilities[:, i])

    # Plot Precision-Recall curve for each class
    plt.figure()
    for i in range(10):
        plt.step(recall[i], precision[i], where='post', label=f'Class {i}, {L[i]}')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve for each class')
    plt.legend()
    plt.show()
    #print(print(list(zip(np.arange(10), L))))
```

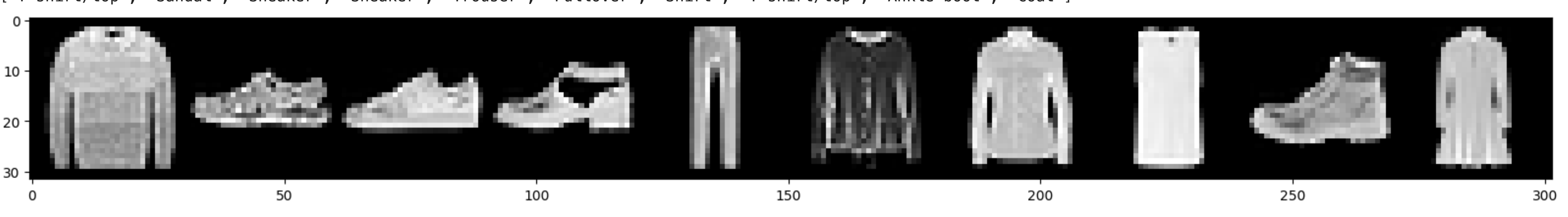
```
[[820   1   9  64   7   1  80   0  18   0]
 [  3 944  11  33   6   0   2   0   1   0]
 [ 19   2 634  15 215   2 102   0  11   0]
 [ 29  10   8 866  44   1  38   0   4   0]
 [  0   0  51  31 827   1  82   0   8   0]
 [  0   0   0   1   0 940   0  34   3  22]
 [163   1  81  55 141   2 522   0  35   0]
 [  0   0   0   0   0  54   0 897   0  49]
 [  1   1   9  11   4   6  15   5 948   0]
 [  0   0   0   0   0  25   0  38   1 936]]
Accuracy score: 0.8334
Precision score: 0.8339055447711818
Recall score: 0.8334
F1 score: 0.8312579963286413
```




```
[(0, 'Ankle boot'), (1, 'Pullover'), (2, 'Trouser'), (3, 'Trouser'), (4, 'Shirt'), (5, 'Trouser'), (6, 'Coat'), (7, 'Shirt'), (8, 'Sandal'), (9, 'Sneaker')]
None
```

In [ ]:
```
# 4.4 - visualize example images with predicted classes
first_batch = next(iter(test_loader))
print(type(first_batch))
images = first_batch[0][20:30]
labels = first_batch[1][20:30]

# list_pred_labels
logps = model(images)
ps = torch.exp(logps)
pred = torch.argmax(ps,1)
pred = pred.numpy()
print(pred)

true_labels = [idx_to_class[i] for i in labels.numpy()]
pred_labels = [idx_to_class[i] for i in pred]
print(true_labels)
print(pred_labels)

plt.figure(figsize=(20,10))
show(torchvision.utils.make_grid(images, nrow=10))
plt.show()
```

```
<class 'list'>
[0 5 7 7 1 2 6 0 9 4]
['Pullover', 'Sandal', 'Sneaker', 'Ankle boot', 'Trouser', 'Coat', 'Shirt', 'T-shirt/top', 'Ankle boot', 'Dress']
['T-shirt/top', 'Sandal', 'Sneaker', 'Sneaker', 'Trouser', 'Pullover', 'Shirt', 'T-shirt/top', 'Ankle boot', 'Coat']
```



Task 2: Image Denoising

1. define a Neural Network
2. define optimization procedure on NoisyFashionMNIST
3. train denoising model
4. Evaluate model

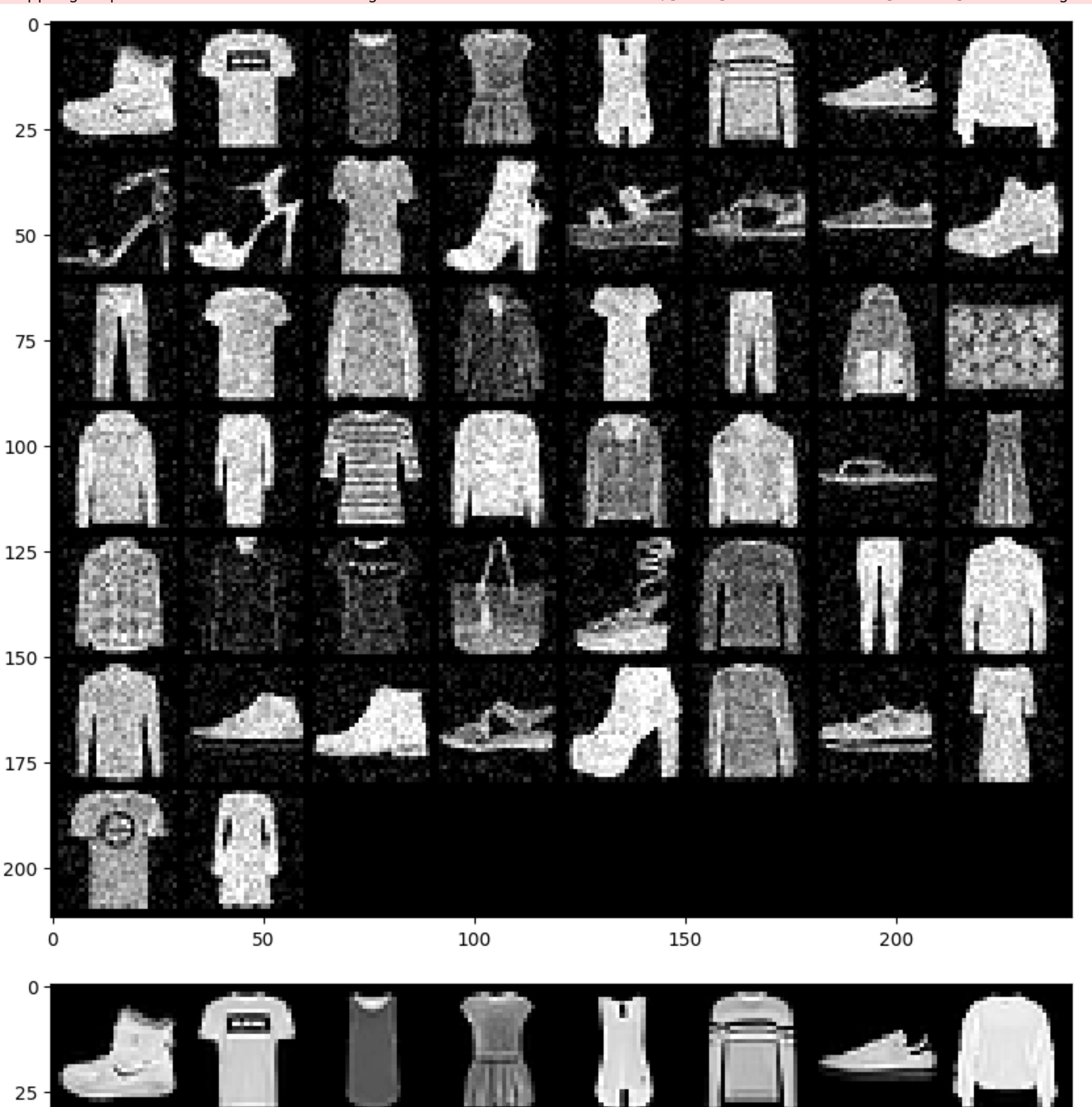
Dataset

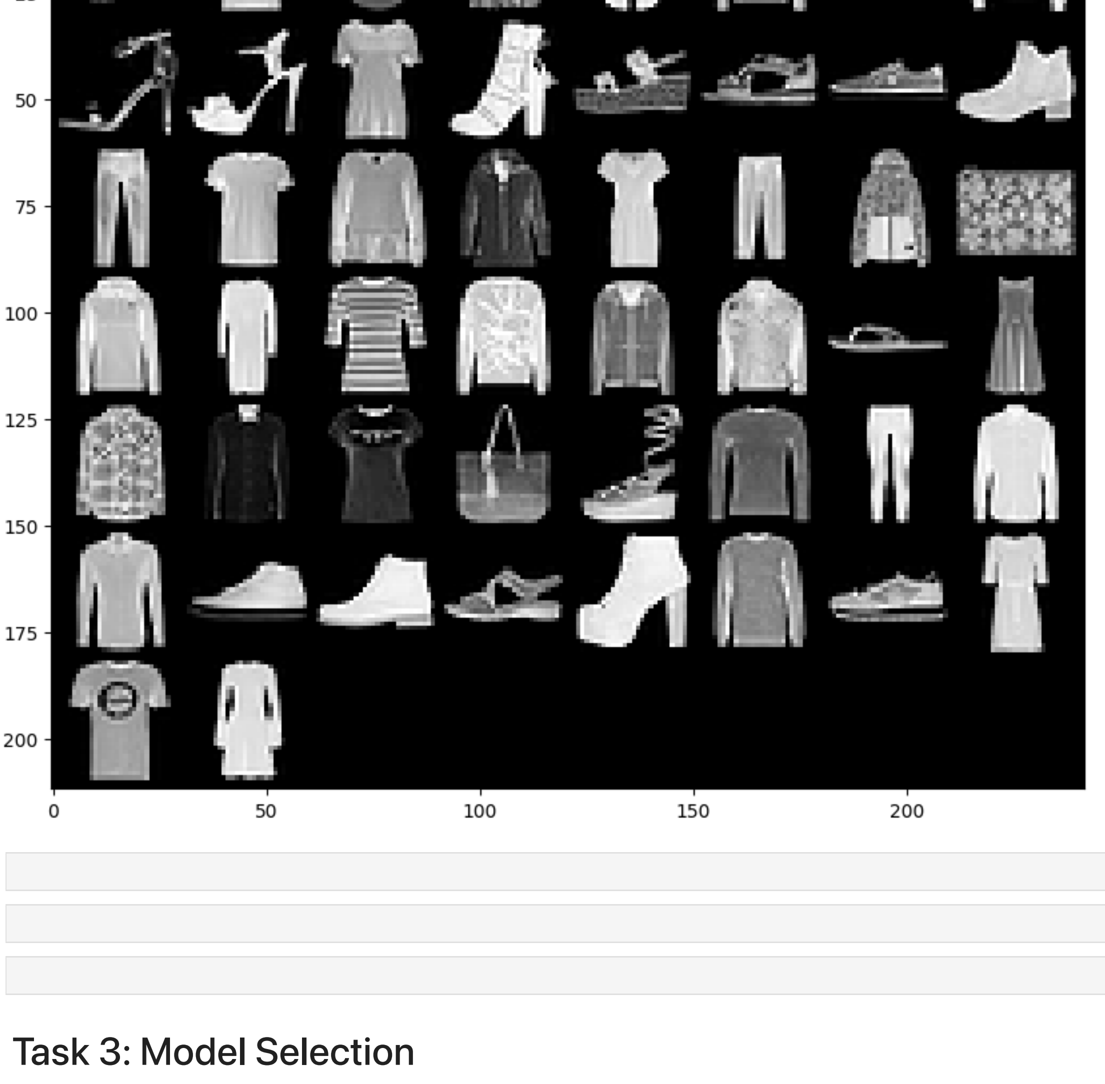
Random augmentations are added to the original dataset.

In [ ]:
```
train_dataset = NoisyFashionMNIST("./data", True)
test_dataset = NoisyFashionMNIST("./data", False)
```

In [ ]:
```
x = [train_dataset[i][0] for i in range(50)]
y = [train_dataset[i][1] for i in range(50)]

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(x))
plt.show()

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(y))
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```





In [ ]:

In [ ]:

In [ ]:

Task 3: Model Selection

Conduct **3 experiments** for each of the previous tasks and document them.
Evaluate the effect of different parameters on the classification and denoising tasks.
Conduct the following experiments:

- Evaluate the effect of residual connections
- Evaluate the effect of the depth(number of layers)/width (number of channels or number of neurons) of the network
- Evaluate the effect of Batch normalization

Optional experiments:

- How does the loss function affect denoising? Alternative loss functions: MSE, MAE, SSIM?
- How does Dropout affect the performance?
- Use different downsampling/upsampling layers, e.g.pooling, strided convolution, transposed convolution, etc.
- Feel free to explore more variations of your model and training.

In [ ]:

In [ ]:

In [ ]: